

WILLIAM STALLINGS

ORGANIZACJA I ARCHITEKTURA SYSTEMU KOMPUTEROWEGO

Projektowanie
systemu
a jego wydajność

Wydawnictwa Naukowo-Techniczne

WILLIAM STALLINGS

**ORGANIZACJA
I ARCHITEKTURA
SYSTEMU
KOMPUTEROWEGO**

Projektowanie
systemu
a jego wydajność

Wydawnictwa Naukowo-Techniczne • Warszawa

Dane ooryginale:

William Stallings

Computer Organization and Architecture

Design for Performance

Fourth Edition

Translation copyright © 2000

by Wydawnictwa Naukowo-Techniczne

Computer Organization and Architecture by William Stallings

Copyright © 1996

All Rights Reserved

Published by arrangement with the original publisher,
Prentice Hall, Inc., a Simon & Schuster Company

Redaktor *Izabela Ewa Mika*

Okladkę i strony tytułowe projektował *Andrzej Pilich*

Skład i łamanie *Krzysztof Niedziółka & Sp.*

Korekta *Zespół*

Podręcznik akademicki dotowany przez Ministerstwo Edukacji Narodowej

© Copyright for the Polish edition by Wydawnictwa Naukowo-Techniczne
Warszawa 2000

All Rights Reserved

Printed in Poland

Utwór w całości ani we fragmentach nie może być powielany ani rozpowszechniany
za pomocą urządzeń elektronicznych, mechanicznych, kopiujących, nagrywających i innych
bez pisemnej zgody posiadacza praw autorskich.

Adres poczty elektronicznej: wnt@pol.pl

Strona WWW: www.wnt.com.pl

ISBN 83-204-2517-4

Mojej kochającej żonie, Tricii

Spis treści

Przedmowa 11

CZĘŚĆ I PRZEGLĄD 17

Rozdział 1 Wstęp 19

1.1 Organizacja i architektura 19

1.2 Struktura i działanie 21

1.3 Zawartość książki 28

Rozdział 2 Ewolucja i wydajność komputerów 32

2.1 Krótka historia komputerów 32

2.2 Projektowanie zorientowane na wydajność 55

2.3 Ewolucja procesorów Pentium i PowerPC 60

2.4 Polecana literatura 64

2.5 Problemy do rozwiązania 64

CZĘŚĆ II SYSTEM KOMPUTEROWY 67

Rozdział 3 Magistrale systemowe 69

3.1 Zespoły komputera 69

3.2 Działanie komputera 72

3.3 Struktury połączeń 87

3.4 Połączenia magistralowe 89

3.5 Magistrala PCI 99

3.6 Magistrala Futurebus+ 109

3.7 Polecana literatura 121

3.8 Problemy do rozwiązania 121

Dodatek 3A Wykresy przebiegów czasowych 125

Rozdział 4 Pamięć wewnętrzna 127

4.1 Przegląd systemów pamięci komputerowych 127

4.2 Półprzewodnikowa pamięć główna 136

4.3 Pamięć podręczna 151

4.4 Złożona organizacja pamięci DRAM 174

- 4.5 Polecana literatura 180
- 4.6 Problemy do rozwiązania 181
- Dodatek 4A Charakterystyka wydajności pamięci dwupoziomowych 183

Rozdział 5 Pamięć zewnętrzna 191

- 5.1 Dysk magnetyczny 191
- 5.2 Pamięć RAID 197
- 5.3 Pamięć optyczna 207
- 5.4 Taśma magnetyczna 213
- 5.5 Polecana literatura 214
- 5.6 Problemy do rozwiązania 215

Rozdział 6 Wejście-wyjście 218

- 6.1 Urządzenia zewnętrzne 219
- 6.2 Moduły wejścia-wyjścia 224
- 6.3 Programowane wejście-wyjście 228
- 6.4 Wejście-wyjście sterowane przerwaniem 233
- 6.5 Bezpośredni dostęp do pamięci 243
- 6.6 Kanały i procesory wejścia-wyjścia 246
- 6.7 Interfejs zewnętrzny 248
- 6.8 Polecana literatura 265
- 6.9 Problemy do rozwiązania 265

Rozdział 7 Wspieranie systemu operacyjnego 269

- 7.1 Przegląd systemów operacyjnych 269
- 7.2 Szeregowanie czasowe 283
- 7.3 Zarządzanie pamięcią 290
- 7.4 Polecana literatura 314
- 7.5 Problemy do rozwiązania 315

CZĘŚĆ III JEDNOSTKA CENTRALNA

319

Rozdział 8 Arytmetyka komputera 321

- 8.1 Jednostka arytmetyczno-logiczna (ALU) 321
- 8.2 Reprezentacja liczb całkowitych 322
- 8.3 Arytmetyka liczb całkowitych 327
- 8.4 Reprezentacja zmiennopozycyjna 342
- 8.5 Arytmetyka zmiennopozycyjna 349
- 8.6 Polecana literatura 359
- 8.7 Problemy do rozwiązania 360
- Dodatek 8A Systemy liczbowe 362

Rozdział 9 Listy rozkazów: własności i funkcje 368

- 9.1 Własności rozkazów maszynowych 368
- 9.2 Rodzaje argumentów 376
- 9.3 Rodzaje operacji 381
- 9.4 Język assemblerowy 402

9.5	Polecana literatura	404
9.6	Problemy do rozwiązania	405
	Dodatek 9A Stosy	410
	Dodatek 9B Struktury „cienko-”, „grubo-” i „dwukońcowe”	416
Rozdział 10 Listy rozkazów: tryby adresowania i formaty		421
10.1	Adresowanie	421
10.2	Formaty rozkazów	436
10.3	Polecana literatura	449
10.4	Problemy do rozwiązania	449
Rozdział 11 Struktura i działanie jednostki centralnej		452
11.1	Organizacja procesora	452
11.2	Organizacja rejestrów	454
11.3	Cykl rozkazu	461
11.4	Potokowe przetwarzanie rozkazów	465
11.5	Procesor Pentium	479
11.6	Procesor Power PC	486
11.7	Polecana literatura	493
11.8	Problemy do rozwiązania	494
Rozdział 12 Komputery o zredukowanej liście rozkazów		497
12.1	Własności wykonywania rozkazów	498
12.2	Użycie dużej tablicy rejestrów	503
12.3	Optymalizacja rejestrów za pomocą kompilatora	509
12.4	Architektura o zredukowanej liście rozkazów	511
12.5	Przetwarzanie potokowe w architekturze RISC	519
12.6	Motorola 88000	523
12.7	MIPS R4000	529
12.8	Porównanie architektur RISC i CISC	537
12.9	Polecana literatura	539
12.10	Problemy do rozwiązania	539
Rozdział 13 Procesory superskalarne		544
13.1	Przegląd	544
13.2	Zagadnienia dotyczące projektowania	550
13.3	PowerPC	559
13.4	Pentium	567
13.5	Polecana literatura	572
13.6	Problemy do rozwiązania	573

CZĘŚĆ IV JEDNOSTKA STERUJĄCA

577

Rozdział 14 Działanie jednostki sterującej		579
14.1	Mikrooperacje	580
14.2	Sterowanie jednostką centralną	587
14.3	Rozwiązanie układowe	599

14.4	Polecana literatura	602
14.5	Problemy do rozwiązania	603
Rozdział 15 Sterowanie za pomocą mikroprogramu		604
15.1	Koncepcje podstawowe	605
15.2	Szeregowanie mikrorozkazów	614
15.3	Wykonywanie mikrorozkazów	620
15.4	TI 8800	633
15.5	Zastosowania mikroprogramowania	644
15.6	Polecana literatura	645
15.7	Problemy do rozwiązania	646

CZĘŚĆ V ORGANIZACJA RÓWNOLEGŁA 649

Rozdział 16 Przetwarzanie równoległe		651
16.1	Przetwarzanie wieloprocessorowe	652
16.2	Spójność pamięci podręcznych i protokół MESI	662
16.3	Obliczenia wektorowe	669
16.4	Processory równoległe	684
16.5	Polecana literatura	690
16.6	Problemy do rozwiązania	691

Dodatek A Cyfrowe układy logiczne		694
A.1	Algebra Boole'a	694
A.2	Bramki	696
A.3	Układy kombinacyjne	699
A.4	Układy sekwencyjne	722
A.5	Polecana literatura	733
A.6	Problemy do rozwiązania	733

Słownik	737
Literatura	748
Skorowidz	761

Przedmowa

Cele

Oto książka o strukturze i działaniu komputerów. Pisząc ją, chciałem możliwie jasno i wyczerpująco przedstawić naturę i własności nowoczesnych systemów komputerowych.

Z wielu powodów zadanie to jest wyzwaniem.

Po pierwsze, istnieje ogromna różnorodność wyrobów, które mogą nosić miano komputera - od pojedynczego mikroprocesora za kilka dolarów do superkomputerów kosztujących dziesiątki milionów dolarów. Różnorodność dotyczy nie tylko kosztów, ale również rozmiarów, wydajności i obszaru zastosowań. Po drugie, szybkie zmiany, które zawsze charakteryzowały technikę komputerową, zachodzą nadal i wcale nie słabną. Zmiany te obejmują wszystkie aspekty techniki komputerowej - od podstawowej technologii układów scalonych, służących do budowy zespołów komputera, do coraz większego wykorzystania koncepcji organizacji równoległej przy łączeniu tych zespołów.

Pomimo tej różnorodności i tempa zmian w dziedzinie komputerów pewne podstawowe rozwiązania pozostają nadal aktualne. Z pewnością jednak zastosowanie ich zależy od bieżącego stanu techniki oraz od założonej przez projektanta relacji między ceną a wydajnością. Intencją moją było dogłębne omówienie podstaw organizacji i architektury komputerów oraz ich powiązanie ze współczesnymi zagadnieniami projektowania.

Temat i podejście przyjęte w tej książce odzwierciedla podtytuł. Osiąganie dużej wydajności było zawsze ważne przy projektowaniu systemów komputerowych. Nigdy jednak wymaganie to nie było tak ważne i jednocześnie tak trudne do spełnienia jak dziś. Wartości wszystkich podstawowych parametrów związanych z wydajnością systemów komputerowych, w tym szybkości procesora, szybkości pamięci, pojemności pamięci oraz szybkości przepływu danych, stale szybko rosną. Ponadto ich wzrost następuje w różnym tempie. Utrudnia to zaprojektowanie zrównoważonego systemu, który maksymalnie wykorzystuje

wydajność wszystkich elementów. W tej sytuacji projektowanie komputera w coraz większym stopniu polega na zmianie struktury lub działania w jednym obszarze w celu skompensowania niezrównoważenia wydajności w innym. Tego rodzaju postępowanie będzie widoczne przy licznych zagadnieniach projektowania przedstawionych w tej książce.

System komputerowy, tak jak każdy system, składa się z powiązanego zbioru zespołów. Można go najlepiej scharakteryzować przez określenie jego struktury (czyli sposobu powiązania zespołów) i określenie jego funkcjonowania (czyli działania poszczególnych zespołów). Ponadto należy uwzględnić to, że struktura komputera jest hierarchiczna. Każdy główny zespół można następnie analizować dalej, rozkładając go na główne podzespoły i opisując z kolei ich strukturę i działanie. Uznałem, że bardziej przejrzyste i łatwiejsze do zrozumienia będzie przedstawienie tej hierarchicznej organizacji w sposób „od góry do dołu”.

- Q *System komputerowy.* Główne zespoły to procesor, pamięć i urządzenia wejścia-wyjścia.
- *Procesor.* Główne zespoły to jednostka sterująca, rejestry, jednostka arytmetyczno-logiczna i jednostka wykonująca rozkazy.
- a *Jednostka sterująca.* Główne zespoły to pamięć sterowania, zespół szeregowania mikrorozkazów i rejestry.

Bardzo zależało mi na jak najprostszym przedstawieniu nowego materiału. Sądzę, że przyjęte przeze mnie podejście „od góry do dołu” uchroni czytelnika od zagubienia się.

System jest rozpatrywany zarówno z punktu widzenia architektury (to znaczy atrybutów systemu widzialnych dla programującego w języku maszynowym), jak i organizacji (to znaczy jednostek operacyjnych i ich połączeń tworzących architekturę).

Przykładowe systemy

W celu wyjaśnienia przedstawionych koncepcji, a także potwierdzenia ich słuszności wykorzystałem przykłady pochodzące z dziesiątków różnych maszyn - głównie dwóch rodzin komputerów: Intel Pentium i PowerPC. Te właśnie projekty komputerów wyznaczają większość bieżących tendencji w projektowaniu. Pentium to w zasadzie komputery o złożonej liście rozkazów (CISC), podczas gdy PowerPC to - również w zasadzie - komputer o zredukowanej liście rozkazów (RISC). W obu systemach wykorzystano zasady projektowania super-skalarnego i oba mają konfigurację wieloprocessorową.

Plan książki

Książka jest podzielona na pięć części.

- I. Przegląd. Ta część zawiera przegląd zagadnień omawianych w książce.
- II. System komputerowy. System komputerowy składa się z procesora, pamięci i urządzeń wejścia-wyjścia oraz z połączeń między tymi głównymi zespołami. W tej części omówiłem kolejno każdy z tych zespołów z wyjątkiem procesora. Jest on na tyle złożony, że poświęciłem mu część III.
- III. Jednostka centralna. Jednostka centralna składa się z jednostki sterującej, rejestrów, jednostki arytmetyczno-logicznej, jednostki wykonywania rozkazów oraz połączeń między tymi zespołami. Opisałem zagadnienia dotyczące architektury, takie jak projektowanie list rozkazów i rodzaje danych, a także zagadnienia organizacyjne, takie jak kształtowanie potoków danych.
- IV. Jednostka sterująca. Jednostka sterująca jest tym elementem procesora, który uruchamia różne zespoły. Ta część książki dotyczy funkcjonowania jednostki sterującej i jej realizacji za pomocą mikroprogramowania.
- V. Organizacja równoległa. Ostatnią część poświęciłem na omówienie zagadnień dotyczących organizacji wieloprocesorowej i organizacji wykorzystującej przetwarzanie wektorowe.

Dokładniejsze streszczenie zawartości książki znajduje się na końcu rozdziału 1.

Usługi internetowe

Istnieje strona www związana z tą książką, do której mogą się odwołać studenci i wykładowcy. Strona ta zawiera odniesienia do odpowiednich lokalizacji, wzorce przezroczy rysunków w formacie PDF (Adobe Acrobat) i informacje wymagane do skorzystania z internetowej listy pocztowej związanej z książką. Lista została zestawiona w sposób umożliwiający wykładowcom wymianę informacji, sugestii i pytań pomiędzy sobą i z autorem. Adres strony jest następujący:

<http://www.shore.net/~ws/COA4e.html>.

Errata odnosząca się do tej książki jest osiągalna na mojej stronie www pod adresem:

<http://www.shore.net/~ws/welcome.html>.

Co nowego w czwartym wydaniu

W ciągu trzech lat, które upłynęły od trzeciego wydania tej książki, wprowadzono w rozpatrywanej dziedzinie liczne innowacje i ulepszenia. W nowym czwartym wydaniu starałem się je uchwycić. Uwagi, które przekazali mi czytelnicy trzeciego wydania, istotnie wpłynęły na układ zamieszczonego tu materiału.

Najbardziej oczywiste zmiany w czwartym wydaniu to: zwiększony nacisk na wydajność (co zostało odzwierciedlone w nowym podtytule) oraz dokładniejsze omówienie mikroprocesorów. Ponieważ parametry charakteryzujące wydajność systemów komputerowych poprawiają się bardzo szybko, ale w różnym tempie, zadanie zaprojektowania systemu o zrównoważonych i zoptymalizowanych własnościach stawało się z każdym minionym rokiem coraz trudniejsze. Wpłynęło to na sposób projektowania magistrali, organizację i protokoły pamięci podręcznych, architekturę list rozkazów, architekturę wejścia-wyjścia i inne. Doprowadziło również do zwiększenia nacisku na spotęgowanie strumienia rozkazów, innowacyjne struktury połączeń magistralowych oraz systemy wieloprocessorowe. Wszystko to zostało uwzględnione w prezentowanym tu materiale.

W książce skupiłem się na mikroprocesorach, co odzwierciedla ich wzrastającą dominację w przemyśle komputerowym i ważną rolę, jaką odgrywają one w niemal każdej innowacyjnej pracy z zakresu projektowania systemów komputerowych. Dwoma systemami, które zawierają wiele spośród współczesnych inicjatyw projektowych, są Pentium oraz PowerPC i dlatego one właśnie są wykorzystywane w książce jako podstawowe przykłady.

Zachowałem w przybliżeniu tę samą organizację rozdziałów, jednak znaczną część materiału zrewidowałem; wprowadziłem też nowy materiał. W niniejszym wydaniu książki zamieściłem dodatkowych 35 tablic i 66 rysunków; dodałem także 78 pozycji literatury i 25 problemów do rozwiązania przez czytelników.

Oto niektóre najbardziej znaczące zmiany.

- a Jako przykłady wykorzystałem specyfikacje dwóch nowych magistrali systemowych: PCI i Futurebus+. Są one być może dwoma najważniejszymi standardami magistrali systemowych, a ponieważ są całkowicie różne, obejmują większość ważnych koncepcji projektowania magistrali.
- a Materiał dotyczący pamięci podręcznych rozszerzyłem przez dołączenie rozważań na temat odrębnych pamięci rozkazów i danych oraz na temat dwóch poziomów pamięci podręcznych. Omówiłem też nowe ważne opracowania dotyczące organizacji pamięci DRAM.
- a Uwzględniłem technikę RAID.
- Q Przeanalizowałem szczegółowo specyfikacje ważnych magistrali - równoległej SCSI i szeregową P1394.
- a Przeredagowałem materiał dotyczący arytmetyki liczb całkowitych. Rozszerzyłem omówienie normy zmiennopozycyjnej IEEE 754.
- Q Rozpatrzyłem zagadnienie spójności pamięci podręcznych w systemach wieloprocessorowych, a także opisałem powszechnie stosowany protokół MESI.
- Q Omówiłem organizacje równoległe wykorzystujące wiele mikroprocesorów.

Podziękowania

Nowe wydanie zyskało, gdyż zostało przejrane przez wielu ludzi, którzy hojnie zaoferowali swój czas i doświadczenie. Wiele osób przeczytało poprzednie wydanie i przekazało mi konstruktywne uwagi, które uwzględniłem w nowym wydaniu. Pracę tę wykonali m.in.: Richard King z Oregon Institute of Technology, Ata Elahi z Southern Connecticut State University, a także J. Rotherman, R. Berrsford i L. Thornhill - wszyscy z University of Portsmouth w Anglii. G. Dattareya z University of Texas poczynił użyteczne spostrzeżenia dotyczące prezentowania pojęć arytmetyki liczb całkowitych. Marie Sullivan, Marvin Denman i Henry Warren - wszyscy z IBM - przejrzeni materiał dotyczący PowerPC. Doug Carmean i Theodore Omzigt z firmy Intel oraz Mike Schmit z Quantasm przeczytali fragmenty dotyczące Pentium. Rodney Van Meter dostarczył wielu interesujących zadań do rozwiązania. Na zakończenie chciałbym podziękować Nikitasowi Alexandridisowi za zgodę na wykorzystanie wybranych zadań z jego książki zatytułowanej *Design of Microprocessor-Based Systems*.

William Stallings

PRZEGLĄD

Część I stanowi przegląd; jest w niej przedstawiony kontekst reszty książki.

W rozdziale I jest wprowadzona koncepcja komputera jako systemu hierarchicznego. Komputer może być widziany jako struktura zespołów, a jego działanie - jako zbiorowe działanie współpracujących ze sobą zespołów. Z kolei każdy zespół może być opisywany poprzez swoją wewnętrzną strukturę i działanie. Przedstawimy główne poziomy tego hierarchicznego obrazu. Pozostała część książki jest zorganizowana stosownie do tych poziomów, poczynając od najwyższego.

Rozdział 2 zawiera krótką historię komputerów, poczynwszy od ich mechanicznych przodków aż do współczesnych systemów. Służy ona do pokazania pewnych ważnych zagadnień dotyczących projektowania komputerów, a także obrazu struktury komputera widzianego z najwyższego poziomu. W tym samym rozdziale jest następnie przedstawiony kluczowy temat tej książki: projektowanie zorientowane na wydajność. Jest też pokazane, jak ważne jest osiąganie równowagi, kiedy korzysta się z zespołów o znacznie różniących się wydajnościach.

Tematem tej książki jest struktura i działanie komputerów, celem zaś - możliwie jasne i kompletne przedstawienie natury i własności współczesnych systemów komputerowych. Z dwóch powodów zadanie to jest wyzwaniem.

Po pierwsze, istnieje ogromna różnorodność wyrobów, które mogą nosić miano komputera - od pojedynczego mikrokomputera kosztującego kilka dolarów do superkomputerów o wartości dziesiątków milionów dolarów. Różnorodność ta dotyczy nie tylko kosztu, ale także rozmiaru, wydajności i obszaru zastosowań. Po drugie, szybkie zmiany, które zawsze charakteryzowały technikę komputerową, zachodzą nadal i wcale nie słabną. Zmiany te obejmują wszystkie aspekty techniki komputerowej - od mającej podstawowe znaczenie technologii układów scalonych wykorzystywanej do budowy zespołów komputera, aż do coraz szerszego wykorzystania koncepcji organizacji równoległej przy łączeniu tych zespołów.

Pomimo tej różnorodności i tempa zmian w dziedzinie komputerów pewne podstawowe rozwiązania pozostają nadal aktualne. Z pewnością zastosowanie ich zależy od bieżącego stanu techniki oraz od założonej przez projektanta relacji między ceną a wydajnością, dlatego celem książki jest wnikliwe przedyskutowanie podstaw organizacji i architektury komputerów oraz powiązanie ich ze współczesnymi zagadnieniami projektowania komputerów. Rozdział ten ma charakter wprowadzenia i jest ujęty opisowo. Zawiera on przegląd pozostałej części książki.

1.1

ORGANIZACJA I ARCHITEKTURA

Przy opisywaniu systemów komputerowych często czynione jest rozróżnienie między *architekturą* komputera a jego *organizacją*. Chociaż precyzyjne zdefiniowanie tych pojęć jest trudne, jednak istnieje zgoda co do zagadnień, których dotyczy (patrz np. [VRAN80], [SIEW82] i [BELL78a]).

Architektura komputera odnosi się do tych atrybutów systemu, które są widzialne dla programisty. Innymi słowy, atrybuty te mają bezpośredni wpływ na logiczne wykonywanie programu. **Organizacja komputera** odnosi się do jednostek operacyjnych i ich połączeń, które stanowią realizację specyfikacji typu architektury. Przykładami atrybutów architektury są: lista rozkazów, liczba bitów wykorzystywanych do prezentacji różnych typów danych (np. liczb czy znaków), mechanizmy wejścia-wyjścia oraz metody adresowania pamięci. Do atrybutów organizacyjnych należą rozwiązania sprzętowe niewidzialne dla programisty, takie jak sygnały sterujące, interfejsy między komputerem a urządzeniami peryferyjnymi oraz wykorzystywana technologia pamięci.

Na przykład to, czy w komputerze występuje rozkaz mnożenia, jest zagadnieniem projektowania architektury. Zagadnieniem organizacyjnym jest natomiast to, czy ten rozkaz będzie wykonywany przez specjalną jednostkę mnożącą, czy też przez wielokrotne wykorzystanie jednostki sumującej systemu. Decyzja organizacyjna może wynikać ze spodziewanej częstości wykorzystywania rozkazu mnożenia, ze względnej szybkości obu rozwiązań, a także z kosztu i fizycznych rozmiarów specjalnej jednostki mnożącej.

Historycznie, a także współcześnie, rozróżnienie między architekturą a organizacją jest ważne. Wielu producentów komputerów oferuje rodziny modeli o tej samej architekturze, różniące się jednak organizacją. W wyniku tego poszczególne modele w tej samej rodzinie mają różne ceny i parametry określające wydajność. Ponadto, architektura może przeżyć wiele lat, natomiast organizacja zmienia się wraz z technologią. Wybitnym przykładem obu tych zjawisk jest architektura Systemu 370 IBM. Architektura ta była po raz pierwszy wprowadzona w roku 1970 i obejmowała wiele modeli. Klient o umiarkowanych wymaganiach mógł kupić model tańszy, lecz wolniejszy. Jeśli jego wymagania wzrosły, mógł sięgnąć po droższy i szybszy model, nie rezygnując jednakże z dotychczas opracowanego oprogramowania. Przez całe lata IBM wprowadzał wiele nowych modeli wykorzystujących ulepszoną technologię w celu zastąpienia starszych modeli, oferując klientowi większą szybkość, niższy koszt lub i jedno, i drugie. Nowe modele zachowywały tę samą architekturę, co chroniło inwestycje poniesione przez klienta na oprogramowanie. Jest godne uwagi, że architektura Systemu 370, nieznacznie ulepszona, przeżyła do dzisiaj i nadal występuje w głównej linii wyrobów IBM.

W klasie systemów nazywanych mikrokomputerami zależność między architekturą a organizacją jest bardzo ścisła. Zmiany technologii nie tylko wpływają na organizację, ale także umożliwiają budowanie komputerów o potężniejszej i bogatszej architekturze. W przypadku małych maszyn wymagania dotyczące kompatybilności między generacjami są mniejsze. Możliwości różnorodnego łączenia decyzji organizacyjnych i architektonicznych są zatem większe. Intrigującym tego przykładem jest komputer o zredukowanej liście rozkazów (RISC), który omówimy w rozdz. 12.

W książce tej jest rozpatrywana zarówno organizacja, jak i architektura komputerów. Być może większy nacisk jest położony na organizację. Ponieważ jednak organizacja komputera musi być zaprojektowana w celu wdrożenia określonej architektury, dokładne przeanalizowanie organizacji wymaga również szczegółowego zbadania architektury.

12

STRUKTURA I DZIAŁANIE

Komputer jest systemem złożonym; współczesne komputery zawierają miliony elementów elektronicznych. Jak więc można je prosto opisać? Kluczem jest rozpoznanie hierarchicznej struktury najbardziej złożonych systemów, w tym komputera [SIMO69]. System hierarchiczny jest układem wzajemnie powiązanych podsystemów, z których każdy również ma strukturę hierarchiczną, aż do osiągnięcia najniższego poziomu - podsystemu elementarnego.

Hierarchiczna struktura złożonych systemów ma podstawowe znaczenie zarówno dla projektowania, jak i opisu. W określonym momencie projektant operuje na jednym tylko szczególnym poziomie systemu. Na każdym poziomie system obejmuje zespół składników i ich wzajemne zależności. Na każdym poziomie zachowanie zależy jedynie od uproszczonych, abstrakcyjnych własności następnego niższego poziomu. Na każdym poziomie projektant zajmuje się strukturą i funkcjami składników systemu [KOES78].

a *Struktura* to sposób wzajemnego powiązania składników.

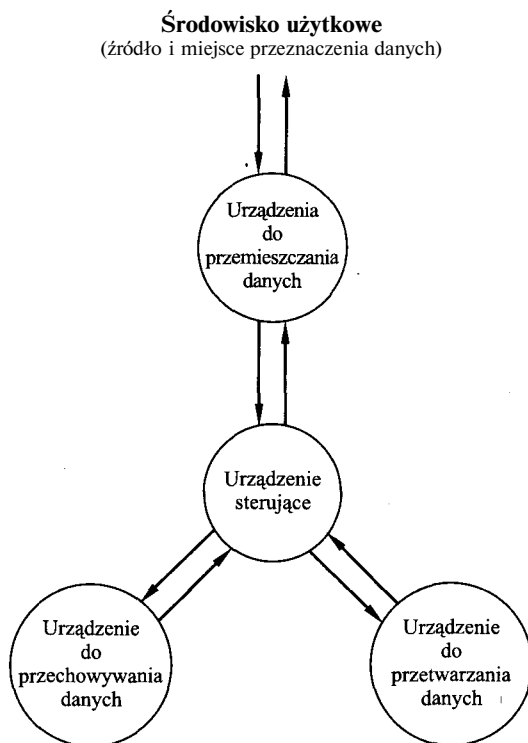
□ *Funkcje* określają działanie poszczególnych składników jako części struktury.

Istnieją dwie możliwości opisu systemu: zbudowanie kompletnego opisu poczynając od poziomu najniższego lub też rozpoczęcie od obrazu widzianego na poziomie najwyższym oraz dekompozycja systemu na części. Jest wiele powodów na to, że podejście „od góry do dołu” jest jaśniejsze i najbardziej efektywne [WEIN75].

Podejście przyjęte w tej książce odzwierciedla ten właśnie punkt widzenia. System komputerowy będzie opisany „od góry do dołu”. Rozpoczniemy od głównych składników systemu, opiszemy ich strukturę i funkcje, a następnie rozpatrzemy sukcesywnie niższe poziomy hierarchii. Pozostała część tego podrozdziału zawiera krótki opis tych zamierzeń.

Działanie

Zarówno struktura, jak i funkcjonowanie komputera są w zasadzie proste. Na rysunku 1.1 są pokazane podstawowe funkcje, które może realizować komputer. Ogólnie są to tylko cztery funkcje:



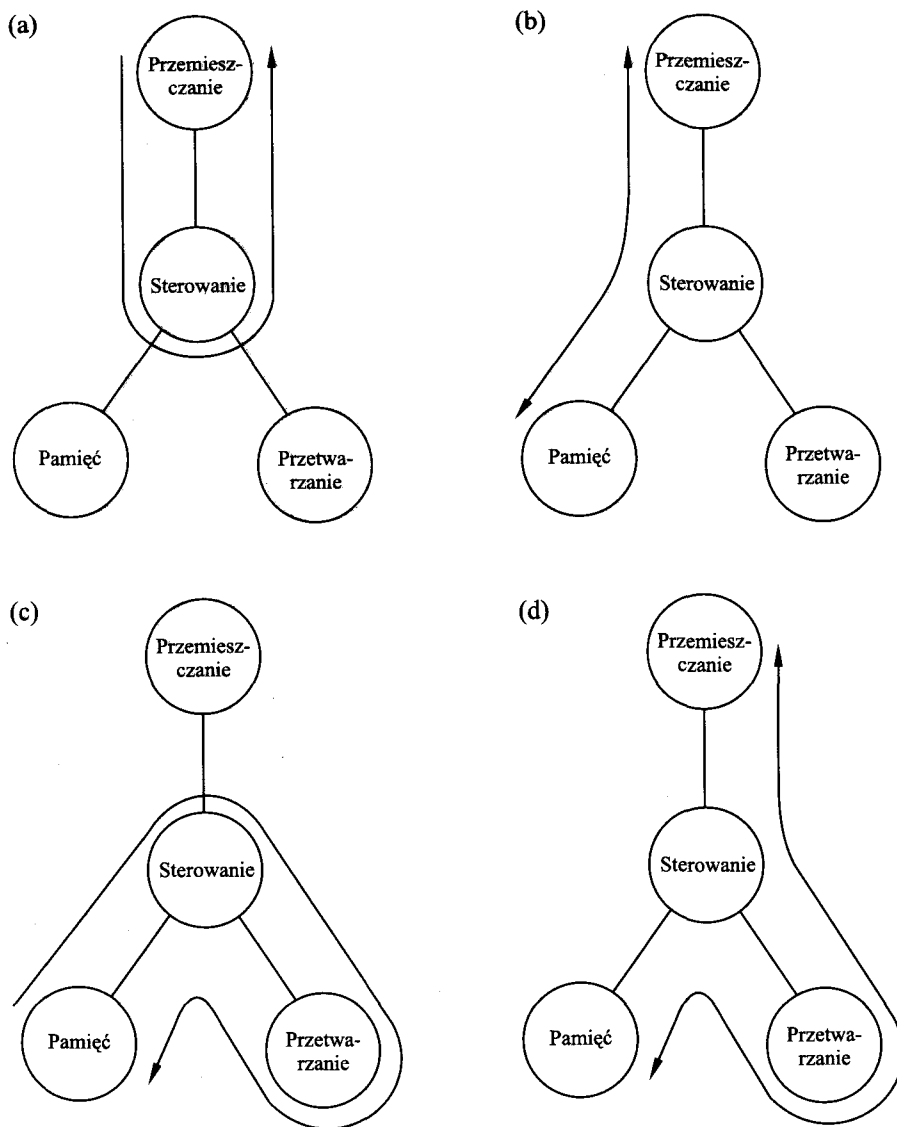
RYSUNEK 1.1. Obraz funkcjonalny komputera

- a przetwarzanie danych;
- a przechowywanie danych;
- a przenoszenie danych;
- a sterowanie.

Oczywiście komputer musi móc **przetwarzać dane**. Dane mogą przybierać różne formy, a zakres wymagań odnoszących się do przetwarzania jest szeroki. Zobaczymy jednak, że istnieje tylko niewiele podstawowych metod, lub typów, przetwarzania danych.

Jest również bardzo ważne, aby komputer **przechowywał dane**. Nawet jeśli komputer przetwarza dane jedynie „w locie” (tzn. dane **przychodzą**, są przetwarzane i natychmiast **wychodzą**), musi on czasowo przechowywać chociażby te dane, które w danym momencie są przetwarzane. Występuje więc przynajmniej funkcja krótkotrwałego przechowywania. Równie ważne jest, aby komputer realizował funkcję długotrwałego przechowywania danych. Pliki danych są przechowywane w komputerze, co umożliwia ich późniejsze pobieranie i aktualizację.

Komputer musi móc **przenosić dane** pomiędzy sobą a światem zewnętrznym. Otoczenie operacyjne komputera składa się z urządzeń, które są albo źródłami, albo odbiorcami danych. Jeśli dane są otrzymywane od urządzenia bez-



RYSUNEK 1.2. Operacje możliwe do wykonania przez komputer

pośrednio połączonego z komputerem lub do niego dostarczane, to taki proces jest określany jako **proces wejścia-wyjścia**, a samo urządzenie nazywa się **peryferyjnym**. Jeśli dane są przenoszone na większe odległości, do odległego urządzenia lub od niego, to proces taki jest określany jako **transmisja danych**.

Musi wreszcie istnieć możliwość **sterowania** tymi trzema funkcjami. W ostateczności sterowanie to jest wykonywane przez osoby, które wydają komputerowi polecenia. Wewnątrz systemu komputerowego jednostka sterująca **zarządza** zasobami komputera i koordynuje działanie jego składników funkcjonalnych, zależnie od wprowadzonych poleceń.

Na tym ogólnym poziomie dyskusji liczba możliwych do prowadzenia operacji jest niewielka. Na rysunku 1.2 są pokazane cztery możliwe rodzaje operacji. Komputer może funkcjonować jako urządzenie do przenoszenia danych (rys. 1.2a), przenosząc po prostu dane od jednego urządzenia peryferyjnego lub linii transmisyjnej do drugiego. Może również funkcjonować jako urządzenie do przechowywania danych (rys. 1.2b), przy czym dane są przenoszone z otoczenia do komputera (odczyt) i odwrotnie (zapis). Na dwóch ostatnich diagramach pokazano operacje obejmujące przetwarzanie danych - zarówno tych, które są przechowywane (rys. 1.2c), jak i będących w drodze między miejscem przechowywania a otoczeniem.

Dotychczasowa dyskusja może wyglądać na absurdalnie ogólną. Z pewnością jest możliwe, nawet na najwyższym poziomie struktury komputera, wyróżnienie wielu funkcji. Zacytujmy tu jednak [SIEW82]:

Jest godne uwagi, jak nieznacznie kształtuje się strukturę komputera zależnie od funkcji, która ma być wykonywana. U źródła tego zjawiska leży z natury ogólne przeznaczenie komputerów, przy czym cała specjalizacja funkcjonalna następuje w czasie programowania, a nie podczas projektowania.

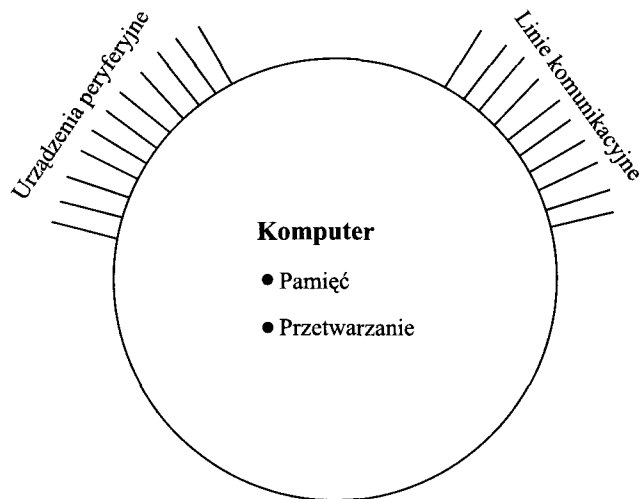
Struktura

Na rysunku 1.3 jest pokazany możliwie najprostszy obraz komputera. Komputer jest urządzeniem, które jest w pewien sposób powiązane ze swoim otoczeniem zewnętrznym. Ogólnie rzecz biorąc, wszystkie jego powiązania z otoczeniem mogą być sklasyfikowane jako urządzenia peryferyjne lub linie transmisyjne. Będziemy musieli bliżej przedstawić oba typy tych powiązań.

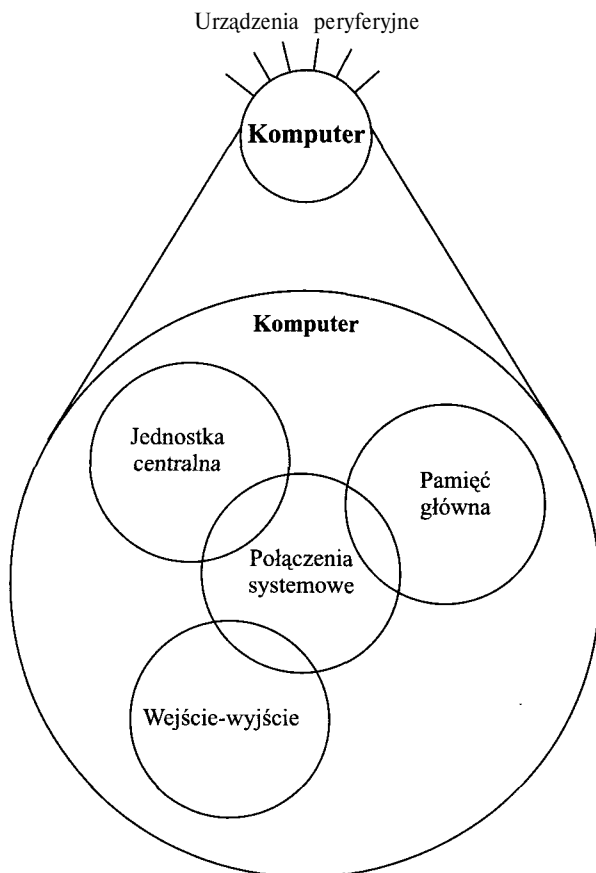
Jednak największą uwagę skupimy na wewnętrznej strukturze samego komputera, co zostało pokazane w górnej części rys. 1.4. Istnieją cztery główne składniki strukturalne komputera:

- a Jednostka centralna (CPU).** Steruje ona działaniem komputera i realizuje jego funkcję przetwarzania danych. Często jest po prostu nazywana procesorem.
- ☐ **Pamięć główna.** Przechowuje dane.
- Q Wejście-wyjście.** Przenosi dane między komputerem a jego otoczeniem zewnętrznym.
- ☐ **Połączenia systemu.** Mechanizmy zapewniające łączność między procesorem, pamięcią główną a wejściem-wyjściem.

Każdy z wymienionych składników może występować w komputerze pojedynczo lub w większej liczbie. Tradycyjnie występowała tylko jedna jednostka centralna. W ostatnich latach narastało wykorzystywanie wielu procesorów w pojedynczym systemie. Pewne problemy projektowania odnoszące się do systemów wieloprocessorowych są przedyskutowane w dalszej części książki. W rozdziale 16 skupimy się na takich systemach.



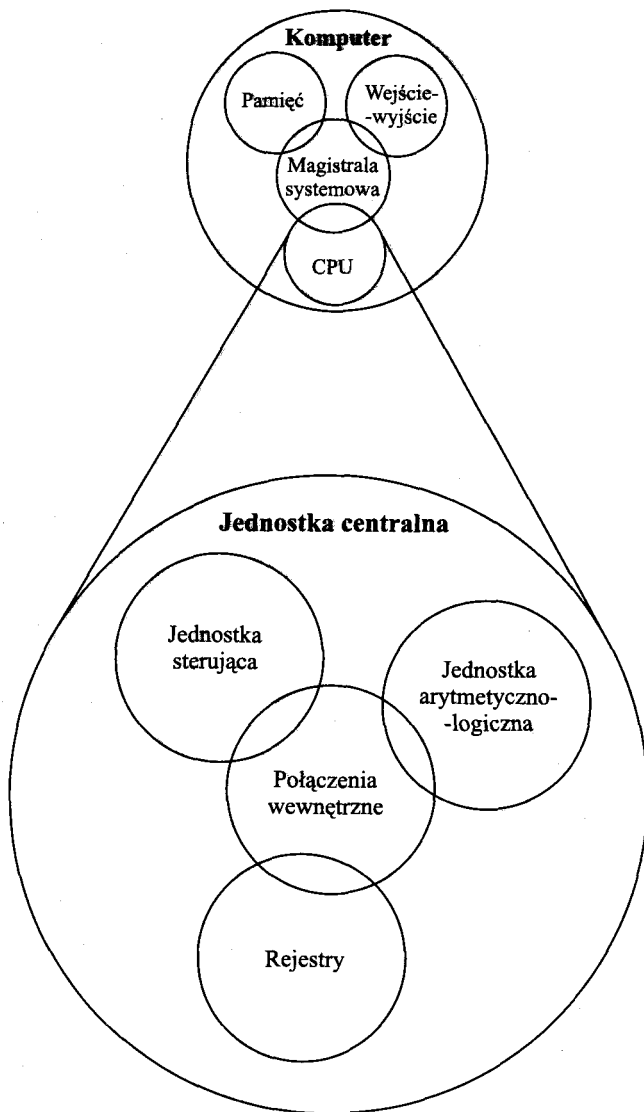
RYSUNEK 1.3. Komputer



RYSUNEK 1.4. Struktura komputera

Każdy z wymienionych składników omówimy szczegółowo w części II. Dla naszych celów najbardziej interesującym i pod pewnymi względami najbardziej złożonym składnikiem jest jednostka centralna. Jej strukturę widać na rys. 1.5. Głównymi składnikami strukturalnymi procesora są:

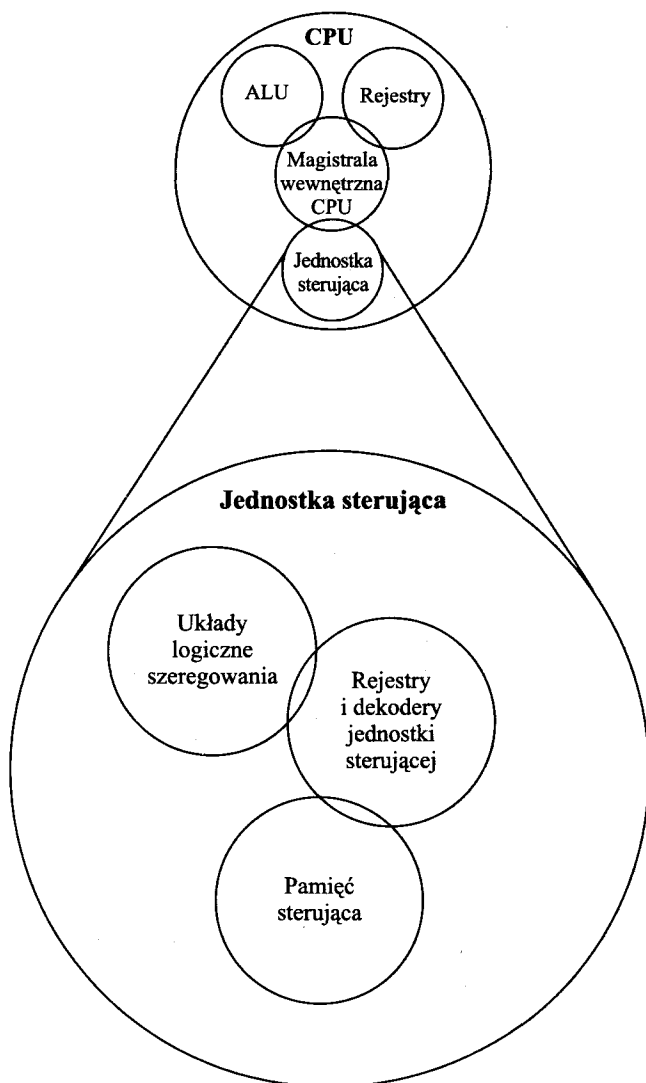
- a **Jednostka sterująca.** Steruje działaniem procesora i przez to komputera.
- **Jednostka arytmetyczno-logiczna (ALU).** Realizuje funkcję przetwarzania danych przez komputer.



RYSUNEK 1.5. Jednostka centralna

- a Rejestry. Realizują wewnętrzne przechowywanie danych w procesorze.
- a Połączenia procesora. Mechanizm zapewniający łączność między jednostką sterującą, ALU i rejestrami.

Każdym z tych składników zajmiemy się szczegółowo w części III. Najbardziej interesującym dla nas składnikiem jest jednostka sterująca. Istnieje obecnie kilka podejść do projektowania jednostek centralnych, jednak dalece najpowszechniejszym jest wykorzystywanie **mikroprogramowania**. W tym przypadku struktura jednostki sterującej wygląda tak, jak na rys. 1.6. Struktura ta zostanie przeanalizowana w części IV.



RYСУNEK 1.6. Jednostka sterująca

ZAWARTOŚĆ KSIĄŻKI

Rozdział ten jest wprowadzeniem do całej książki; zawiera on krótkie streszczenia pozostałych rozdziałów.

Ewolucja i wydajność komputerów

Rozdział 2 służy dwóm celom. Po pierwsze, przedyskutowanie historii techniki komputerowej jest łatwą i interesującą drogą do wprowadzenia podstawowych koncepcji organizacji i architektury komputerów. W rozdziale tym są również przedstawione tendencje rozwoju techniki, które sprawiły, że wydajność stała się głównym celem przy projektowaniu systemów komputerowych. Są w nim omówione różne techniki i strategie wykorzystywane do osiągnięcia zrównoważonych i wydajnych systemów.

Struktury połączeń

Na najwyższym poziomie komputer składa się z procesora, pamięci oraz urządzeń wejścia-wyjścia. Funkcjonalnie rzecz biorąc, działanie systemu polega na wymianie danych i sygnałów sterujących między tymi zespołami. W celu umożliwienia tej wymiany zespoły muszą być wzajemnie połączone. Rozdział 3 rozpoczyna się od krótkiej analizy poszczególnych zespołów komputera oraz ich wymagań wejściowo-wyjściowych. Następnie są rozpatrzone kluczowe zagadnienia, które wpływają na projektowanie połączeń, a zwłaszcza na potrzebę obsługi przerwań. Większość rozdziału jest poświęcona analizie najpopularniejszego rozwiązania połączeń, jakim jest wykorzystanie struktury magistrali.

Pamięć wewnętrzna

Pamięci komputerowe obejmują szeroki zakres typów, technologii, organizacji, wydajności i cen. Typowy system komputerowy zawiera całą hierarchię podsystemów pamięciowych. Niektóre z nich są wewnętrzne (tzn. bezpośrednio dostępne dla procesora), pozostałe zaś zewnętrzne (dostępne dla procesora za pośrednictwem modułów wejścia-wyjścia). Rozdział 4 rozpoczyna się od przeglądu tej hierarchii, po którym znajduje się opis problemów projektowych dotyczących pamięci wewnętrznych. Po pierwsze, zajmiemy się naturą i organizacją półprzewodnikowej pamięci głównej. Następnie rozpatrzymy szczegółowo projektowanie pamięci podręcznych, uwzględniając odrębne pamięci dla programów i danych, a także dwupoziomowe pamięci podręczne. Na zakończenie przeanalizujemy organizację współczesnych, złożonych pamięci DRAM.

Pamięci zewnętrzne

Rozdział 5 dotyczy problemów projektowania i wydajności związanych z pamięciami dyskowymi. Zawiera też analizę układów RAID, które stają się coraz powszechniejsze, oraz dyskusję dotyczącą systemów pamięci optycznych i taśmowych.

Wejście-wyjście

Moduły wejścia-wyjścia są połączone z procesorem i pamięcią główną, poza tym każdy z nich steruje jednym lub wieloma urządzeniami zewnętrznymi. W rozdziale 6 są opisane mechanizmy, za pomocą których moduł wejścia-wyjścia współdziała z resztą systemu komputerowego, z uwzględnieniem techniki programowanego wejścia-wyjścia, wykorzystywania przerwań oraz bezpośredniego dostępu do pamięci (DMA). Rozdział ten zawiera również opis interfejsu między modułami wejścia-wyjścia a urządzeniami zewnętrznymi.

Współpraca z systemem operacyjnym

W tym miejscu stosowne jest spojrzenie na system operacyjny, aby wyjaśnić, jak zarządza się podstawowymi zespołami komputera w celu wykonania użytecznej pracy oraz jak organizacja sprzętu komputerowego zapewnia współpracę z systemem operacyjnym. Na początku rozdziału 7 przedstawimy krótką historię, która służy do określenia głównych rodzajów systemów operacyjnych oraz do umotywowania ich wykorzystywania. Następnie wyjaśnimy wieloprogramowość i opiszemy funkcje szeregowania długo- i krótkookresowego. Na zakończenie zajmiemy się zarządzaniem pamięcią z uwzględnieniem segmentowania, stronicowania i pamięci wirtualnej.

Arytmetyka komputera

Rozdział 8 rozpoczyna się od szczegółowego omówienia działania procesora z uwzględnieniem arytmetyki komputerowej. Typowo procesory realizują dwa rodzaje arytmetyki: stałopozycyjną (liczb całkowitych) oraz zmiennopozycyjną. Najpierw, dla obu przypadków, opiszemy sposób reprezentacji liczb, po czym - działania arytmetyczne. Szczegółowo przeanalizujemy normę zmiennopozycyjną IEEE 754.

Listy rozkazów

? punktu widzenia programisty najlepszą drogą do zrozumienia działania procesora jest nauczenie się listy rozkazów maszynowych, które on wykonuje. Rozdział 9 zawiera opis kluczowych własności list rozkazów maszynowych,

z uwzględnieniem różnorodnych typów danych i operacji powszechnie występujących w tych listach. Znaleźć tam również można krótkie objaśnienie zależności między rozkazami procesora a assemblerem. Rozdział 10 dotyczy sposobów adresowania. Na zakończenie są rozpatrzone problemy formatowania rozkazów, z uwzględnieniem niezbędnych kompromisów.

Struktura i działanie procesora

Rozdział 11 jest poświęcony wewnętrznej strukturze i działaniu procesora. Jest w nim opisana ogólna organizacja (obejmująca ALU, jednostkę sterującą i zestaw rejestrów), a następnie - organizacja zestawu rejestrów. Pozostała część rozdziału dotyczy funkcjonowania procesora w czasie wykonywania rozkazów maszynowych. Przeanalizujemy cykl rozkazu w celu pokazania funkcjonowania i wzajemnych zależności cykli pobierania, adresowania pośredniego, wykonywania i przerwania. Na zakończenie szczegółowo opiszemy przetwarzanie potokowe stosowane w celu poprawienia wydajności.

Komputery o zredukowanej liście rozkazów (RISC)

Jedną z najbardziej znaczących innowacji w organizacji i architekturze komputerów w ostatnich latach jest komputer o zredukowanej liście rozkazów (RISC). Architektura RISC stanowi istotne odstępstwo od historycznych tendencji rozwoju architektury procesorów. Analiza tego rozwiązania naprowadza nas na wiele istotnych problemów dotyczących organizacji i architektury komputerów. W rozdziale 12 jest przedstawione rozwiązanie RISC i jego porównanie z koncepcją komputera o złożonej liście rozkazów (CISC).

Procesory superskalarne

W rozdziale 13 jest przedstawiona nowsza i równie ważna innowacja projektowa: procesor superskalarny. Chociaż technika superskalarna może być wykorzystana w dowolnym procesorze, jest ona szczególnie dostosowana do architektury RISC.

Działanie jednostki sterującej

Rozdział 14 zawiera kontynuację rozważań na temat funkcjonowania procesora. W szczególności zajmiemy się w nim sposobem sterowania poszczególnymi składnikami procesora przez jednostkę sterującą w celu umożliwienia realizacji funkcji procesora. Pokażemy, że każdy cykl rozkazu jest budowany za pomocą zestawu mikrooperacji, które generują sygnały sterujące. Wykonywanie rozkazów jest zapewniane przez te właśnie sygnały sterujące, pochodzące z jednostki

sterującej i kierowane do ALU, rejestrów i do struktury połączeń systemu. Na zakończenie przedstawimy rozwiązanie układowe w budowie jednostek sterujących.

Sterowanie mikroprogramowe

W rozdziale 15 jest opisane, jak jednostka sterująca może być konstruowana z wykorzystaniem metody mikroprogramowania. Najpierw zajmiemy się przetwarzaniem mikrooperacji na mikrorozkazy, a następnie - formą pamięci sterującej zawierającej mikroprogram dla każdego rozkazu maszynowego. Wyjaśnimy też strukturę i funkcje mikroprogramowanej jednostki sterującej.

Wieloprocessorowość i przetwarzanie wektorowe

Tradycyjnie komputer był widziany jako maszyna sekwencyjna. W miarę ewolucji technologii komputerowej i spadku kosztów sprzętu komputerowego, projektanci komputerów coraz częściej widzieli możliwość rozwiązań równoległych, co prowadziło do poprawy wydajności, a w pewnych przypadkach także niezawodności. W rozdziale 16 są przedstawione dwa najwybitniejsze i najbardziej udane zastosowania organizacji równoległej: systemy wieloprocessorowe i organizacja wektorowa. Dla przypadku systemów wieloprocessorowych jest w tym rozdziale przedyskutowany kluczowy problem projektowy, jakim jest zgodność pamięci podręcznych.

Procesory równoległe

Rozdział 16 dotyczy procesorów równoległych, które są skonfigurowane wieloprocessorowo. Organizacja taka wnosi nowe problemy, nie występujące w tradycyjnych konfiguracjach wieloprocessorowych po prostu ze względu na skalę organizacji, która może obejmować setki a nawet tysiące procesorów.

Logika cyfrowa

W podstawowej części książki binarne elementy pamięciowe i funkcje cyfrowe są traktowane jako podstawowe składniki systemu cyfrowego. W dodatku do książki jest opisane, jak te elementy pamięciowe i funkcje mogą być wykorzystane w cyfrowych układach logicznych. Dodatek rozpoczyna się od krótkiego przeglądu algebry Boole'a. Następnie jest wprowadzona koncepcja bramki, a na zakończenie są opisane układy kombinacyjne i sekwencyjne, które mogą być zbudowane z bramek.

Ewolucja i wydajność komputerów

Nasze studiowanie zagadnień dotyczących komputerów rozpoczniemy od krótkiej historii. Historia ta jest interesująca sama w sobie, a także umożliwia wstępne zapoznanie się ze strukturą i działaniem komputerów. Następnie rozpatrzymy problem wydajności. Rozważymy potrzebę zrównoważonego wykorzystywania zasobów komputera, co okaże się ważnym kontekstem w całej książce. Na zakończenie omówimy ewolucję dwóch systemów, które służą jako podstawowe przykłady w całej książce: Pentium i PowerPC.

2.1

KRÓTKA HISTORIA KOMPUTERÓW

Pierwsza generacja: lampy próżniowe

ENIAC

ENIAC (skrót od ang. *Electronic Numerical Integrator And Computer* - elektroniczne urządzenie numeryczne całkujące i liczące) został zaprojektowany i zbudowany pod nadzorem Johna Mauchly i Johna Prespera Eckerta z University of Pennsylvania. Był pierwszym na świecie elektronicznym komputerem cyfrowym o ogólnym przeznaczeniu.

Przedsięwzięcie to było odpowiedzią na wojenne potrzeby USA. Wojskowe Laboratorium Badań Balistycznych (BRL), odpowiedzialne za opracowywanie tablic zasięgu i toru dla nowych broni, miało trudności z dostarczaniem tych tablic w rozsądnym czasie i przy zachowaniu wymaganej dokładności. Bez tych tablic nowe rodzaje broni i artyleria były bezużyteczne. BRL zatrudniało ponad 200 osób, w większości kobiet, które posługując się biurkowymi kalkulatorami rozwiązywały niezbędne równania balistyczne. Przygotowanie tablic dla pojedynczej broni zajmowało jednej osobie wiele godzin, a nawet dni.

Mauchly, który był profesorem elektrotechniki University of Pennsylvania, oraz Eckert, jeden z jego studentów, zaproponowali zbudowanie komputera o ogólnym przeznaczeniu przy wykorzystaniu lamp próżniowych, mając na uwadze zastosowanie go w BRL. W roku 1943 propozycja ta została zaakceptowana przez wojsko, po czym rozpoczęto prace nad maszyną ENIAC. Urządzenie było ogromne: ważyło 30 t, zajmowało ok. 1400 m² i zawierało ponad 18 000 lamp próżniowych. W czasie pracy zużywało moc 140 kW. Było znacznie szybsze niż jakikolwiek komputer elektromechaniczny, wykonując 5000 operacji dodawania na sekundę.

ENIAC był maszyną raczej dziesiętną niż binarną. Liczby były reprezentowane w formie dziesiętnej i arytmetyka była realizowana w systemie dziesiętnym. Jego pamięć składała się z 20 „akumulatorów”, z których każdy mógł przechowywać 10-cyfrową liczbę dziesiętną. Każda cyfra była reprezentowana przez pierścień złożony z 10 lamp próżniowych. W określonym czasie tylko jedna lampa próżniowa znajdowała się w stanie włączonym (ON), reprezentując jedną z 10 cyfr. Główną wadą ENIAC-a było to, że musiał on być programowany ręcznie przez ustawianie przełączników oraz wtykanie i wyjmowanie kabli.

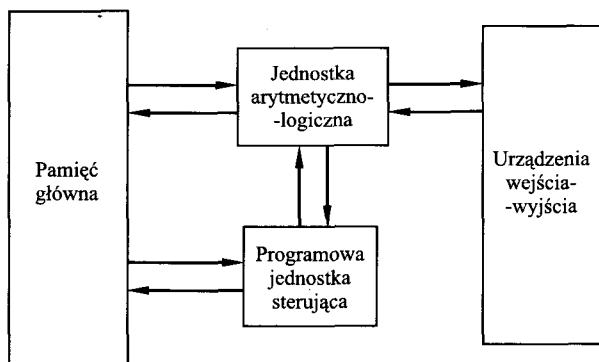
ENIAC został ukończony w roku 1946, zbyt późno, by mógł być wykorzystany w wojnie. Zamiast tego, jego pierwszym zastosowaniem było przeprowadzenie serii złożonych obliczeń, które były pomocne przy określaniu wykonalności bomby wodorowej. Zastosowanie ENIAC-a do zadania innego niż to, dla którego został zbudowany, było demonstracją jego z natury ogólnego przeznaczenia. Rok 1946 zapowiedział więc erę komputerów elektronicznych, co było kulminacją całych lat wysiłków. ENIAC działał w laboratorium BRL do roku 1955, po czym został rozebrany.

Maszyna von Neumanna

Jak już wspomnieliśmy, wprowadzanie i zmiana programów dla ENIAC-a było ekstremalnie nudne. Proces programowania mógłby być ułatwiony, jeśliby program mógł być reprezentowany w formie odpowiedniej do przechowywania w pamięci razem z danymi. Komputer pobierałby wtedy rozkazy przez odczytywanie ich z pamięci, a program mógłby być instalowany lub zmieniany przez zmianę zawartości części pamięci.

Idea ta, znana jako **koncepcja przechowywanego programu**, jest zwykle przypisywana projektantom ENIAC-a, głównie zaś matematykowi Johnowi von Neumannowi, który był konsultantem w tym przedsięwzięciu. Niemal w tym samym czasie koncepcja taka została opracowana przez Turinga. Po raz pierwszy idea ta została opublikowana w roku 1945 w propozycji von Neumanna opracowania nowego komputera pod nazwą EDVAC (skrót od ang. *Electronic Discrete Variable Computer*).

W roku 1946 von Neumann i jego koledzy rozpoczęli projektowanie nowego komputera wykorzystującego program przechowywany w pamięci. Miało to



RYSUNEK 2.1. Struktura komputera IAS

miejsce w Princeton Institute for Advanced Studies, a komputer określono skrótem IAS. Komputer IAS, chociaż nie został ukończony do roku 1952, był prototypem wszystkich następnych komputerów o ogólnym przeznaczeniu.

Na rysunku 2.1 jest pokazana ogólna struktura komputera IAS. Jego składnikami są:

- ❑ Pamięć główna, w której są przechowywane zarówno dane, jak i rozkazy.
- ❑ Jednostka arytmetyczno-logiczna (ALU) mogąca wykonywać działania na danych binarnych.
- ❑ Jednostka sterująca, która interpretuje rozkazy z pamięci i powoduje ich wykonanie.
- ❑ Urządzenia wejścia-wyjścia, których pracą kieruje jednostka sterująca.

Struktura ta została przedstawiona we wcześniejszej propozycji von Neumanna. Warto ją tutaj zacytować [VONM45]:

2.2. Po pierwsze, ponieważ urządzenie to jest przede wszystkim komputerem, najczęściej będzie wykonywało elementarne operacje arytmetyczne - dodawanie, odejmowanie, mnożenie i dzielenie (+, -, x, :). Jest więc rozsądne, że powinno posiadać wyspecjalizowane „organy” do wykonywania tych operacji.

Należy jednak zauważyć, że chociaż powyższa zasada jako taka brzmi rozsądnie, to szczegółowy sposób jej realizacji wymaga głębokiego zastanowienia... W każdym przypadku *centralna, arytmetyczna* część urządzenia będzie prawdopodobnie musiała istnieć, co oznacza występowanie *pierwszej specyficznej części komputera: CA*.

2.3. Po drugie, logiczne sterowanie urządzeniem, to znaczy odpowiednie szeregowanie jego operacji, może być najefektywniej realizowane przez centralny organ sterujący. Jeśli urządzenie ma być *elastyczne*, to znaczy możliwie *uniwersalne*, należy rozróżniać specyficzne rozkazy związane z określonym problemem i ogólne „organy” sterujące dbające o wykonanie tych rozkazów - czymkolwiek by one nie były. Te pierwsze muszą być w jakiś sposób przechowywane; te drugie - reprezentowane przez określone działające części urządzenia. Przez *sterowanie centralne* rozumiemy tylko tę ostatnią funkcję, a „organy”, które ją realizują, tworzą *drugą specyficzną część urządzenia: CC*.

2.4. Po trzecie, jakiekolwiek urządzenie, które ma wykonywać długie i skomplikowane sekwencje działań (w szczególności obliczenia), musi mieć odpowiednio dużą pamięć...

(b) Rozkazów kierujących rozwiązywaniem skomplikowanego problemu może być bardzo dużo, zwłaszcza wtedy, gdy kod jest przypadkowy (a tak jest w większości przypadków). Muszą one być pamiętane...

W każdej sytuacji całkowita pamięć stanowi *trzecią, specyficzną część urządzenia: M*.

2.6. Trzy specyficzne części CA, CC (razem C) oraz M odpowiadają neuronom skojarzeniowym w systemie nerwowym człowieka. Pozostają do przedyskutowania równoważniki neuronów *sensorycznych* (doprowadzających) i *motorycznych* (odprowadzających). Są to organy *wejścia* i *wyjścia* naszego urządzenia.

Urządzenie musi mieć możliwość utrzymywania kontaktu z wejściem i wyjściem z pomocą specjalistycznego narzędzia (porównaj 1.2). Narzędzie to będzie nazywane *zewnętrznym narzędziem rejestrującym urządzenia: R...*

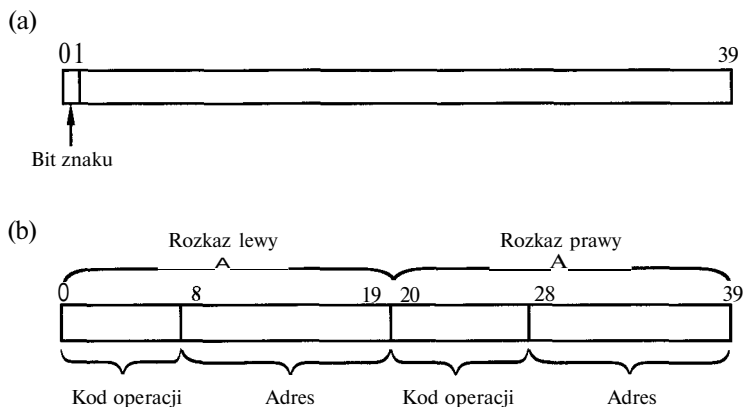
2.7. Po czwarte, urządzenie musi być wyposażone w organy przenoszące ... informację z R do swoich specyficznych części C i M. „Organ” te stanowiącego wejście, a więc *czwartą, specyficzną część: L*. Zobaczmy, że najlepiej jest dokonywać wszystkich przeniesień z R (poprzez I) do M, a nigdy bezpośrednio do C ...

2.8. Po piąte, urządzenie musi mieć „organy” służące do przenoszenia ... ze swoich specyficznych części C i M do R. Organów te tworzącego *wyjście*, a więc *piątą specyficzną część: O*. Zobaczmy, że znów najlepiej jest dokonywać wszystkich transferów z M (poprzez O) do R, nigdy zaś bezpośrednio z C ...

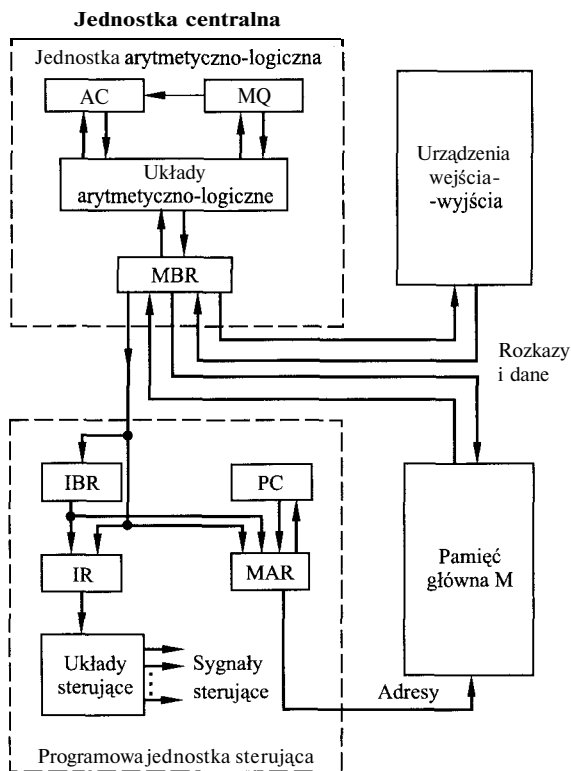
Z rzadkimi wyjątkami wszystkie dzisiejsze komputery mają tę samą ogólną strukturę i funkcję, są wobec tego określane jako maszyny von Neumanna. Warto więc w tym momencie krótko opisać działanie komputera IAS [BURK46]. Zgodnie z pracą [HAYE88] terminologia i notacja von Neumanna powinny być zmienione stosownie do wymagań współczesnych; przykłady i ilustracje towarzyszące tej dyskusji są zgodne z ostatnią spośród wyżej wymienionych pozycji literatury.

Pamięć IAS składa się z 1000 miejsc przechowywania, zwanych słowami, z których każde zawiera 40 cyfr binarnych (bitów). Przechowywane są tam zarówno dane, jak i rozkazy. Liczby muszą więc być prezentowane w formie binarnej, a każdy rozkaz również musi być w kodzie binarnym. Formaty te są pokazane na rys. 2.2. Każda liczba jest reprezentowana przez bit znaku i wartość 39-bitową. Słowo może także zawierać dwa rozkazy 20-bitowe, przy czym każdy rozkaz składa się z 8-bitowego kodu operacji określającego operację, która ma być realizowana, oraz 12-bitowego adresu określającego jedno ze słów w pamięci (ponumerowanych od O do 999).

Jednostka sterująca uruchamia IAS, pobierając rozkaz z pamięci i wykonując go – jeden rozkaz w określonym momencie. W celu wyjaśnienia tego działania potrzebny jest bardziej szczegółowy obraz struktury, pokazany na rys. 2.3. Na rysunku widać, że zarówno jednostka sterująca, jak i ALU zawierają miejsca pamięci zwane rejestrami, określane następująco:



RYSUNEK 2.2. Formaty pamięci IAS: (a) słowo danych; (b) słowo rozkazów



RYSUNEK 2.3. Struktura komputera IAS

- Rejestr buforowy pamięci (ang. *memory bufferregister* - MBR). Zawiera słowo, które ma być przechowywane w pamięci, lub też jest wykorzystywany do pobierania słów z pamięci.

- Rejestr adresowy pamięci (ang. *memory address register* - MAR). Określa adres w pamięci dotyczący słowa, które ma być zapisane w rejestrze MBR lub z niego odczytane.
- a Rejestr rozkazów (ang. *instruction register* - IR). Zawiera 8-bitowy kod operacyjny rozkazu, który jest wykonywany.
- a Buforowy rejestr rozkazów (ang. *instruction buffer register* - IBR). Jest wykorzystywany do czasowego przechowywania podręcznego rozkazu pochodzącego ze słowa w pamięci.
- a Licznik programu (ang. *program counter* - PC). Zawiera adres następnej pary rozkazów, która ma być pobrana z pamięci.
- a Akumulator (AC) i rejestr mnożenia-dzielenia (ang. *multiplier-quotier* - MQ). Wykorzystywane do czasowego przechowywania argumentów i wyników operacji prowadzonych przez ALU. Na przykład, wynikiem mnożenia dwóch liczb 40-bitowych jest liczba 80-bitowa; 40 najbardziej znaczących bitów przechowuje się w akumulatorze (AC), a najmniej znaczące w rejestrze MQ.

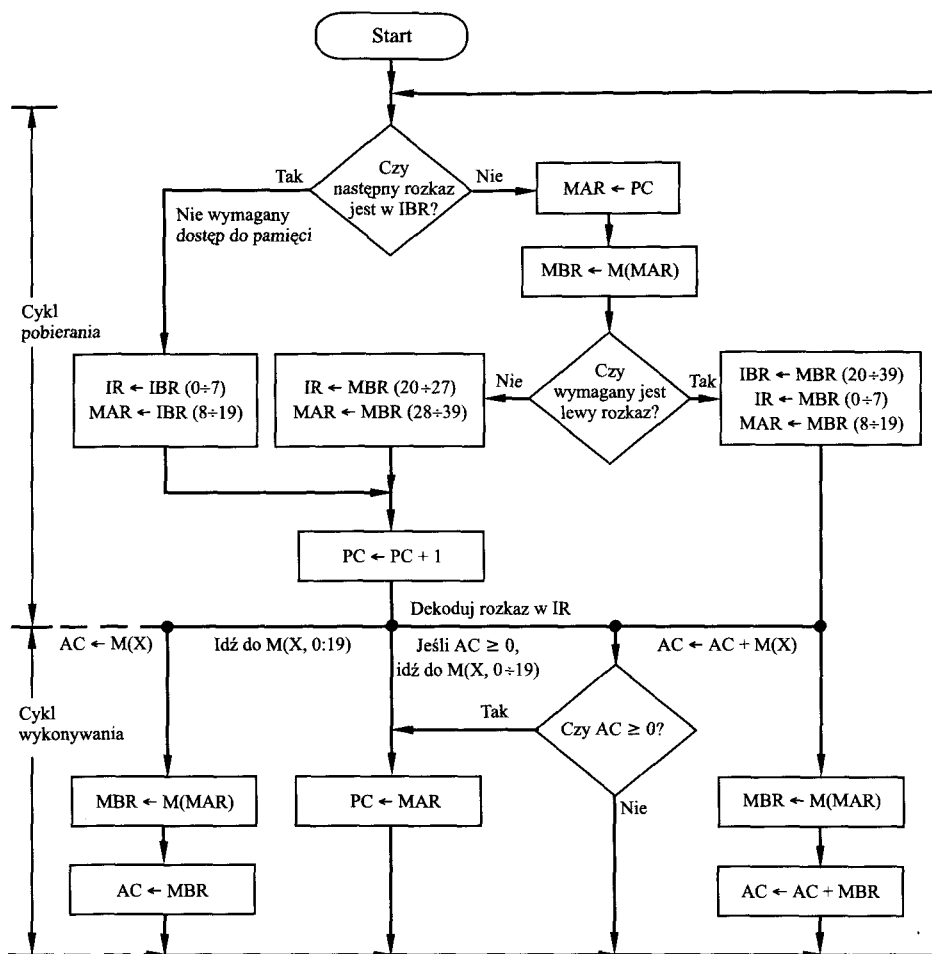
Komputer IAS działa przez powtarzalne wykonywanie **cyklu rozkazu** w sposób pokazany na rys. 2.4. Każdy cykl rozkazu składa się z dwóch podcykli. Podczas cyklu pobrania rozkazu kod operacji następnego rozkazu jest ładowany do rejestru rozkazu, natomiast część adresowa - do rejestru MAR. Rozkaz ten może pochodzić z rejestru IBR lub też może być uzyskany z pamięci przez załadowanie słowa do rejestru MBR, a następnie do rejestrów IBR, IR i MAR.

Dlaczego odbywa się to pośrednio? Po prostu wszystkie te operacje są sterowane za pomocą układów elektronicznych i powodują wykorzystywanie ścieżek danych. Aby uprościć układy elektroniczne, tylko jeden rejestr jest używany do określania adresu w pamięci zarówno przy odczycie, jak i przy zapisie. Rolę źródła i miejsca przeznaczenia również gra tylko jeden rejestr.

Gdy kod operacji znajdzie się w rejestrze IR, realizowany jest **cykl wykonywania**. Układy sterujące rozpoznają kod operacji i wykonują rozkaz, wysyłając odpowiednie sygnały sterujące, które powodują, że przenoszone są dane lub ALU wykonuje operację.

Komputer IAS ma łącznie 21 rozkazów. Są one wymienione w tabeli 2.1. Można je zakwalifikować do następujących grup:

- *Przenoszenie danych*. Należą tu rozkazy powodujące przesyłanie danych między pamięcią a rejestrami ALU lub między dwoma rejestrami ALU.
- a *Rozgałęzienia bezwarunkowe*. Zwykle jednostka sterująca wykonuje szeregowo rozkazy pochodzące z pamięci. Sekwencja rozkazów może być zmieniona przez rozkaz skoku. Umożliwia to wykonywanie operacji powtarzalnych.
- a *Rozgałęzienia warunkowe*. Skoki mogą być wykonywane zależnie od pewnego warunku, co oznacza występowanie punktów decyzyjnych.
- a *Arytmetyka*. Operacje wykonywane przez ALU.



$M(X)$ - zawartość lokacji pamięci o adresie X
 $(X+Y)$ - bity od X do Y

RYSUNEK 2.4. Częściowa sieć działań komputera IAS

- ❑ *Modyfikowanie adresu.* Należą tu rozkazy, które umożliwiają obliczanie adresów w ALU, a następnie wstawianie ich do rozkazów przechowywanych w pamięci. Dzięki temu program uzyskuje znaczną elastyczność adresowania.

W tabeli 2.1 są przedstawione rozkazy w formie symbolicznej, łatwej do odczytania. W rzeczywistości każdy rozkaz musi być zgodny z formatem przedstawionym na rys. 2.2b. Część stanowiąca kod operacji (pierwszych 8 bitów) służy do ustalenia, który z 21 rozkazów ma być wykonywany. Część adresowa (pozostałych 12 bitów) służy do ustalenia, które z 1000 miejsc pamięci ma być zaangażowane w wykonywanie rozkazu.

TABELA 2.1. Lista rozkazów IAS

Rodzaj rozkazu	Kod operacji	Reprezentacja symboliczna	Opis
Transfer danych	00001010	LOAD MQ	przenieś zawartość rejestru MQ do akumulatora AC
	00001001	LOAD MQ,M(X)	przenieś zawartość lokacji pamięci X do MQ
	00100001	STOR M(X)	przenieś zawartość akumulatora do lokacji pamięci X
	00000001	LOAD M(X)	przenieś M(X) do akumulatora
	00000010	LOAD - M(X)	przenieś - M(X) do akumulatora
	00000011	LOAD M(X)	przenieś wartość bezwzględną M(X) do akumulatora
	00000100	LOAD - M(X)	przenieś - M(X) do akumulatora
Rozgałęzienie bezwarunkowe	00001101	JUMP M(X,0:19)	pobierz następny rozkaz z lewej połowy M(X)
	00001110	JUMP M(X,20:39)	pobierz następny rozkaz z prawej połowy M(X)
Rozgałęzienie warunkowe	00001111	JUMP+ M(X,0:19)	jeśli liczba w akumulatorze jest nieujemna, pobierz następny rozkaz z lewej połowy M(X)
	00010000	JUMP+ M(X,20:39)	jeśli liczba w akumulatorze jest nieujemna, pobierz następny rozkaz z prawej połowy M(X)
Arytmetyczne	00000101	ADD M(X)	dodaj M(X) do akumulatora; umieść wynik w akumulatorze
	00000111	ADD M(X)	dodaj M(X) do akumulatora; umieść wynik w akumulatorze
	00000110	SUB M(X)	odejmij M(X) od akumulatora; umieść wynik w akumulatorze
	00001000	SUB M(X)	odejmij M(X) od akumulatora; umieść resztę w akumulatorze
	00001011	MUL M(X)	pomnóż M(X) przez MQ; umieść najbardziej znaczące bity wyniku w akumulatorze, a najmniej znaczące w MQ
	00001100	DIV M(X)	podziel zawartość akumulatora przez M(X); umieść iloraz w MQ, a resztę w akumulatorze
	00010101	LSH	pomnóż zawartość akumulatora przez 2, tzn. przesun w lewo o jedną pozycję
	00010010	RSH	podziel zawartość akumulatora przez 2, tzn. przesun w prawo o jedną pozycję
Modyfikacja adresu	00010010	STOR M(X,8:19)	zamień położone po lewej pole adresowe M(X) na 12 bitów akumulatora położonych po prawej
	00010011	STOR M(X,28:39)	zamień położone po prawej pole adresowe M(X) na 12 bitów akumulatora położonych po prawej

Na rysunku 2.4 widać kilka przykładowych rozkazów wykonywanych przez jednostkę sterującą. Zauważmy, że każda operacja wymaga kilku kroków. Niektóre z nich są całkiem złożone. Operacja mnożenia wymaga 39 podoperacji, po jednej dla każdej pozycji bitowej z wyjątkiem bitu znaku!

Komputery komercyjne

W latach pięćdziesiątych narodził się przemysł komputerowy, przy czym dwie firmy - Sperry i IBM - zdominowały rynek.

W roku 1947 Eckert i Mauchly utworzyli Eckert-Mauchly Computer Corporation, aby produkować komputery w celach komercyjnych. Ich pierwszą udaną maszyną był UNIVAC I (skrót od ang. *Universal Automatic Computer*), który został wykorzystany przez Bureau of the Census (Biuro Spisu Ludności) do obliczeń roku 1950. Eckert-Mauchly Computer Corporation stała się częścią oddziału UNIVAC firmy Sperry-Rand Corporation, która rozpoczęła budowę szeregu maszyn pochodnych.

UNIVAC I był pierwszym udanym komputerem komercyjnym. Był przeznaczony, jak wynika z jego nazwy, zarówno do zastosowań naukowych, jak i komercyjnych. W pierwszym artykule opisującym system tego komputera wymieniono następujące przykładowe zadania, które może on realizować: macierzowe rachunki algebraiczne, problemy statystyczne, obliczenia premii dla firmy ubezpieczającej na życie oraz problemy logistyczne.

UNIVAC II, o większej pojemności pamięci i większej wydajności niż UNIVAC I, powstał w późnych latach pięćdziesiątych i stanowił ilustrację kilku tendencji, które pozostały charakterystyczne dla przemysłu komputerowego. Po pierwsze, postęp techniczny umożliwia firmom budowanie coraz większych i potężniejszych komputerów. Po drugie, każda z firm próbuje zapewnić *kompatybilność nowych maszyn w stosunku do starszych*. Oznacza to, że programy napisane dla starszych maszyn mogą być stosowane w nowej maszynie. Strategia ta wynika z nadziei na zachowanie klientów; jeśli klient zdecyduje się na kupno nowej maszyny, zechce ją nabyć raczej w tej samej firmie, aby uniknąć ewentualnych strat środków zainwestowanych w programy.

Oddział UNIVAC zapoczątkował także rozwój serii komputerów 1100, które miały stanowić jego źródło utrzymania. Seria ta jest ilustracją rozróżnienia, które występowało przez pewien czas. Pierwszy model (UNIVAC 1103) oraz jego następcy przez wiele lat byli przeznaczeni do zastosowań naukowych, obejmujących długie i złożone obliczenia. Inne firmy skoncentrowały się na zastosowaniach w biznesie, polegających na przetwarzaniu dużych ilości danych tekstowych. Podział ten w większości zanikł, był jednak zauważalny przez wiele lat.

IBM, który pomógł zbudować urządzenie Mark I, a następnie był największym producentem urządzeń do przetwarzania wykorzystujących karty perforowane, wyprodukował swój pierwszy komputer elektroniczny z przechowywa-

nym programem - model 701 - w roku 1953. Komputer 701 był pierwotnie przeznaczony do zastosowań naukowych [BASH81]. W roku 1955 IBM wprowadził komputer 702, mający wiele cech sprzętowych, które odpowiadały zastosowaniom w biznesie. Były to pierwsze modele z długiej serii komputerów 700/7000, która umożliwiła firmie IBM osiągnięcie pozycji dominującego producenta komputerów.

Druga generacja: tranzystory

Pierwsza wielka zmiana w komputerach elektronicznych nadeszła wraz z zastąpieniem lampy próżniowej przez tranzystor. Tranzystor jest mniejszy, tańszy i wydziela mniej ciepła niż lampa próżniowa, a może być używany do budowy komputerów podobnie jak lampa. W przeciwieństwie do lampy próżniowej, która wymaga drutów, płytek metalowych i obudowy szklanej, tranzystor jest *elementem półprzewodnikowym* wykonanym z krzemu.

Tranzystor został wynaleziony w Bell Laboratories w roku 1947, a w latach pięćdziesiątych wywołał rewolucję elektroniczną. Dopiero jednak w późnych latach pięćdziesiątych całkowicie tranzystorowe komputery stały się osiągalne handlowo. I znów IBM nie był pierwszą firmą, która wykorzystała tę nową technologię. NCR oraz (z większym sukcesem) RCA były pierwszymi firmami dostarczającymi małe maszyny tranzystorowe. IBM wkroczył niedługo po nich z serią 7000.

Wykorzystanie tranzystora oznaczało początek *drugiej generacji komputerów*. Szeroko przyjęte jest klasyfikowanie komputerów na generacje wynikające z podstawowej technologii sprzętu (tabela 2.2). Każda następna generacja wyróżnia się większą szybkością, większą pojemnością pamięci i mniejszymi rozmiarami niż poprzednia.

TABELA 2.2. Generacje komputerów

Generacja	Lata (w przybliżeniu)	Technologia	Typowa szybkość (operacji na sekundę)
1	1946-1957	lampa próżniowa	40 000
2	1958-1964	tranzystor	200 000
3	1965-1971	mała i średnia skala integracji	1 000 000
4	1972-1977	wielka skala integracji	10 000 000
5	1978-	bardzo wielka skala integracji	100 000 000

Są również jeszcze inne zmiany. W drugiej generacji wprowadzono bardziej złożone jednostki arytmetyczno-logiczne oraz sterujące, zastosowano ulepszone języki programowania, a także rozpoczęto dostarczanie wraz z komputerem *oprogramowania systemowego*.

Wraz z drugą generacją pojawiła się firma Digital Equipment Corporation (DEC). DEC została założona w roku 1957 i w tym samym roku wyprodukowała pierwszy komputer - PDP-1. Ten komputer i ta firma dali początek mini-komputerom, które stały się tak ważne w trzeciej generacji.

IBM 7094

Od wprowadzenia serii 700 w roku 1952 do wprowadzenia ostatniego modelu rodziny 7000 w roku 1964 linia wyrobów IBM przechodziła ewolucję typową dla wyrobów komputerowych. Kolejne maszyny miały coraz większą wydajność, większą pojemność oraz (lub) niższy koszt.

Dane zawarte w tabeli 2.3 ilustrują tę tendencję. Rozmiar pamięci głównej, liczony w wielokrotnościach 2^{10} słów 36-bitowych, wzrósł z 2 K ($1\text{ K} = 2^{10}$) do 32 K słów, podczas gdy czas dostępu do słowa w pamięci, to znaczy **czas cyklu pamięci**, zmniejszył się od 30 us do 1,4 us. Liczba kodów operacji wzrosła od umiarkowanych 24 aż do 185. W ostatniej kolumnie jest podana względna szybkość działania procesora.

Zwiększenie szybkości osiągnięto w wyniku rozwoju elektroniki (np. tranzystory działają szybciej niż lampy próżniowe) oraz przez wzrost złożoności układów. Na przykład w IBM 7094 zastosowano pomocniczy rejestr rozkazów (ang. *Instruction Backup Register* - IBR), wykorzystywany do przechowywania (buforowania) następnego rozkazu. Na rozkaz pobrania jednostka sterująca pobiera dwa sąsiednie słowa z pamięci. Z wyjątkiem występowania rozkazu skoku, co nie jest zbyt częste, jednostka sterująca potrzebuje zaledwie połowę cyklu rozkazu na dostęp do rozkazu w pamięci. To tzw. wstępne pobieranie (lub inaczej pobieranie z wyprzedzeniem, ang. *prefetching*) znacznie redukuje przeciętny czas cyklu rozkazu.

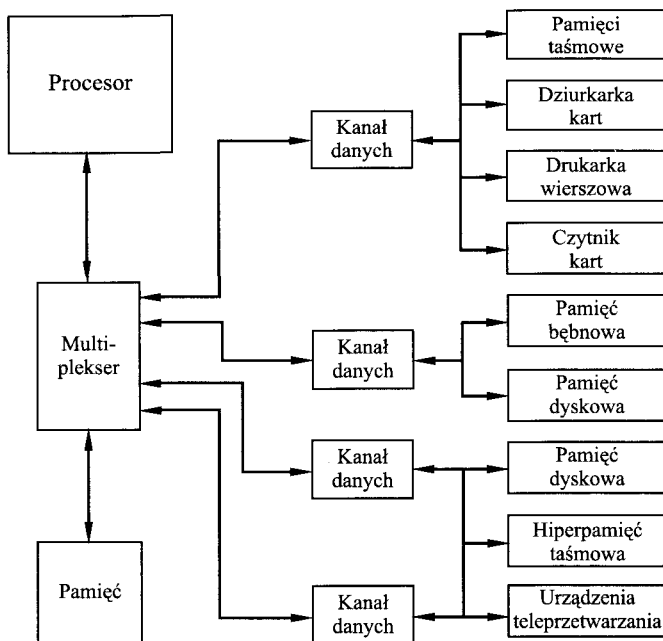
Pozostałe kolumny tabeli 2.3 staną się zrozumiałe w dalszym ciągu naszych rozważań.

Na rysunku 2.5 jest pokazana rozszerzona (z wieloma urządzeniami peryferyjnymi) konfiguracja IBM 7094, która jest reprezentatywna dla komputerów drugiej generacji [BELL71a]. Warto odnotować kilka różnic w stosunku do komputera IAS. Najważniejszą z nich jest użycie **kanałów danych**. Kanał danych jest niezależnym modułem wejścia-wyjścia z własnym procesorem i własną listą rozkazów. W systemie komputerowym wyposażonym w takie urządzenia procesor nie wykonuje szczegółowych rozkazów wejścia-wyjścia. Rozkazy takie są przechowywane w pamięci głównej, po czym są wykonywane przez specjalny procesor w samym kanale danych. Procesor inicjuje przesyłanie danych z wejścia-wyjścia przez wysłanie sygnału sterującego do kanału danych, powodując wykonanie przez niego sekwencji rozkazów zawartych w pamięci. Kanał danych realizuje swoje zadanie niezależnie od procesora, sygnalizuje tylko procesorowi zakończenie operacji. W takim rozwiązaniu procesor jest odciążony od wykonywania części przetwarzania.

Inną nową cechą jest występowanie **multipleksera**, który stanowi centralny punkt zbieżny kanałów danych, procesora i pamięci. Multiplekser szereguje dostęp procesora i kanałów danych do pamięci, pozwalając tym urządzeniom na działanie niezależne.

TABELA 2.3. Przykładowe modele serii 700/7000 IBM

Numer modelu	Pierwsza dostawa	Technologia CPU	Technologia pamięci	Czas cyklu [μ s]	Rozmiar pamięci [KB]	Liczba kodów operacji	Liczba rejestrów modyfikacji	Układowe rozwiązanie obliczeń zmiennie-pozycyjnych	Nakładanie się wejścia-wyjścia (kanały)	Nakładanie się pobierania rozkazów	Szybkość (przybliżona)
701	1952	lampy próżniowe	lampy elektrostatyczne	30	244	24	0	nie	nie	nie	1
704	1955	lampy próżniowe	rdzenie	12	4÷32	80	3	tak	nie	nie	2,5
709	1958	lampy próżniowe	rdzenie	12	32	140	3	tak	tak	nie	4
7090	1960	tranzystory	rdzenie	2,18	32	169	3	tak	tak	nie	25
7094 I	1962	tranzystory	rdzenie	2	32	185	7	tak (podwójna dokładność)	tak	tak	30
7094 II	1964	tranzystory	rdzenie	1,4	32	185	7	tak (podwójna dokładność)	tak	tak	50



RYSUNEK 2.5. Konfiguracja komputera IBM 7094

Trzecia generacja: układy scalone

Pojedynczy, zamknięty we własnej obudowie tranzystor jest nazywany podzespołem (elementem) **dyskretnym**. Przez lata pięćdziesiąte i wczesne sześćdziesiąte urządzenia elektroniczne były złożone w większości z podzespołów dyskretnych - tranzystorów, rezystorów, kondensatorów itd. Podzespoły dyskretny były produkowane oddzielnie, montowane w obudowy oraz lutowane lub mocowane w inny sposób na płytkach drukowanych, które z kolei były instalowane w komputerach, oscyloskopach i innych urządzeniach elektronicznych. Mała metalowa obudowa w kształcie walca zawierająca mikroskopijny kawałek krzemu, zwana tranzystorem, musiała być przylutowana do obwodu drukowanego. Cały proces wytwarzania, od tranzystora do zmontowanej płytki, był kosztowny i niewygodny.

Powyższe fakty zaczęły być źródłem trudności w przemyśle komputerowym. Wczesne komputery drugiej generacji zawierały około 10 000 tranzystorów. Liczba ta urosła do setek tysięcy, stwarzając narastające trudności producentom nowszych, potężniejszych maszyn.

W roku 1958 pojawiło się rozwiązanie, które zrewolucjonizowało elektronikę i rozpoczęło erę mikroelektroniki: wynaleziono układ scalony. To właśnie układ scalony określa trzecią generację komputerów. W tym podrozdziale dokonamy krótkiego wprowadzenia do technologii układów scalonych. Następnie zapoz-

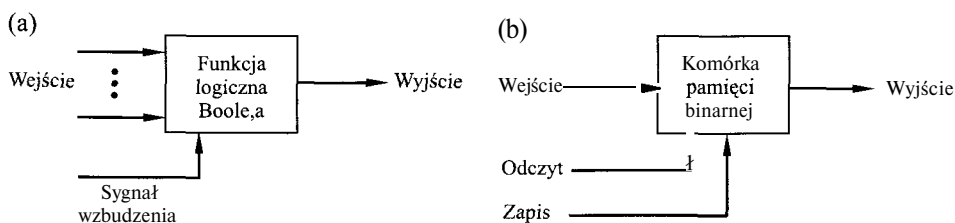
namy się z dwoma być może najważniejszymi przedstawicielami trzeciej generacji, wprowadzonymi na początku ery mikroelektroniki - komputerami IBM System 360 oraz DEC PDP-8.

Mikroelektronika

Mikroelektronika oznacza dosłownie „małą elektronikę”. Od samego początku elektroniki cyfrowej i przemysłu elektronicznego występowała trwała i konsekwentna tendencja do redukcji rozmiarów cyfrowych układów elektronicznych. Przed zbadaniem następstw i korzyści wynikających z tej tendencji, musimy powiedzieć więcej o naturze elektroniki cyfrowej. Bardziej szczegółową dyskusję można znaleźć w dodatku A.

Jak wiemy, podstawowe elementy komputera cyfrowego muszą realizować funkcje przechowywania, przenoszenia, przetwarzania i sterowania. Potrzebne są do tego tylko dwa typy podstawowych składników (rys. 2.6): bramki i komórki pamięci. Bramka jest przyrządem, który realizuje prostą funkcję logiczną, taką jak „jeśli A i B są prawdziwe, to prawdziwe jest C” (bramka I, ang. *AND*). Przyrządy takie są nazywane bramkami, ponieważ sterują one przepływem danych podobnie jak bramki (śluzы) w kanałach. Komórka pamięci jest przyrządem, który może przechowywać pojedynczy bit danych; oznacza to, że przyrząd ten w określonym czasie może znajdować się w jednym z dwóch stabilnych stanów. Przez połączenie dużej liczby tych podstawowych przyrządów możemy zbudować komputer. Możemy to następująco odnieść do naszych czterech podstawowych funkcji:

- Q *Przechowywanie danych.* Realizowane przez komórki pamięciowe.
- *Przetwarzanie danych.* Realizowane przez bramki.
- Q *Przenoszenie danych.* Ścieżki między podzespołami są wykorzystywane do przenoszenia danych z pamięci do pamięci oraz z pamięci za pośrednictwem bramek do pamięci.
- *Sterowanie.* Ścieżki między podzespołami mogą przenosić sygnały sterujące. Na przykład bramka może mieć jedno lub dwa wejścia danych oraz wejście sygnału sterującego, które aktywuje tę bramkę. Jeśli sygnał sterujący jest włączony (ON), bramka realizuje swoją funkcję na danych wejściowych i wytwarza dane wyjściowe. Podobnie komórka pamięci przechowuje bit do-



RYSUNEK 2.6. Podstawowe elementy komputera: (a) bramka; (b) komórka pamięci

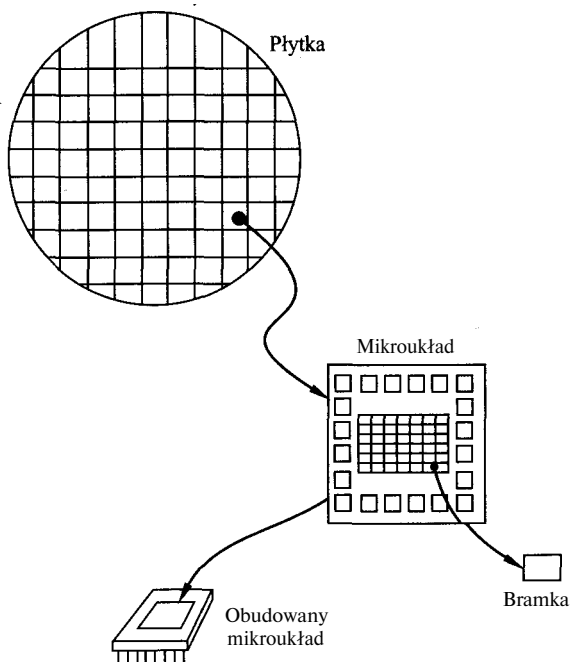
przewodzony do swojego wejścia, jeśli sygnał sterujący ZAPISZ (WRITE) będzie włączony (ON), natomiast umieści ten bit na swoim wyjściu, jeśli w stanie ON będzie sygnał sterujący CZYTAJ (READ).

Komputer składa się więc z bramek, komórek pamięci i połączeń między nimi. Bramki i komórki pamięci są z kolei zbudowane z pojedynczych podzespołów elektronicznych.

Chociaż technika tranzystorowa wprowadzona w drugiej generacji komputerów stanowiła znaczne udoskonalenie w stosunku do lamp próżniowych, trudności pozostały. Tranzystory były pojedynczo montowane w odrębnych obudowach i łączone za pomocą obwodów drukowanych. Był to proces złożony, czasochłonny i podatny na błędy.

W układach scalonych wykorzystano możliwość wytwarzania tranzystorów, rezystorów i przewodników w półprzewodniku, jakim jest krzem. Dzięki rozwojowi technologii ciała stałego, powstała możliwość wytworzenia całego układu w niewielkim kawałku krzemu, zamiast montowania dyskretnych podzespołów wykonanych z odrębnych kawałków krzemu. Setki a nawet tysiące tranzystorów można jednocześnie wytworzyć w pojedynczej płytce krzemowej. Równie ważne jest, że tranzystory te mogą być połączone w procesie metalizacji, tworząc układy.

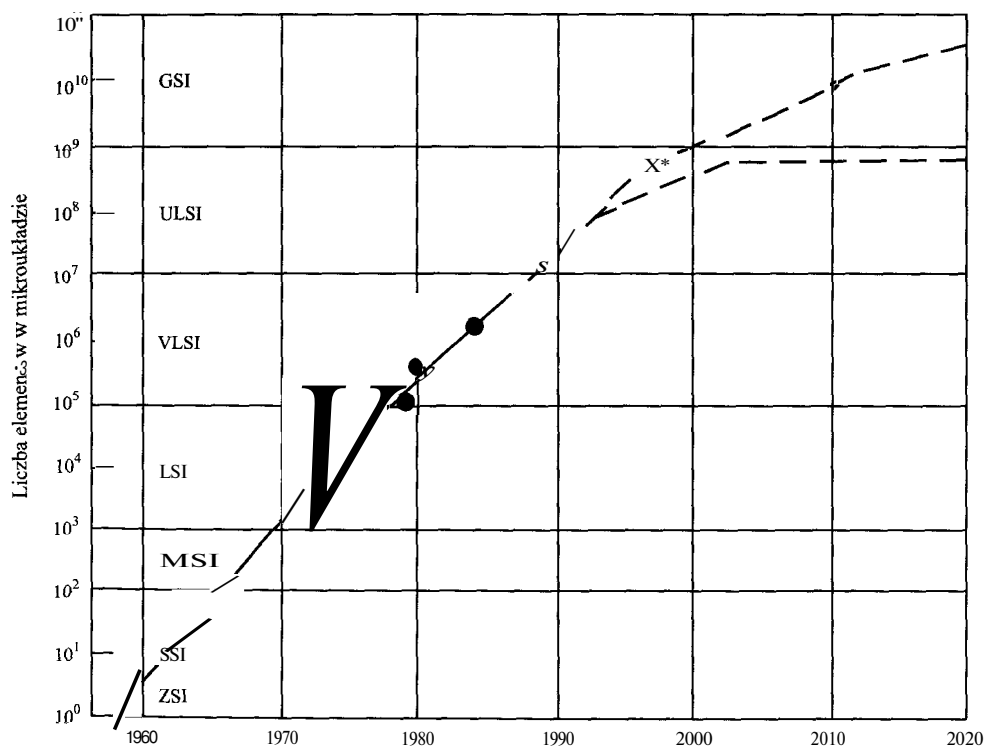
Na rysunku 2.7 jest przedstawiona kluczowa koncepcja wytwarzania układów scalonych. Cienka płytka krzemowa jest dzielona na wiele małych obsza-



RYSUNEK 2.7. Związek między płytką, mikroukładem i bramką

rów o powierzchni kilku milimetrów kwadratowych. W każdym obszarze jest wytwarzany identyczny układ (mikroukład, ang. *chip*), po czym płytką jest dzielona. Każdy mikroukład składa się z wielu bramek oraz z pewnej liczby kontaktów wejściowych i wyjściowych. Jest on następnie montowany w obudowie stanowiącej ochronę i zaopatrzonej w końcówki do montażu. Na płycie z obwodem drukowanym można następnie zmontować pewną liczbę takich obudów w celu wytworzenia większych i bardziej złożonych układów.

Początkowo można było niezawodnie wytwarzać jedynie kilka bramek lub komórek pamięciowych w jednym mikroukładzie. Te wczesne układy scalone noszą miano układów o **małej skali integracji** (ang. *small-scale integration* - SSI). W miarę upływu czasu stało się możliwe upakowywanie coraz większej ilości elementów w tym samym mikroukładzie. Ten wzrost gęstości jest zilu-



- ZSI - zerowa skala integracji (elementy dyskretne)
- SSI - mała skala integracji
- MSI - średnia skala integracji
- LSI - wielka skala integracji
- VLSI - bardzo wielka skala integracji
- ULSI - ultrawielka skala integracji
- GSI - gigantyczna skala integracji

RYСУNEK 2.8. Wzrost gęstości upakowania mikroukładów [MEIN87]

strowany na rys. 2.8; jest to jedna z najbardziej godnych uwagi tendencji technologicznych, jakie kiedykolwiek zarejestrowano. Począwszy od jedności w roku 1959, liczba elementów w mikroukładzie podwajała się corocznie w latach sześćdziesiątych. W latach siedemdziesiątych szybkość zmian zmalała, jednak tylko do (nadal budzącego respekt) poziomu 4-krotnego wzrostu w ciągu każdych 3 lat. Poziom ten ustabilizował się we wczesnych latach dziewięćdziesiątych, po czym wpływ ograniczeń fizycznych prawdopodobnie znów spowolni ten rozwój. Pomimo tego, zgodnie z bardziej optymistycznymi przewidywaniami, integracja o gigaskali (ang. *gigascale integration* - GSI) - miliard elementów w mikroukładzie - zostanie osiągnięta w końcu stulecia. (Przewidywania różnią się w zależności od założeń dotyczących ograniczeń nałożonych przez technologię wytwarzania mikroukładów).

Dla wytwórcy komputerów wykorzystanie coraz gęściej upakowanych układów scalonych oznacza wiele korzyści:

1. Koszt mikroukładu pozostaje w zasadzie niezmienny w przedstawionym okresie szybkiego wzrostu gęstości upakowania. Oznacza to, że koszt układów logicznych i pamięciowych komputera maleje drastycznie.
2. Ponieważ elementy logiczne i pamięciowe są umieszczane coraz bliżej w coraz gęściej upakowywanych mikroukładach, wzrasta szybkość działania.
3. Komputer staje się mniejszy, co umożliwia umieszczanie go w najróżniejszym otoczeniu.
4. Maleją wymagania dotyczące mocy i chłodzenia.
5. Połączenia wewnątrz układu scalonego są o wiele bardziej niezawodne niż połączenia lutowane. Im więcej układów w każdym mikroukładzie, tym mniej połączeń między mikroukładami.

System 360 IBM

W roku 1964 IBM panował na rynku komputerowym, produkując maszyny serii 7000. W tym samym roku firma zaanonsowała System 360, nową rodzinę komputerów. Choć ta zapowiedź nie była niespodzianką, zawierała pewne nieprzyjemne nowości dla aktualnych klientów IBM: linia 360 nie była kompatybilna ze starszymi maszynami IBM. Tak więc przejście do rodziny 360 było dla tych klientów trudne. Ze strony IBM było to przedsięwzięcie zuchwałe, jednak firma uważała je za konieczne dla przełamania pewnych ograniczeń architektury komputerów z rodziny 7000 oraz w celu wytworzenia systemu, który miał ewoluować wraz z technologią układów scalonych [PADE81], [GIFF87]. Strategia ta opłaciła się zarówno finansowo, jak i technicznie. Seria 360 była sukcesem dekady i umocniła pozycję IBM jako dominującego dostawcy komputerów, którego udział w rynku przekroczył 70%. Z pewnymi rozszerzeniami i modyfikacjami architektura serii 360 pozostaje do dziś architekturą dużych komputerów IBM. Przykłady wykorzystania tej architektury można znaleźć w dalszej części książki.

System 360 był pierwszą zaplanowaną rodziną komputerów. Rodzinę tą stanowiły komputery znacznie różniące się wydajnością i ceną. W tablicy 2.4 są pokazane pewne kluczowe własności różnych modeli z roku 1965 (każdy model rodziny jest wyróżniony numerem modelu). Różne modele były kompatybilne w tym sensie, że program napisany dla jednego modelu mógł być wykonywany na innym modelu tej serii, przy czym jedyną różnicą był czas potrzebny na wykonanie programu.

TABELA 2.4. Podstawowe parametry komputerów rodziny System 360 IBM

Parametr	Model 30	Model 40	Model 50	Model 65	Model 75
Maksymalna pojemność pamięci (w bajtach)	64 K	256 K	256 K	512 K	512 K
Szybkość pobierania danych z pamięci [MB/s]	0,5	0,8	2,0	8,0	16,0
Czas cyklu procesora [μ s]	1,0	0,625	0,5	0,25	0,2
Względna szybkość	1	3,5	10	21	50
Maksymalna liczba kanałów danych	3	3	4	6	6
Maksymalna szybkość przesyłania danych przez jeden kanał [KB/s]	250	400	800	1250	1250

Koncepcja rodziny kompatybilnych komputerów była zarówno nowa, jak i bardzo udana. Klient o umiarkowanych wymaganiach i budżecie mógł zacząć od względnie taniego modelu 30. Później, jeśli potrzeby klienta wzrosły, możliwe było zainteresowanie się szybszą maszyną o większej pamięci bez utraty inwestycji poniesionych dotychczas na oprogramowanie. Własności rodziny 360 są następujące:

- *Podobna lub identyczna lista rozkazów.* W wielu przypadkach dokładnie ta sama lista rozkazów służyła wszystkim członkom rodziny. Program wykonywany na jednej maszynie mógł być zatem wykonywany na dowolnej innej. W pewnych przypadkach najmniejsze maszyny szeregu posługiwały się listą rozkazów, która stanowiła część listy rozkazów maszyn największych. Oznaczało to, że program mógł być wykonywany przez większe maszyny, natomiast najmniejsze maszyny nie mogły wykorzystywać programów opracowanych dla dużych maszyn.
- a *Podobny lub identyczny system operacyjny.* Ten sam podstawowy system operacyjny był osiągalny dla wszystkich członków rodziny. W niektórych przypadkach największe maszyny wyróżniały się pewnymi uzupełnieniami.
- a *Rosnąca szybkość.* Szybkość wykonywania rozkazów rosła przy przechodzeniu od najmniejszych do największych maszyn rodziny.
- a *Rosnąca liczba urządzeń wejścia-wyjścia.* Liczba ta rosła przy przechodzeniu od najmniejszych do największych maszyn rodziny.
- Q *Rosnący rozmiar pamięci.* Rozmiar ten rósł przy przechodzeniu od najmniejszych do największych maszyn rodziny.

a *Rosnąca cena.* Cena rosła przy przechodzeniu od najmniejszych do największych maszyn rodziny.

W jaki sposób taka koncepcja rodziny maszyn mogła być zrealizowana? Różnice wynikały z trzech czynników: podstawowej szybkości, rozmiaru i stopnia jednoczesności [STEV64]. Na przykład, większa szybkość wykonywania określonego rozkazu mogła być uzyskana przez zastosowanie bardziej złożonych układów arytmetyczno-logicznych, pozwalających na równoległe wykonywanie podoperacji. Innym sposobem zwiększenia szybkości było zwiększenie szerokości ścieżki danych między pamięcią główną a procesorem. W modelu 30 tylko 1 bajt (8 bitów) mógł być w określonym momencie pobierany z pamięci, podczas gdy w modelu 70 możliwe było pobieranie 8 bajtów.

System 360 nie tylko wyznaczał przyszły kierunek rozwoju IBM, lecz także miał ogromny wpływ na cały przemysł. Wiele jego cech stało się normą dla dużych komputerów.

DEC PDP-8

W tym samym roku, w którym IBM wprowadził swój system 360, pojawiło się inne doniosłe opracowanie: komputer PDP-8 firmy DEC. Podczas gdy przeciętny komputer wymagał klimatyzowanego pomieszczenia, PDP-8 (ochrzczony w kręgach przemysłowych jako minikomputer na podobieństwo modnych wówczas minispódniczek) był dostatecznie mały, aby można go było umieścić na stole laboratoryjnym lub wbudować do innego urządzenia. Nie mógł on dokonać wszystkiego tego, co duża maszyna, jednak przy cenie 16 000 dolarów był tani na tyle, że każdy technik laboratoryjny mógł dysponować takim urządzeniem. Dla kontrastu, komputery Systemu 360 wprowadzone na rynek zaledwie kilka miesięcy wcześniej kosztowały setki tysięcy dolarów.

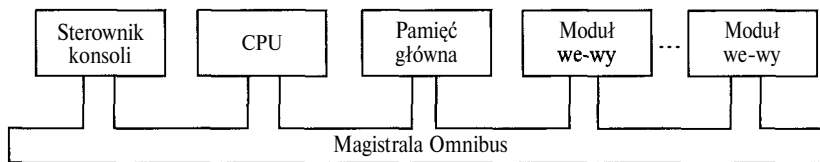
Niska cena i mały rozmiar PDP-8 umożliwiły innym wytwórcom zamawianie go i włączanie do systemu wyższego rzędu z przeznaczeniem na sprzedaż. Wytwórcy ci zostali określani mianem wytwórców sprzętu oryginalnego (ang. *original equipment manufacturers* - OEM), zaś rynek OEM stał się i pozostaje nadal głównym segmentem rynku komputerowego.

Komputer PDP-8 natychmiast stał się hitem, dzięki czemu firma DEC zrobiła fortunę. Maszyna ta, a także inne modele rodziny PDP-8 (patrz tab. 2.5), osiągnęła status produkcyjny zarezerwowany uprzednio dla komputerów IBM. W ciągu następnych 12 lat sprzedano około 50 000 maszyn. Jak podaje oficjalna historia firmy DEC, PDP-8 „określił koncepcję minikomputerów, torując drogę przemysłowi o wartości wielu miliardów dolarów”. Pozwolił również na osiągnięcie przez DEC pozycji pierwszego dostawcy minikomputerów. Kiedy zaś PDP-8 osiągnął kres swojej drogi życiowej, DEC stała się drugim po IBM producentem komputerów.

W przeciwieństwie do architektury opartej na centralnym przełączaniu (rys. 2.5) wykorzystanej przez IBM w systemach 700/7000 i 360, w ostatnich

TABELA 2.5. Ewolucja PDP-8 [VOEL88]

Model	Pierwsza sprze- daż	Koszt procesora + pamięć 4 K słów 12-bitowych (1000 USD)	Szybkość pobierania danych z pamięci [słów/μs]	Objętość (stóp sześcienn- ych)	Innowacje i udoskonalenia
PDP-8	4/65	16,2	1,26	8,0	automatyczne wykonywanie połączeń owijanych (ang. <i>wirewrapping</i>)
PDP-8/5	9/66	8,79	0,08	3,2	wdrożenie rozkazów szeregowych
PDP-8/1	4/68	11,6	1,34	8,0	układy o średniej skali integracji
PDP-8/L	11/68	7,0	1,26	2,0	mniejsza obudowa
PDP-8/E	3/71	4,99	1,52	1,8	Omnibus
PDP-8/M	6/72	3,69	1,52	1,8	o połowę mniejsza obudowa i mniej gniazd w porównaniu z 8/E
PDP-8/A	1/75	2,6	1,34	1,2	pamięć półprzewodnikowa; procesor zmiennopozycyjny



RYSUNEK 2.9. Struktura magistralowa komputera PDP-8

lach PDP-8 zastosowano strukturę, która praktycznie stała się uniwersalna dla minikomputerów i mikrokomputerów: strukturę magistralową. Jest ona przedstawiona na rys. 2.9. Magistrala PDP-8, nazwana **Omnibusem**, składa się z 96 oddzielnych ścieżek sygnałów, używanych do przenoszenia sygnałów sterowania, adresu i danych. Ponieważ wszystkie urządzenia systemu używają wspólnego systemu ścieżek sygnałowych, ich wykorzystywaniem musi sterować procesor. Architektura ta jest wysoce elastyczna, gdyż umożliwia dołączanie różnych modułów do magistrali w celu tworzenia różnych konfiguracji.

Późniejsze generacje

Powyżej trzeciej generacji zgoda w sprawie definiowania generacji komputerów nie jest już pełna. Jak wynika z tabeli 2.2, powstały już generacje czwarta i piąta, wykorzystujące postępy w technologii układów scalonych. Wprowadzenie wielkiej skali integracji (ang. *large-scale integration* - LSI) pozwoliło na umieszczenie ponad 1000 elementów w pojedynczym mikroukładzie. Bardzo wielka

10 000 elementów w mikrostrukturze, a współczesne mikroukłady VLSI mogą zawierać ponad 100 000 elementów.

Przy szybkim postępie technologii, częstym wprowadzaniu nowych wyrobów, a także ważności oprogramowania i łączności zbliżonej do ważności samego sprzętu, klasyfikacja generacji staje się mniej jasna i mniej znacząca. Można stwierdzić, że główne zmiany spowodowane komercyjnym wykorzystaniem nowych rozwiązań nastąpiły we wczesnych latach siedemdziesiątych, wyniki tych zmian są zaś wciąż jeszcze eksploatowane. W tym podrozdziale przedstawiamy dwa najważniejsze spośród tych wyników.

Pamięć półprzewodnikowa

Pierwszym zastosowaniem technologii układów scalonych w komputerach było zbudowanie procesora (jednostki sterującej oraz jednostki arytmetyczno-logicznej) z układów scalonych. Stwierdzono jednak, że ta sama technologia może być użyta do budowy pamięci.

W latach pięćdziesiątych i sześćdziesiątych większość pamięci komputerowych była zbudowana z niewielkich pierścieni z materiału ferromagnetycznego, z których każdy miał średnicę około 1/16 cala (ok. 1,6 mm). Pierścienie te były nawlekane na siatki cienkich drutów, które zawieszano na małych ekranach wewnątrz komputera. Po namagnesowaniu w jednym kierunku pierścień (zwany **rdzeniem**) reprezentował jedynekę; namagnesowanie w przeciwnym kierunku reprezentowało zero. Pamięć na rdzeniach magnetycznych była raczej szybka; odczytanie bitu przechowywanego w pamięci zabierało około jednej milionowej części sekundy. Była jednak droga, duża i wykorzystywała odczyt niszczący; prosty akt odczytania rdzenia wymazywał przechowywaną w nim informację. Było więc konieczne zainstalowanie układów regenerujących dane tuż po ich odczytaniu.

Następnie, w roku 1970, firma Fairchild wyprodukowała pierwszą względnie pojemną pamięć półprzewodnikową. Mikroukład ten, o rozmiarze pojedynczego rdzenia, mógł pamiętać 256 bitów. Odczyt był nieniszczący i znacznie szybszy niż w przypadku rdzeni. Odczytanie bitu zabierało zaledwie 70 miliardowych części sekundy. Jednak koszt przypadający na bit był wyższy niż dla rdzeni.

W roku 1974 zaszło wydarzenie brzemienne w skutki: cena przypadająca na bit pamięci półprzewodnikowych spadła poniżej tej ceny pamięci rdzeniowej. W dalszym ciągu następował ciągły i szybki spadek kosztu pamięci, któremu towarzyszył odpowiedni wzrost fizycznej gęstości pamięci. Utorowało to drogę do mniejszych i szybszych maszyn z pamięciami o rozmiarach takich samych, jakie występowały w większych i droższych komputerach jeszcze kilka lat temu. Rozwój technologii pamięci wraz z rozwojem technologii procesorów przedyskutujemy w dalszej części książki. Zmienił on naturę komputerów w czasie krótszym od dekady. Chociaż wielkie, drogie maszyny pozostały częścią krajobrazu,



komputery stały się dostępne użytkownikowi w postaci maszyn biurowych i komputerów osobistych.

Od roku 1970 pamięci półprzewodnikowe przebyły osiem generacji: 1 K, 4 K, 16 K, 64 K, 256 K, 1 M, 4 M i obecnie 16 M bitów w jednym mikroukładzie ($1\text{ K} = 2^{10}$, $1\text{ M} = 2^{20}$). Każda generacja przyniosła 4-krotny wzrost gęstości pamięci w stosunku do poprzedniej generacji, czemu towarzyszyło zmniejszenie kosztu na bit i zmniejszenie czasu dostępu.

Mikroprocesory

W czasie gdy rosła gęstość elementów w mikroukładach pamięciowych, wzrastała również gęstość elementów w mikroukładach procesorowych. W miarę upływu czasu w każdej mikrostrukturze umieszczano coraz więcej elementów, dzięki czemu do budowy pojedynczego procesora potrzebna była coraz mniejsza liczba mikroukładów.

Przełom osiągnięto w roku 1971, kiedy to w firmie Intel opracowano układ 4004. Był to pierwszy mikroukład zawierający *wszystkie* elementy procesora w jednym mikroukładzie; narodził się **mikroprocesor**.

Mikroprocesor 4004 mógł dodawać dwie liczby 4-bitowe, ale mnożyć mógł tylko przez wielokrotne dodawanie. Według dzisiejszych standardów 4004 był beznadziejnie prymitywny, jednak zapoczątkował ciągłą ewolucję możliwości i mocy mikroprocesorów.

Ewolucję tę najłatwiej przeanalizować, rozważając liczbę bitów, którymi operuje procesor w określonym momencie. Nie istnieje jasna miara tej możliwości, jednak być może najlepszą miarą jest szerokość magistrali danych: liczba bitów danych, które mogą być wprowadzane lub wyprowadzane z procesora w ustalonej chwili. Inną miarą jest liczba bitów w akumulatorze lub w zespole rejestrów ogólnego przeznaczenia. Często miary te są zgodne, jednak nie zawsze. Jest na przykład pewna liczba mikroprocesorów, które operują na liczbach 16-bitowych w rejestrach, ale mogą czytać lub zapisywać w tym samym czasie jedynie 8 bitów.

Następnym głównym krokiem w ewolucji mikroprocesorów było wprowadzenie przez firmę Intel w roku 1972 modelu 8008. Był to pierwszy mikroprocesor 8-bitowy, prawie dwa razy bardziej złożony od 4004.

Żaden z tych kroków nie miał jednak tak istotnego wpływu, jak następne ważne wydarzenie: w roku 1972 Intel wprowadził układ 8080. Był to pierwszy mikroprocesor ogólnego przeznaczenia. Podczas gdy 4004 i 8008 zostały zaprojektowane dla szczególnych zastosowań, 8080 zaprojektowano jako jednostkę centralną mikrokomputera ogólnego przeznaczenia. Podobnie jak 8008, 8080 jest mikroprocesorem 8-bitowym. Jednak 8080 jest szybszy, ma bogatszą listę rozkazów i większe możliwości adresowania.

Prawie równocześnie zaczęto opracowywać mikroprocesory 16-bitowe. Jednak dopiero w końcu lat siedemdziesiątych pojawiły się potężne, 16-bitowe mi-

kroprocesory ogólnego przeznaczenia. Jednym z nich był mikroprocesor 8086. Następny krok w tym rozwoju nastąpił w roku 1981, kiedy to zarówno w Bell Labs, jak i w firmie Hewlett-Packard opracowano jednoukładowy mikroprocesor 32-bitowy. Intel wprowadził swój mikroprocesor 32-bitowy - 80386 - w roku 1985 (tab. 2.6).

TABELA 2.6. Ewolucja mikroprocesorów firmy Intel

Parametr	8008	8080	8086	80386	80486
Rok wprowadzenia	1972	1974	1978	1985	1989
Liczba rozkazów	66	111	133	154	235
Szerokość szyny adresowej	8	16	20	32	32
Szerokość szyny danych	8	8	16	32	32
Liczba znaczników stanu	4	5	9	14	14
Liczba rejestrów	8	8	16	8	8
Adresowalność pamięci	16 KB	64 KB	1 MB	4 GB	4 GB
Porty wejścia-wyjścia	24	256	64 K	64 K	64 K
Szerokość pasma magistrali [MB/s]	–	0,75	5	32	32
Czas dodawania rejestr - rejestr [μ s]	–	1,3	0,3	0,125	0,06

2.2

PROJEKTOWANIE ZORIENTOWANE NA WYDAJNOŚĆ

Rok po roku koszt systemów komputerowych drastycznie malał, podczas gdy wydajność i pojemność tych systemów rosły równie szybko. W wielu sklepach możemy wybrać komputer osobisty w cenie poniżej 1000 dolarów, który bije na głowę komputery IBM sprzed 10 lat. We wnętrzu tego komputera osobistego znajduje się ok. 100 milionów tranzystorów. Nie możemy kupić 100 milionów czegokolwiek tak tanio. Taka liczba listków papieru toaletowego kosztuje ponad 100 000 dolarów.

Otrzymujemy więc moc komputera praktycznie za darmo, a ta nieustająca rewolucja technologiczna umożliwiła rozwój zastosowań o zdumiewającej złożoności i mocy. Na przykład dzisiejsze biurkowe systemy komputerowe wykorzystujące mikroprocesory umożliwiają m.in.:

- Q przetwarzanie obrazów;
- a rozpoznawanie mowy;
- Q wideokonferencje;
- a opracowania multimedialne;
- Q dźwiękowe i wizyjne ilustrowanie plików.

Komputerowe stacje robocze są obecnie wykorzystywane w niezwykle złożonych opracowaniach technicznych i naukowych, a także w systemach symulacyjnych oraz przy grupowym przetwarzaniu obrazów nagrań wideo. Ponadto przedsiębiorcy wykorzystują serwery o rosnącej mocy do obsługi transakcji,

przetwarzania baz danych oraz do tworzenia rozbudowanych sieci klient-serwer, które zastąpiły potężne centra komputerowe z minionych lat.

Z perspektywy rozwoju organizacji i architektury komputerów najbardziej fascynujące jest to, że z jednej strony podstawowe bloki konstrukcyjne dzisiejszych „cudów komputerowych” są praktycznie takie same, jak w komputerze IAS sprzed 50 lat, podczas gdy z drugiej strony metody „wyciskania ostatnich kropli” wydajności z dostępnych materiałów stały się tak wyrafinowane.

Powyższa obserwacja stała się myślą przewodnią tej książki. Podczas rozpatrywania różnych elementów i zespołów komputera mamy na uwadze dwa cele: po pierwsze - wyjaśnienie fundamentalnej funkcjonalności w każdym rozpatrywanym obszarze, po drugie zaś - zbadanie metod osiągania maksymalnej wydajności. W pozostałej części tego podrozdziału przedstawimy pewne czynniki wzmacniające potrzebę projektowania zorientowanego na wydajność.

Szybkość mikroprocesora

Tym, co daje mikroprocesorom Pentium i PowerPC tak wielką, pobudzającą umysł moc, jest nieugięte dążenie producentów struktur procesorowych do szybkości. Ewolucja tych układów nadal wykracza poza granice określone przez prawo Moore’a. Prezes firmy Intel Gordon Moore zauważył w latach sześćdziesiątych, że przez zmniejszanie rozmiarów mikroskopijnych linii tworzących układy tranzystorowe w krzemie o ok. 10% rocznie, producenci mikroukładów mogą wprowadzać nową generację swoich wyrobów co 3 lata, uzyskując 4 razy tyle tranzystorów w mikroukładzie. W przypadku mikroukładów pamięciowych, proces ten doprowadził do 4-krotnego zwiększania pojemności pamięci dynamicznej o swobodnym dostępie (DRAM) - wciąż stanowiącej podstawowe rozwiązanie głównej pamięci komputerów - co każde 3 lata. Natomiast w przypadku mikroprocesorów dodanie nowych układów oraz zwiększenie szybkości wynikające ze zmniejszenia odległości między nimi poprawiało wydajność 4- a nawet 5-krotnie w ciągu każdych 3 lat od czasu wprowadzenia przez firmę Intel w roku 1979 rodziny X86.

Jednak sama tylko szybkość mikroprocesora mogłaby pozostać zaledwie potencjalną możliwością, jeśli nie byłby on „zasilany” ciągłym strumieniem zadań do wykonania w postaci rozkazów komputerowych. Wszystko, co staje na drodze temu ciągłemu napływowi rozkazów, obniża moc procesora. Dlatego właśnie, podczas gdy wytwórcy mikroukładów uczyli się, jak wytwarzać struktury o wciąż większej gęstości, projektanci procesorów musieli wykorzystywać coraz bardziej wymyślne metody „karmienia” tych potworów. Wśród rozwiązań wbudowanych we współczesne procesory znajdują się:

Q Przewidywanie rozgałęzienia (ang. *branch prediction*). Procesor z wyprzedzeniem przegląda programy i przewiduje, jakie rozgałęzienia lub grupy rozkazów mogą być przetwarzane w dalszej kolejności. Jeśli procesor w większości przypadków zgaduje prawidłowo, to może wstępnie pobierać

prawidłowe rozkazy i buforować je, mając w ten sposób ciągle zajęcie. W bardziej wyrafinowanych przykładach zastosowania takiej strategii procesor potrafi przewidywać nie tylko jedno, ale wiele rozgałęzień. Przewidywanie rozgałęzień zwiększa więc ilość pracy dostępnej do wykonania przez procesor.

- a **Analiza przepływu danych.** Procesor analizuje, które rozkazy są zależne od wyników innych rozkazów lub od danych w celu stworzenia optymalnego harmonogramu rozkazów. Rozkazy mogą być uszeregowane do wykonania wcześniej, niezależnie od oryginalnego polecenia wynikającego z programu. Zapobiega to zbędnym opóźnieniom.
- **Spekulatywne wykonywanie rozkazów.** Wykorzystując przewidywanie skoków oraz analizę przepływu danych, niektóre procesory „spekulatywnie” wykonują rozkazy, zanim pojawiają się one w programie, przechowując wyniki w lokacjach tymczasowych. Wykonując rozkazy, które najprawdopodobniej będą potrzebne, procesor lepiej wykorzystuje swój potencjał.

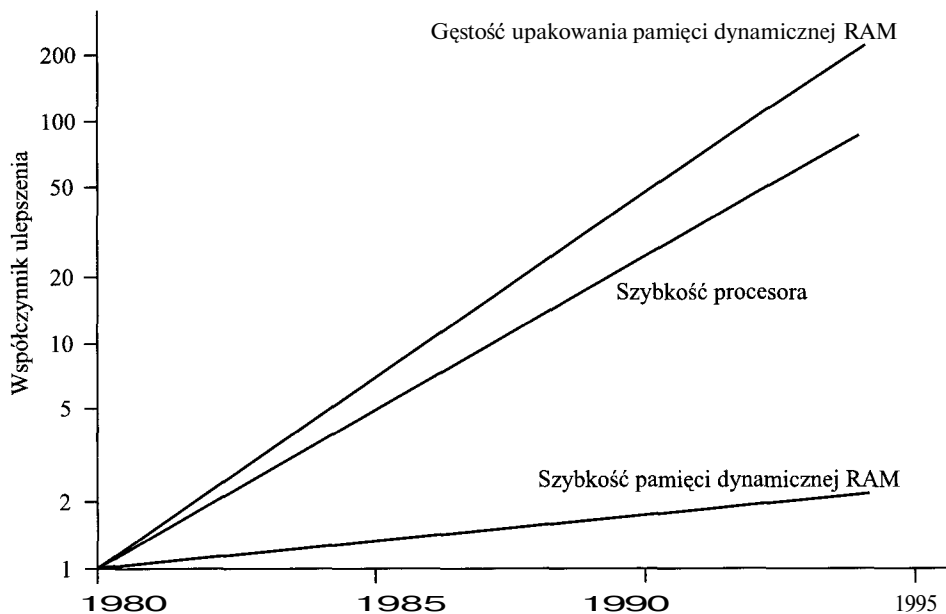
Te i inne wyrafinowane metody stały się konieczne wobec ogromnej mocy procesora. Umożliwiają one wykorzystanie jego szybkości.

Równowaga wydajności

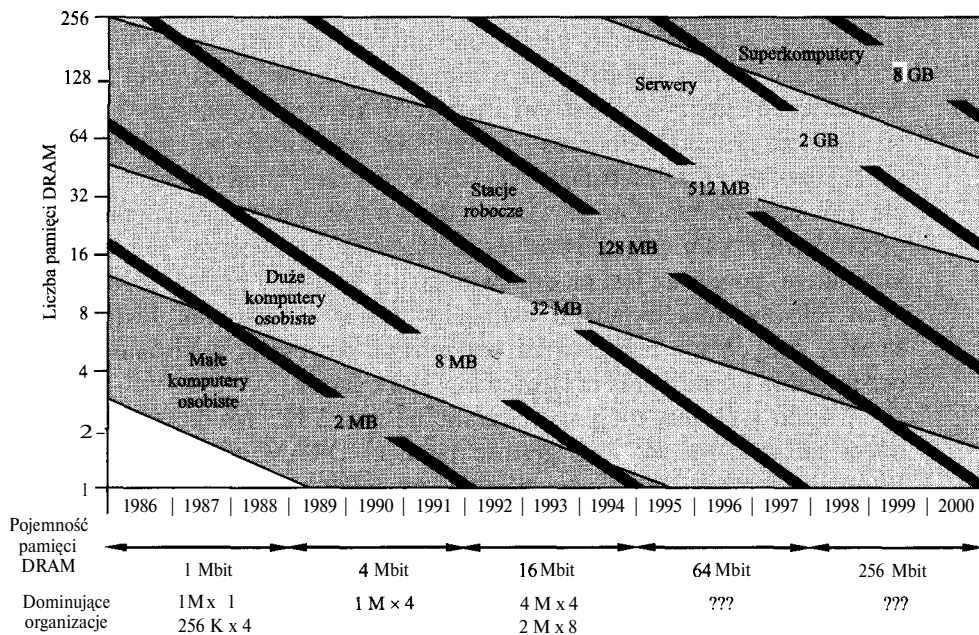
Podczas gdy moc procesorów rosła z zawrotną szybkością, rozwój innych krytycznych zespołów komputera był zbyt wolny. Powstała potrzeba szukania równowagi wydajności: dostosowania organizacji i architektury w celu skompensowania niezgodności między możliwościami różnych zespołów.

Nigdzie indziej problem tego niedopasowania nie jest bardziej krytyczny niż w interfejsie między procesorem a pamięcią główną. Rozważmy historię przedstawioną na rys. 2.11. Podczas gdy szybkość procesora i pojemność pamięci rosły szybko, możliwa do osiągnięcia szybkość przesyłania danych między pamięcią główną a procesorem stawała się daleko niewystarczająca. Interfejs między procesorem a pamięcią główną jest najbardziej krytycznym elementem całego komputera, ponieważ jest on odpowiedzialny za ciągły przepływ rozkazów i danych między mikroukładami pamięciowymi a procesorem. Jeśli pamięć lub ścieżka danych nie nadążają za potrzebami procesora, pozostaje on w stanie oczekiwania i cenny czas przetwarzania jest tracony.

Wyniki tych tendencji są pokazane na rys. 2.12. Wielkość wymaganej pamięci głównej rośnie, lecz gęstość pamięci DRAM rośnie jeszcze szybciej. W rezultacie przeciętna liczba mikroukładów DRAM w systemie maleje. Grube czarne linie na rysunku pokazują, że przy ustalonej pojemności pamięci liczba wymaganych układów DRAM maleje. To jednak ma wpływ na szybkość przesyłania, ponieważ mniejsza liczba układów DRAM oznacza mniej możliwości równoległego przesyłania danych. Zacieniowane pasma pokazują, że dla danego typu systemu rozmiar pamięci głównej rósł powoli, podczas gdy liczba układów DRAM malała.



RYSUNEK 2.11. Ewolucja własności pamięci DRAM i procesorów



RYSUNEK 2.12. Tendencje rozwoju zastosowań pamięci DRAM [PRZY94]

Są sposoby, które architekt systemów może wykorzystać w celu rozwiązania tego problemu. Wszystkie znajdują odbicie we współczesnych projektach komputerów. Oto przykłady:

- ❑ Zwiększanie liczby bitów, które są jednocześnie wyprowadzane, przez projektowanie raczej „szerokich” niż „głębokich” układów DRAM, a także stosowanie szerokich magistrali danych.
- ❑ Stosowanie bardziej efektywnych interfejsów pamięci DRAM przez umieszczanie pamięci podręcznych lub innych układów buforowych w strukturach DRAM.
- a Redukowanie częstości sięgania do pamięci przez wprowadzanie coraz bardziej złożonych i efektywnych struktur pamięci podręcznych między procesorem a pamięcią główną. Do tych rozwiązań należy umieszczanie jednej lub wielu pamięci podręcznych w mikroukładzie procesora, jak również umieszczanie odrębnej pamięci podręcznej w pobliżu struktury procesora.
- ❑ Zwiększanie szerokości pasma przenoszenia między procesorami a pamięcią przez wykorzystanie magistrali o dużej szybkości oraz wprowadzanie hierarchicznego systemu magistrali w celu buforowania i strukturalizowania przepływu danych.

Innym obszarem skupiającym uwagę projektantów są urządzenia wejścia-wyjścia. W miarę wzrostu szybkości i możliwości komputerów powstają bardziej wyrafinowane zastosowania, które mają wysokie wymagania w stosunku do urządzeń wejścia-wyjścia. W tabeli 2.7 są przedstawione przykłady typowych urządzeń peryferyjnych stosowanych z komputerami osobistymi i stacjonarnymi. Urządzenia te stwarzają ogromne wymagania co do przepustowości danych. Podczas gdy procesory współczesnej generacji są w stanie przetwarzać dane „pompowane” przez te urządzenia, pozostaje problem zapewnienia przepływu danych między procesorem a urządzeniami peryferyjnymi. Do stosowanych tu rozwiązań należą: pamięci podręczne i układy buforujące, a także wykorzystanie szybszych i zhierarchizowanych magistrali. Ponadto wymaganiom wejścia-wyjścia może pomóc sprostać konfiguracja wieloprocessorowa.

Kluczową sprawą jest w tym wszystkim równowaga. Projektanci stale dążą do zrównoważenia przepustowości oraz wymagań odnoszących się do przetwa-

TABELA 2.7. Typowa szerokość pasma wymagana dla różnych technologii urządzeń peryferyjnych

Urządzenie peryferyjne	Technologia	Wymagana szerokość pasma [MB/s]
Terminal graficzny	kolor 24-bitowy	30
Sieć lokalna	100BASEX lub FDDI	12
Sterownik dysku	SCSI lub P1394	10
Terminal wizyjny	rozdzielczość 1024 x 768 przy 30 f/s	> 67
Inne urządzenia wejścia-wyjścia	różne	> 5

rzania między procesorem, pamięcią **główną**, urządzeniami wejścia-wyjścia i strukturami połączeń. Projekty muszą być wciąż analizowane od nowa wobec dwóch stale ewoluujących czynników:

1. Szybkości zmian wydajności procesorów, magistrali, pamięci i urządzeń peryferyjnych różnią się znacznie.
2. Nowe zastosowania i nowe urządzenia peryferyjne ciągle zmieniają naturę wymagań w stosunku do systemu wyrażoną poprzez typowy profil rozkazów oraz wzór dostępu do danych.

Tak więc projektowanie komputerów jest sztuką stale ewoluującą. Książka ta ma na celu przedstawienie podstaw tej sztuki oraz jej aktualnego stanu.

2.3

EWOLUCJA PROCESORÓW PENTIUM I POWERPC

W książce tej zostało wykorzystanych wiele konkretnych przykładów projektów komputerów w celu zilustrowania koncepcji oraz wyjaśnienia niezbędnych kompromisów. W większości są to przykłady pochodzące z dwóch rodzin komputerów: Pentium firmy Intel i PowerPC. Pentium jest rezultatem trwającej już od dekady pracy nad komputerami o złożonej liście rozkazów (CISC). Zawarto w nich rozwiązania występujące uprzednio tylko w komputerach stacjonarnych i w **superkomputerach**. Stanowią one znakomite przykłady projektowania CISC. PowerPC jest bezpośrednim potomkiem pierwszego systemu RISC, IBM 801, i jest jednym z najpotężniejszych i najlepiej zaprojektowanych systemów RISC narynku.

W tym punkcie dokonamy krótkiego przeglądu obydwu systemów.

Pentium

Firma Intel przez całe dekady miała rangę pierwszego wytwórcy mikroprocesorów. Możliwość osiągnięcia takiej pozycji wydawała się nieprawdopodobna. Ewolucja „flagowych” mikroprocesorów tej firmy jest dobrym wskaźnikiem ogólnej ewolucji techniki **komputerowej**.

Tabela 2.8 stanowi ilustrację tej ewolucji. Chociaż Pentium jest obecnie **główną gwiazdą** linii wyrobów Intela, w roku 1995 firma wprowadziła nowy procesor P6, a na etapie prac rozwojowych znajduje się jeszcze P7. Interesujące jest, że w miarę jak mikroprocesory stają się szybsze i coraz bardziej złożone, Intel przyspiesza ich rozwój. Zwyczajem firmy było wprowadzanie nowych mikroprocesorów co 4 lata. Jednak już w przypadku Pentium odstęp między generacjami skurczył się do 3 lat. Za pomocą mikroukładów P6 i P7 Intel ma nadzieję trzymać swoich rywali z dala przez skrócenie tego odstęp jeszcze o 1 rok.

TABELA 2.8. Mikroprocesory Intel [HOF95]

Parametr	286	386	486	Pentium	P6	P7
Początek projektowania (rok)	1978	1982	1986	1989	1990	1993
Data wprowadzenia	lut. 1982	paźdz. 1985	kwiec. 1989	marzec 1993	III kw. 1995	1997 lub 1998 ^b
Dostawy w dużych ilościach	1983	1986	1990	1994	1996	1998 lub 1999 ^b
Liczba tranzystorów	130 tys.	275 tys.	1,2 mln	3,1 mln	5,5 mln	>10 mln ^b
Szybkość początkowa (MIPS) ^a	1	5	20	100	250 ^b	500 ^b
Rok maksymalnej sprzedaży	1989	1992	1995 ^b	1997 ^b	1999 ^b	2000 ^b
Liczba zainstalowanych układów	9,7 mln	44,2 mln	75 mln	4,5 mln	–	–

^a MIPS oznacza milion rozkazów na sekundę.^b Oszacowanie.

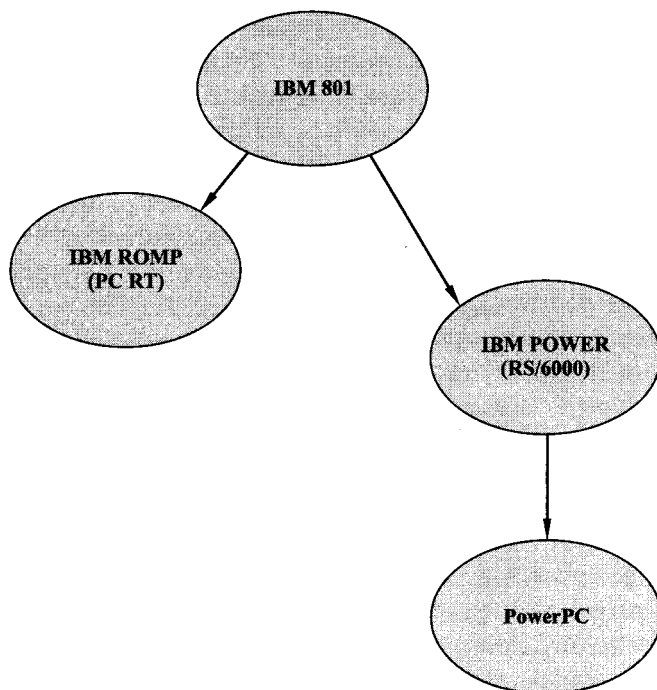
Warto wymienić wybrane elementy ewolucji linii wyrobów Intela:

- a 8080. Pierwszy na świecie mikroprocesor ogólnego przeznaczenia. Było to urządzenie 8-bitowe, z 8-bitową ścieżką danych do pamięci.
- ❑ 8086. Daleko potężniejsze, 16-bitowe urządzenie. Poza szerszą ścieżką danych i większymi rejestrami, mikroprocesor 8086 zawiera podręczną pamięć rozkazów, która wstępnie pobiera kilka rozkazów przed ich wykonaniem.
- ❑ 80286. Stanowi on rozszerzenie mikroprocesora 8086, umożliwiając adresowanie pamięci 16 MB zamiast tylko 1 MB.
- Q 80386. Pierwsze 32-bitowe urządzenie Intela, jednocześnie znaczne odnowienie linii mikroprocesorów. Mając architekturę 32-bitową, mikroprocesor 80386 rywalizował pod względem złożoności i mocy z minikomputerami i komputerami stacjonarnymi wprowadzonymi zaledwie kilka lat wcześniej.
- a 80486. W tym mikroprocesorze wprowadzono znacznie bardziej złożoną i potężną technikę pamięci podręcznej, a także wyrafinowane potoki rozkazów.
- a Pentium. Wraz z Pentium Intel wprowadził do użytku techniki superskalarne, co umożliwiło równoległe wykonywanie wielu rozkazów.
- Q P6. Procesor ten stanowi kontynuację prac nad organizacją superskalarną, które rozpoczął procesor Pentium. Ponadto szeroko jest wykorzystywane przewidywanie rozgałęzień, analiza przepływu danych oraz spekulatywne wykonywanie rozkazów.
- ❑ P7. Przewiduje się wprowadzenie do linii wyrobów technologii opartej na koncepcji RISC.

PowerPC

W roku 1975 w firmie IBM opracowano minikomputer 801, w którym po raz pierwszy wprowadzono wiele koncepcji architektury stosowanych w systemie RISC. Minikomputer 801 wraz z procesorem RISC I z Berkeley zapoczątkował rozwój RISC. Jednakże 801 był tylko prototypem przeznaczonym do zademonstrowania koncepcji projektowych. Sukces minikomputera 801 skłonił IBM do opracowania komercyjnego produktu RISC, pod nazwą RT PC. Maszyna ta nie była sukcesem komercyjnym i miała wielu rywali o porównywalnej i większej wydajności. W roku 1990 IBM wprowadził trzeci system, w którym wykorzystano lekcję 801 i RT PC. RISC System/6000 był zbliżoną do RISC maszyną superskalarną sprzedawaną jako urządzenie stacjonarne o wysokiej wydajności. Wkrótce po jego wprowadzeniu IBM zaczął określać jego architekturę mianem POWER.

Następnie IBM nawiązał współpracę z firmą Motorola, w której opracowano szereg mikroprocesorów 68000, oraz z firmą Apple, która wykorzystywała mikroukłady Motoroli w swoich komputerach Macintosh. Wynikiem współpracy była seria maszyn wykorzystujących architekturę PowerPC. Architektura ta wywodzi się z architektury POWER (rys. 2.13). Dokonano zmian, uzupełniając brakujące własności oraz umożliwiając bardziej efektywne wdrożenie przez wyeliminowanie niektórych rozkazów. Złagodzono pewne wymagania w celu unik-

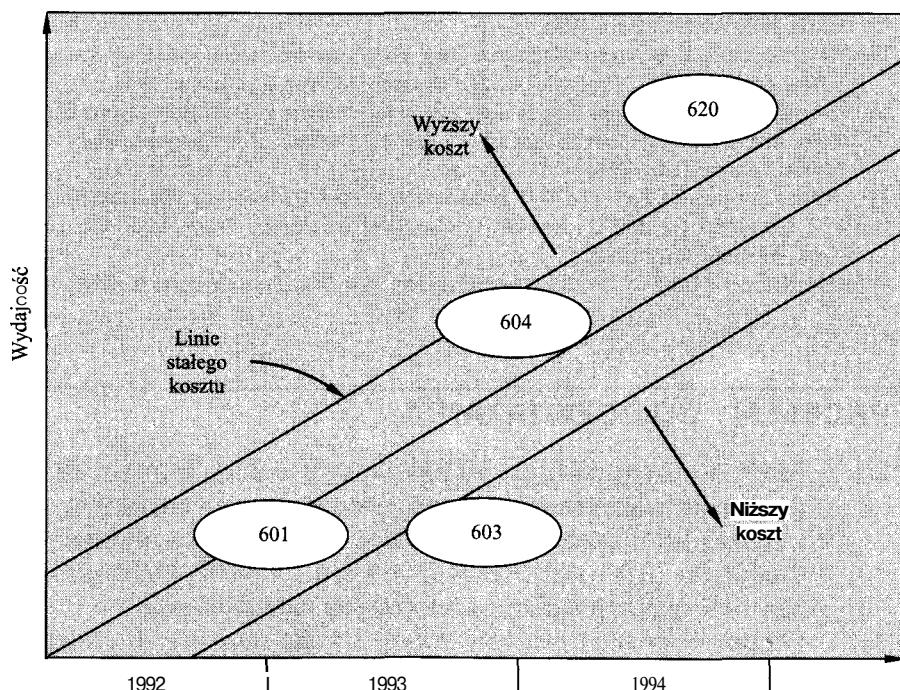


RYSUNEK 2.13. Genealogia PowerPC

nięcia kłopotliwych problemów w niektórych szczególnych przypadkach. Wynikająca stąd architektura PowerPC jest systemem superskalarnym RISC.

Dotychczas opracowano cztery urządzenia należące do rodziny PowerPC (rys. 2.14):

- a 601. Mikroprocesor 601 wyprodukowano, aby wprowadzić na rynek architekturę PowerPC tak szybko, jak było to możliwe. Jest on maszyną 32-bitową.
- a 603. Mikroprocesor przeznaczony dla komputerów biurowych o stosunkowo małej wydajności, przenośnych. Jest to również maszyna 32-bitowa, porównywalna pod względem wydajności z 601, jednak kosztuje mniej, a wdrożenie jej jest bardziej efektywne.
- a 604. Mikroprocesor przeznaczony dla komputerów biurowych oraz serwerów o stosunkowo małej wydajności. Jest to również maszyna 32-bitowa, jednak wykorzystano w niej bardziej zaawansowane rozwiązania superskalarne w celu podniesienia wydajności.
- a 620. Mikroprocesor dla serwerów o większej wydajności. Pierwsze urządzenie z rodziny PowerPC, w którym wykorzystano w pełni 64-bitową architekturę, łącznie z 64-bitowymi rejestrami i ścieżkami danych.



POLECANA LITERATURA

Opis serii IBM 7000 można znaleźć w [BELL71a]. Wyczerpujący opis IBM 360 znajduje się w [SIEW82], a PDP-8 i innych maszyn DEC w [BELL78a]. Te trzy książki zawierają również liczne, szczegółowe przykłady dotyczące innych komputerów i prezentujące historię komputerów we wczesnych latach osiemdziesiątych.

Jednym z najlepszych opisów Pentium jest [ANDE93]. Dobra jest również dokumentacja samego Intela [INTE94a], [INTE94b]. Praca [BREY95] stanowi dobry przegląd linii mikroprocesorów Intela, z akcentem na maszyny 32-bitowe.

W [IBM94] wyczerpująco przedstawiono architekturę PowerPC. W pracy [SHAN94b] podano podobny zakres zagadnień oraz dodatkowo opis mikroprocesora 601. W [WEIS94] znajduje się opis zarówno architektury Power, jak i PowerPC.

ANDE93 Anderson D., Shanley T.: *Pentium Processor System Architecture*. Richardson, Mindshare Press, 1993.

BELL71a Bell C., Newell A.: *Computer Structures: Readings and Examples*. New York, McGraw-Hill, 1971.

BELL78a Bell C., Mudge J., McNamara J.: *Computer Engineering: A DEC View of Hardware Systems Design*. Bedford, Digital Press, 1978.

BREY95 Brey B.: *The Intel 32-bit Microprocessors: 80386, 80486 and Pentium*. Englewood Cliffs, NY, Prentice-Hall, 1995.

IBM94 International Business Machines, Inc.: *The PowerPC Architecture: A Specification for a New Family of RISC Processors*. San Francisco, Morgan-Kaufmann, 1994.

INTE94a Intel Corp.: *Pentium Family User's Manual. Volume J: Data Book*. Santa Clara, 1994.

INTE94b Intel Corp.: *Pentium Family User's Manual. Volume 3: Architecture and Programming Manual*. Santa Clara, 1994.

SHAN94b. Shanley T.: *PowerPC 601 System Architecture*. Richardson, Mindshare Press, 1994.

SIEW82 Siewiorek D., Bell C., Newell A.: *Computer Structures: Principles and Examples*. New York, McGraw-Hill, 1982.

WEIS94 Weiss S., Smith J.: *POWER and PowerPC*. San Francisco, Morgan Kaufmann, 1994.

PROBLEMY DO ROZWIĄZANIA

- 2.1. Niech $A = A(1), A(2), \dots, A(1000)$ oraz $B = B(1), B(2), \dots, B(1000)$ będą dwoma wektorami (tablicami jednowymiarowymi) składającymi się z tysiąca liczb każdy, które mają być dodane w celu utworzenia tablicy C takiej, że $C(I) = A(I) + B(I)$ dla $I = 1, 2, \dots, 1000$. Stosując listę rozkazów IAS, napisz program do rozwiązania tego problemu.
- 2.2. Uzupełnij schemat blokowy z rys. 2.4 dla pozostałych rozkazów IAS z wyjątkiem tych, które zawierają mnożenie lub dzielenie.
- 2.3. W modelach 65 i 75 serii IBM 360 adresy są przechowywane w dwóch oddzielnych jednostkach pamięci głównej (np. wszystkie słowa parzyste

- w jednej jednostce, a nieparzyste w drugiej). Jaki może być cel tego rozwiązania?
- 2.4. Omów wady i zalety przechowywania programów i danych w tej samej pamięci.
 - 2.5. Wymień te rozkazy z tabeli 2.1, które są nadmiarowe (redundantne). Rozkaz jest nadmiarowy, jeśli identyczna funkcja może być zrealizowana przez pewną kombinację innych rozkazów z tej listy. W każdym przypadku należy zademonstrować redundancję przez pokazanie alternatywnego szeregu rozkazów.
 - 2.6. Rozważ względne zalety architektury wykorzystującej centralne przełączanie oraz architektury magistralowej.

SYSTEM KOMPUTEROWY

System komputerowy składa się z procesora, pamięci, urządzeń wejścia-wyjścia oraz połączeń między tymi głównymi zespołami. Z wyjątkiem procesora, który jest dostatecznie złożony, aby poświęcić mu całą część III, w części II przeanalizujemy szczegółowo każdy z tych zespołów.

Na poziomie najwyższym możemy zrozumieć działanie każdego z głównych zespołów, opisując ich połączenia oraz sygnały wymieniane między nimi. Zagadnienia te omówimy w rozdz. 3. Następne dwa rozdziały dotyczą odpowiednio pamięci wewnętrznej i zewnętrznej. W rozdziale 4 jest przedstawiona organizacja pamięci głównej oraz wykorzystanie pamięci podręcznych do zwiększania wydajności. Rozdział 5 zawiera przegląd głównych typów pamięci zewnętrznych i opis organizacji RAID wykorzystywanej do zwiększania wydajności pamięci dyskowych. Rozdział 6 jest poświęcony różnym aspektom organizacji wejścia-wyjścia. Wreszcie w rozdz. 7 są opisane podstawowe zagadnienia dotyczące systemów operacyjnych oraz są przedyskutowane specyficzne rozwiązania sprzętu komputerowego zorientowane na wspieranie systemu operacyjnego.

Magistrale systemowe

Na najwyższym poziomie organizacji komputer składa się z procesora, pamięci i urządzeń wejścia-wyjścia, przy czym każdy z tych modułów może występować pojedynczo lub w większej ilości. Zespoły te są połączone w sposób umożliwiający realizowanie podstawowej funkcji komputera, jaką jest wykonywanie programów. Wobec tego - pozostając na tym najwyższym poziomie - możemy opisać system komputerowy *przez*: (1) przedstawienie zewnętrznego zachowania każdego modułu, to znaczy danych i sygnałów kontrolnych, które wymienia on z innymi modułami, oraz (2) podanie struktury połączeń i sterowania wymaganego do zarządzania tą strukturą.

Obraz struktury i funkcji widziany na najwyższym poziomie organizacji jest ważny, ponieważ ułatwia zrozumienie natury komputera. Równie ważne jest jego wykorzystanie dla zrozumienia wciąż komplikujących się zagadnień oceny wydajności. Uchwycenie struktury i funkcji na tym poziomie pozwala zauważyć wąskie gardła systemu, rozwiązania alternatywne, zasięg uszkodzeń systemu wywołanych przez uszkodzenie zespołu oraz możliwość zwiększenia wydajności. W wielu przypadkach żądaniu większej mocy i odporności na uszkodzenia systemu można sprostać raczej przez zmiany projektowe, niż tylko przez zwiększanie szybkości i niezawodności poszczególnych zespołów.

W tym rozdziale skupimy się na podstawowych strukturach wykorzystywanych do łączenia zespołów komputera. Rozpocniemy od krótkiej analizy podstawowych zespołów i ich wymagań interfejsowych. Następnie dokonamy przeglądu funkcjonalnego. Jesteśmy więc gotowi do przeanalizowania problemu wykorzystywania magistrali do łączenia zespołów komputera.

3.1

ZESPOŁY KOMPUTERA

Jak wykazaliśmy w rozdz. 2, w praktycznie wszystkich współczesnych projektach komputerów wykorzystuje się koncepcje opracowane przez Johna von Neumanna z Institute for Advanced Studies w Princeton. Projekty tego typu są

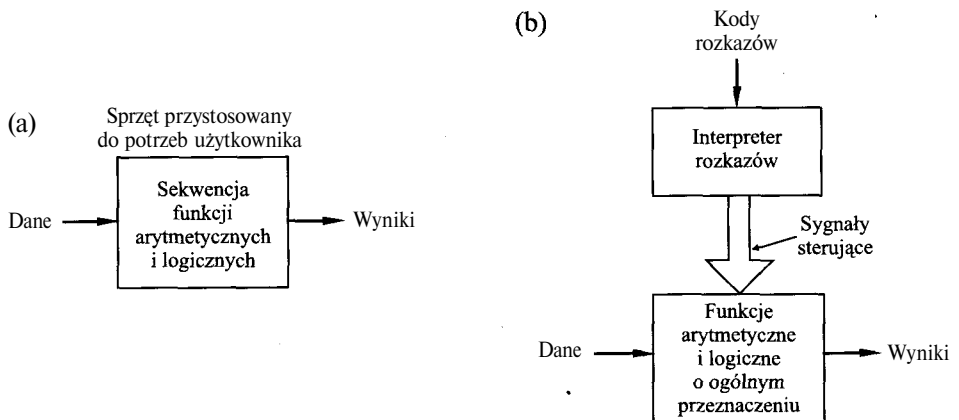
określane jako **architektura von Neumanna**; są w nich wykorzystywane trzy kluczowe koncepcje:

- ❑ Dane i rozkazy są przechowywane w tej samej pamięci umożliwiającej zapis i odczyt.
- ❑ Zawartość tej pamięci może być adresowana przez wskazanie miejsca, bez względu na rodzaj zawartych tam danych.
- ❑ Wykonywanie rozkazów następuje w sposób szeregowy (z wyjątkiem określonych, szczególnych przypadków), rozkaz po rozkazie.

Uwarunkowania leżące u podstaw tych koncepcji przedyskutowaliśmy w rozdz. 1, jednak warto je tutaj podsumować. Istnieje pewien niewielki zestaw podstawowych elementów logicznych, które mogą być łączone na różne sposoby w celu przechowywania danych binarnych oraz wykonywania operacji arytmetycznych i logicznych na tych danych. Jeśli istnieje określony rodzaj obliczenia, które ma być przeprowadzone, to można zbudować specyficzną dla tego obliczenia konfigurację elementów logicznych. Możemy widzieć proces łączenia tych elementów w żadaną konfigurację jako formę programowania. Wynikający stąd „program” ma postać sprzętu i jest nazywany **programem sprzętowym** (ang. *hardwired program*).

Jeśli całe programowanie byłoby realizowane w ten sposób, sprzęt miałby niewielki zakres zastosowań. Rozpatrzmy tę alternatywę. Założmy, że budujemy zbiór funkcji arytmetycznych i logicznych o ogólnym przeznaczeniu. Urządzenie będzie realizowało różne operacje na danych, zależnie od doprowadzonych sygnałów sterujących. W oryginalnym przypadku urządzenia zaprojektowanego do wykonywania konkretnego zadania system przyjmuje dane i dostarcza wyniki (rys. 3.1a).

Natomiast w przypadku urządzenia o ogólnym przeznaczeniu system przyjmuje dane i sygnały sterujące, po czym dostarcza wyniki. Zamiast więc prze-



RYSUNEK 3.1. Rozwiązania sprzętowe i programowe: (a) rozwiązania sprzętowe; (b) rozwiązania programowe

budowywać urządzenie dla każdego nowego programu, programista musi tylko dostarczyć nowy zestaw sygnałów sterujących.

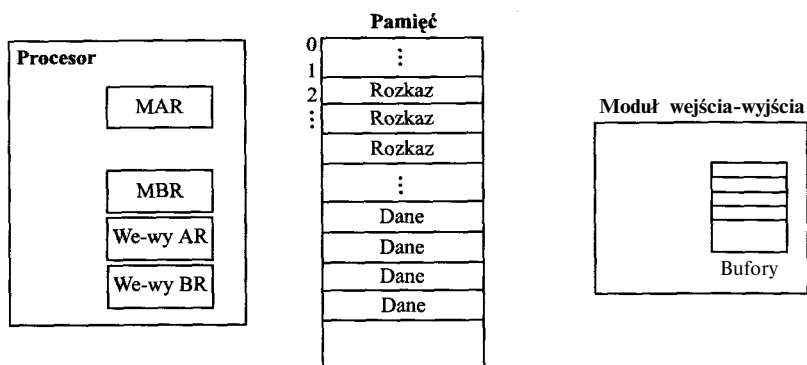
Jak mogą być dostarczane sygnały sterujące? Odpowiedź jest prosta, lecz zawiera subtelności. Cały program jest szeregiem (sekwencją) kroków. W każdym z tych kroków jest wykonywana na danych pewna operacja arytmetyczna lub logiczna. Dla każdego kroku jest wymagany nowy zestaw sygnałów sterujących. Spróbujmy teraz przypisać unikalny kod każdemu z możliwych zestawów sygnałów sterujących i dodajmy do urządzenia o ogólnym przeznaczeniu segment, który przyjmuje kod i generuje sygnały sterujące (rys. 3.1b).

Programowanie jest teraz znacznie łatwiejsze. Zamiast zmieniać połączenia w sprzęcie, musimy jedynie doprowadzić nową sekwencję kodów. Każdy kod jest w rezultacie rozkazem, zaś odpowiednia część urządzenia interpretuje każdy rozkaz i generuje sygnały sterujące. W celu odróżnienia tej nowej metody programowania sekwencja kodów lub rozkazów nazywana jest **oprogramowaniem** (ang. *software*).

Na rysunku 3.1 b są pokazane dwa główne składniki systemu: moduł interpretujący rozkazy oraz moduł realizujący funkcje arytmetyczne i logiczne. Oba razem **tworzą jednostkę centralną**. Aby uzyskać działający komputer, potrzeba jeszcze paru innych zespołów. Do systemu muszą być wprowadzane dane i rozkazy. Do tego celu potrzebujemy pewnego rodzaju modułu wejściowego. Zawiera on podstawowe podzespoły, które przyjmują dane i rozkazy w pewnej formie, po czym dokonują ich konwersji na wewnętrzną postać sygnałów używaną w systemie. Potrzebne jest też urządzenie prezentujące wyniki, będące modułem wyjściowym. Razem są one określane jako **moduły wejścia-wyjścia**.

Potrzebny jest jeszcze jeden zespół. Urządzenie wejściowe doprowadza dane i rozkazy sekwencyjnie. Jednak program nie zawsze jest realizowany sekwencyjnie; może zawierać skoki (np. rozkaz skoku IAS). Podobnie operacje na danych mogą wymagać dostępu do więcej niż jednego elementu w tym samym czasie, w z góry określonej sekwencji. Musi więc istnieć miejsce do czasowego przechowywania zarówno rozkazów, jak i danych. Odpowiedni moduł nazywa się **pamięcią** lub **pamięcią główną** w odróżnieniu od pamięci zewnętrznej lub pamięci występujących w urządzeniach peryferyjnych. Von Neumann zauważył, że ta sama pamięć może służyć zarówno do przechowywania rozkazów, jak i danych. Dane są traktowane jako dane, na których są przeprowadzane obliczenia. Natomiast rozkazy są traktowane jako dane stanowiące kody do generowania sygnałów sterujących.

Na rysunku 3.2 są pokazane zespoły komputera na najwyższym poziomie organizacji oraz oddziaływanie między nimi. Procesor wymienia dane z pamięcią. Do tego celu są wykorzystywane dwa rejestry wewnętrzne (w stosunku do procesora): rejestr adresowy pamięci (MAR), określający adres w pamięci następnego zapisu lub odczytu, oraz rejestr buforowy pamięci (MBR) zawierający dane, które mają być zapisane w pamięci lub dane odczytane z pamięci. Podobnie rejestr adresowy wejścia-wyjścia (I/OAR) określa konkretne urządzenie



RYSUNEK 3.2. Zespoły komputera: struktura ogólna

wejścia-wyjścia. Rejestr buforowy wejścia-wyjścia jest wykorzystywany do wymiany danych między modułem wejścia-wyjścia a jednostką centralną.

Moduł pamięci składa się z zestawu miejsc komórek, określonych przez sekwencyjnie ponumerowane adresy. Każde z miejsc zawiera liczbę binarną, która może być zinterpretowana albo jako rozkaz, albo jako dane. Moduł wejścia-wyjścia przenosi dane z urządzeń zewnętrznych do procesora i pamięci oraz w kierunku przeciwnym. Zawiera wewnętrzne bufory do czasowego przechowywania danych do momentu, aż będą one mogły być wysłane.

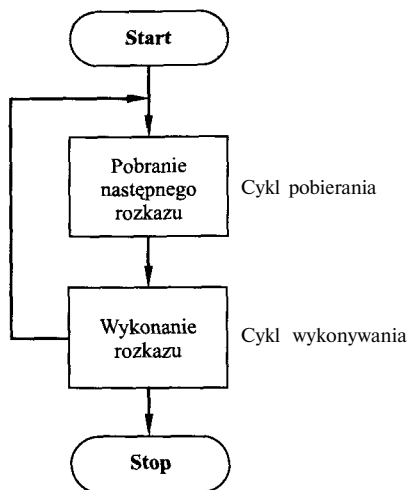
Po tym krótkim przeglądzie głównych zespołów rozpatrzmy ich współdziałanie mające na celu wykonywanie programów.

3.2

DZIAŁANIE KOMPUTERA

Podstawowym zadaniem komputera jest wykonywanie programu. Program przeznaczony do wykonania składa się z zestawu rozkazów przechowywanych w pamięci. Jednostka centralna realizuje pracę, wykonując rozkazy wyszczególnione w programie.

Aby lepiej zrozumieć to zadanie oraz sposób współdziałania zespołów komputera podczas realizacji programu, musimy bardziej szczegółowo przeanalizować proces wykonywania programu. Najprostsze jest rozpatrywanie przetwarzania rozkazów w dwóch krokach: procesor odczytuje (*pobiera*) rozkaz z pamięci, a następnie wykonuje go. Realizacja programu polega na powtarzaniu procesu pobierania i wykonywania rozkazu. Oczywiście, wykonywanie rozkazu może zawierać pewną liczbę kroków (widać to np. w dolnej części rys. 2.4). Na tym etapie możemy podzielić przetwarzanie rozkazów na pobieranie i wykonywanie w sposób następujący. Pobieranie rozkazu jest powszechną operacją, dotyczy każdego rozkazu i polega na odczytaniu rozkazu z pamięci. Natomiast wykonywanie rozkazu może zawierać kilka operacji i zależy od natury rozkazu.



RYSUNEK 3.3. Podstawowy cykl rozkazu

Przetwarzanie wymagane dla pojedynczego rozkazu jest nazywane **cyklem rozkazu**. Jest on pokazany na rys. 3.3, przy czym wykorzystano 2-etapowe ujęcie przedstawione powyżej. Te dwa etapy są określane jako **cykl pobierania** i **cykl wykonywania**. Wykonywanie programu jest wstrzymywane tylko po wyłączeniu maszyny, po wystąpieniu pewnego rodzaju nieodwracalnego błędu lub jeśli wystąpi w programie *rozkaż* zatrzymania komputera.

Cykle pobierania i wykonywania

Na początku każdego cyklu rozkazu procesor pobiera rozkaz z pamięci. W typowym procesorze do śledzenia, który rozkaz ma być pobrany, służy rejestr zwany **licznikiem programu** (PC). Jeśli procesor nie otrzyma innego polecenia, to powoduje inkrementację (elementarny przyrost stanu) licznika PC po każdym pobraniu rozkazu i wykonuje następny rozkaz w ciągu (to znaczy rozkaz zlokalizowany w pamięci pod najbliższym adresem o kolejnym wyższym numerze). Rozpatrzmy na przykład komputer, w którym każdy *rozkaż* zajmuje jedno 16-bitowe słowo w pamięci. Załóżmy, że licznik programu jest ustawiony na pozycji 300. Procesor pobierze rozkaz z pozycji 300. W następnych cyklach rozkazy będą pobierane z miejsc 301, 302, 303 itd. Jak już wyjaśniliśmy, sekwencja ta może być zmieniana.

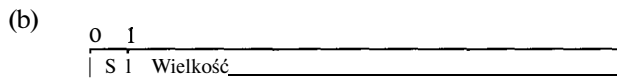
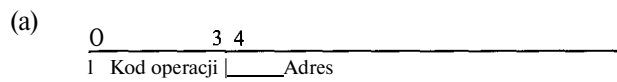
Pobrany rozkaz jest następnie ładowany do rejestru w procesorze zwanego **rejestrem rozkazu** (IR). Rozkaz ten ma formę kodu binarnego określającego działanie, które ma podjąć procesor. Procesor interpretuje *rozkaż* i przeprowadza wymagane działanie. Ogólnie działania te można podzielić na cztery kategorie:

- Q Procesor–pamięć. Dane mogą być przenoszone z procesora do pamięci lub z pamięci do procesora.
- a Procesor–wejście–wyjście. Dane mogą być przenoszone z otoczenia lub do niego, przez przenoszenie ich między procesorem a modułem wejścia–wyjścia.
- a Przetwarzanie danych. Procesor może wykonywać pewne operacje arytmetyczne lub logiczne na danych.
- a Sterowanie. Rozkaz może określać, że sekwencja wykonywania ma być zmieniona (np. rozkaz skoku IAS, tab. 2.1). Na przykład: procesor może pobrać rozkaz z pozycji 149, z którego wynika, że następny rozkaz ma być pobrany z pozycji 182. Procesor zapamięta ten fakt przez ustawienie licznika programu na 182. Dzięki temu w następnym cyklu pobrania rozkaz zostanie pobrany z pozycji 182, a nie 150.

Oczywiście wykonywanie rozkazów może zawierać kombinacje tych działań.

Rozpatrzmy prosty przykład, posługując się hipotetyczną maszyną, której własności są przedstawione na rys. 3.4. Procesor zawiera akumulator (AC) służący do czasowego przechowywania danych. Zarówno rozkazy, jak i dane są słowami 16-bitowymi. Wygodnie jest więc zorganizować pamięć przy użyciu pozycji (lub słów) 16-bitowych. Format rozkazu wskazuje, że występuje $2^4 = 16$ różnych kodów operacji, a liczba słów w pamięci możliwych do bezpośredniego zaadresowania wynosi $2^{12} = 4096$ (4 K).

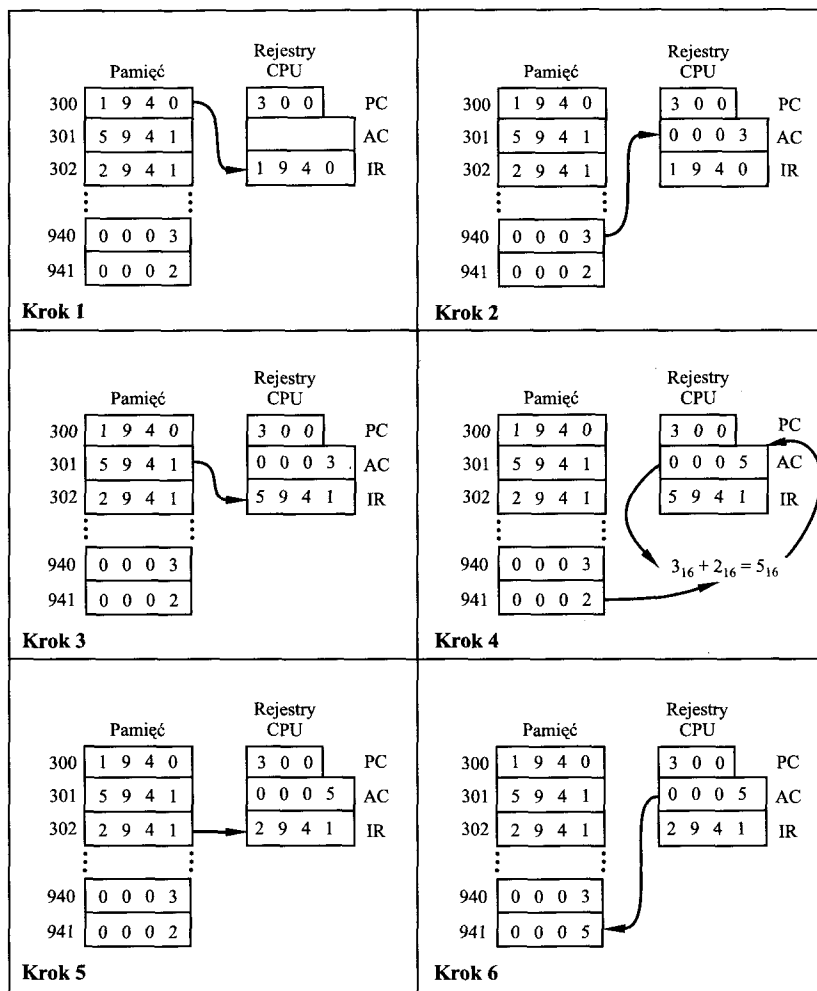
Na rysunku 3.5 jest pokazana częściowa realizacja programu z uwzględnieniem odpowiednich części pamięci i rejestrów procesora.



- (c)
- Licznik rozkazów (PC) = adres rozkazu
 Rejestr rozkazów (IR) = wykonywany rozkaz
 Akumulator (AC) = tymczasowe przechowywanie

- (d)
- 0001 = Ładuj AC z pamięci
 0010 = Zapisz AC w pamięci
 0101 = Dodaj z pamięci do AC

RYSUNEK 3.4. Własności hipotetycznego komputera: (a) format rozkazu; (b) format liczby całkowitej; (c) wewnętrzne rejestry procesora; (d) częściowa lista kodów operacji



RYSUNEK 3.5. Przykład wykonywania programu

Zastosowano notację szesnastkową¹. Pokazany fragment programu polega na dodaniu zawartości słowa pamięci pod adresem 940_{16} do zawartości słowa zapisanego pod adresem 941_{16} oraz na zapisaniu wyniku pod tym drugim adresem. Wymagane są 3 rozkazy, które mogą być opisane jako 3 cykle pobrania i 3 cykle wykonania:

1. Licznik programu zawiera liczbę 300 - adres pierwszego rozkazu. Adres ten jest ładowany do rejestru rozkazów (IR). Zauważmy, że ten proces anga-

¹ W notacji szesnastkowej każdą cyfrę reprezentują 4 bity. Jest to najwygodniejsza notacja do prezentowania zawartości pamięci i rejestrów, gdy długość słowa jest wielokrotnością 4 (np. 8, 16 lub 32). Czytelnikowi, któremu ta notacja nie jest znana, przedstawimy ją w rozdz. 8.

- zuje rejestr adresowy pamięci (MAR) i rejestr buforowy pamięci (MBR). Dla uproszczenia te pośrednie rejestry są pominięte.
2. Pierwsze 4 bity rejestru IR wskazują, że ma być ładowany akumulator (AC). Pozostałe 12 bitów precyzuje adres, w tym przypadku 940.
3. Stan licznika PC jest inkrementowany i jest pobierany następny rozkaz.
4. Poprzednia zawartość akumulatora i zawartość pozycji 941 są dodawane; wynik zaś jest przechowywany w akumulatorze.
5. Stan licznika PC jest zwiększany i jest pobierany następny rozkaz.
6. Zawartość akumulatora jest zapisywana w pozycji 941.

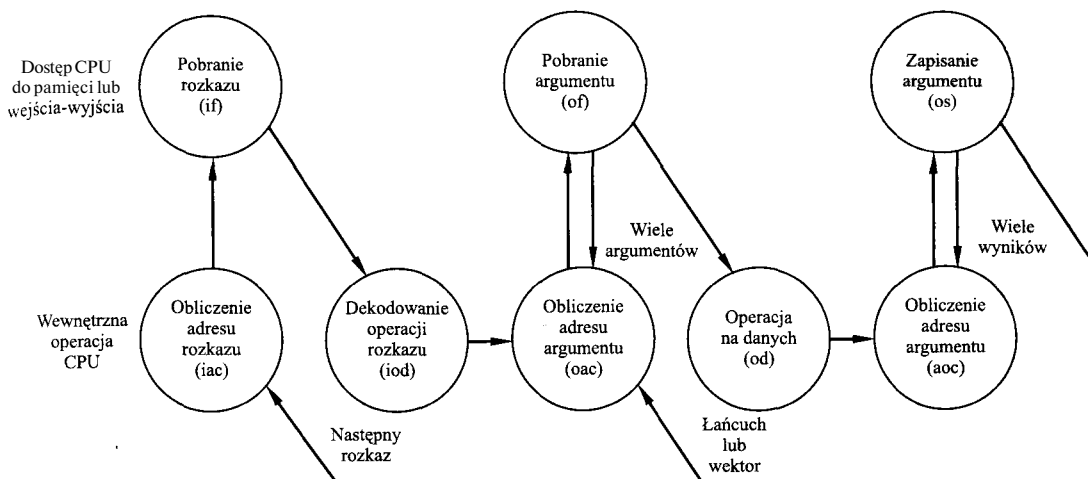
W tym przykładzie do dodania zawartości pozycji 940 do zawartości pozycji 941 są wymagane są trzy cykle rozkazu, z których każdy składa się z cyklu pobierania i cyklu wykonywania. Przy bardziej złożonym zestawie rozkazów liczba wymaganych cykli byłaby mniejsza. Najnowocześniejsze procesory posługują się rozkazami zawierającymi więcej niż jeden adres. Rozkaz komputer PDP-11 wyrażany symbolicznie jako ADD B,A (Dodaj B,A) powoduje zapisanie sumy zawartości pamięci B i A w pozycji A. Wykonywany jest w tym celu pojedynczy cykl rozkazu zawierający następujące kroki:

1. Pobranie rozkazu ADD.
2. Wczytanie zawartości pozycji A z pamięci do procesora.
3. Wczytanie zawartości pozycji B z pamięci do procesora. Żeby zachować zawartość A, procesor musi mieć przynajmniej dwa rejestry do przechowywania danych z pamięci, a nie tylko pojedynczy akumulator.
4. Dodanie obu wartości.
5. Przepisanie wyniku z procesora do pozycji pamięci A.

Tak więc cykl wykonania określonego rozkazu może wykorzystywać więcej niż jedno odniesienie do pamięci. Zamiast odnosić się do pamięci, rozkaz może precyzować operacje wejścia-wyjścia. Mając na uwadze te dodatkowe rozważania, na rys. 3.6 jest przedstawiony dokładniejszy obraz podstawowego cyklu rozkazu z rys. 3.3.

Rysunek ma postać wykresu stanów. Dla danego cyklu rozkazu pewne stany mogą być zerowe (mogą nie występować), inne zaś mogą występować częściej niż raz. Stany te mogą być następujące:

- *Obliczanie adresu rozkazu (iac).* Określenie adresu następnego rozkazu przeznaczonego do wykonania. Zwykle polega to na dodaniu ustalonej liczby do adresu poprzedniego rozkazu. Jeśli na przykład każdy rozkaz zawiera 16 bitów, a organizacja pamięci przewiduje słowa 16-bitowe, to dodaje się 1 do poprzedniego adresu. Jeśli w pamięci przewidziano bezpośrednie adresowanie 8-bitowych bajtów, to dodaje się 2 do poprzedniego adresu.
- *Pobieranie rozkazu (if).* Wczytanie rozkazu z pamięci do procesora.



RYSUNEK 3.6. Graf stanów cyklu rozkazu

- ❑ *Dekodowanie operacji rozkazu (iod)*. Analizowanie rozkazu w celu określenia rodzaju operacji, która ma być przeprowadzona, oraz argumentu (argumentów).
- Q *Obliczanie adresu argumentu (oac)*. Określanie adresu argumentu, jeśli operacja odnosi się do argumentu znajdującego się w pamięci lub dostępnego przez wejście-wyjście.
- ❑ *Pobieranie argumentu (of)*. Pobranie argumentu z pamięci lub z wejścia-wyjścia.
- ❑ *Operacja na danych (do)*. Przeprowadzenie operacji wskazanej w rozkazie.
- Q *Przechowanie argumentu (os)*. Zapisanie wyniku w pamięci lub skierowanie go do wejścia-wyjścia.

Stany w górnej części rys. 3.6 określają wymianę między procesorem a pamięcią lub modułem wejścia-wyjścia. Stany z dolnej części rysunku zawierają tylko wewnętrzne operacje procesora. Stan oac występuje dwukrotnie, ponieważ rozkaz może zawierać odczyt, zapis lub jedno i drugie. Jednak operacja wykonywana w tym stanie jest w zasadzie taka sama w obu przypadkach, potrzebny jest zatem tylko identyfikator jednostanowy.

Zauważmy także, że wykres umożliwia wprowadzanie wielu argumentów i uzyskiwanie wielu wyników, ponieważ wymagają tego pewne rozkazy w niektórych maszynach. Na przykład w PDP-11 wynikiem rozkazu ADD A,B jest następująca sekwencja stanów: iac, if, iod, oac, of, oac, of, do, oac, os.

Wreszcie, w niektórych maszynach pojedynczy rozkaz może określać operację na wektorze (jednowymiarowej tablicy) liczb lub na szeregu (jednowymiarowej tablicy) znaków. Jak widać na rys. 3.6, wymaga to powtarzalnych operacji pobrania i (lub) przechowywania.

Przerwania

Praktycznie we wszystkich komputerach przewidziano mechanizm, za pomocą którego inne moduły (wejście-wyjście, pamięć) mogą przerwać normalne przetwarzanie danych przez procesor. W tabeli 3.1 są wymienione najczęściej występujące klasy przerwań. Szczególną naturą tych przerwań zajmiemy się później, zwłaszcza w rozdz. 6 i 11. Musimy jednak wprowadzić tę koncepcję teraz, aby lepiej zrozumieć naturę cyklu rozkazu oraz implikacje dotyczące struktury połączeń, wynikające z istnienia przerwań. Na tym etapie nie musimy być zainteresowani szczegółami generowania i przetwarzania przerwań, natomiast możemy się skupić tylko na komunikacji między modułami, wynikającej z przerwań.

TABELA 3.1. Klasy przerwań

Programowe	Generowane przez warunek będący wynikiem wykonywania rozkazu, taki jak przepełnienie arytmetyczne, dzielenie przez zero, próba wykonania niedozwolonego rozkazu oraz odniesienie do przestrzeni pamięci zarezerwowanej dla użytkownika.
Zegarowe	Generowane przez wewnętrzny zegar procesora; umożliwia wykonanie pewnych funkcji przez system operacyjny.
Wejścia-wyjścia	Generowane przez sterownik wejścia-wyjścia w celu zasygnalizowania normalnego zakończenia operacji lub w celu zasygnalizowania różnych warunków błędu.
Uszkodzenie sprzętu	Generowane przez uszkodzenie, takie jak defekt zasilania lub błąd parzystości pamięci.

Przerwania były pierwotnie przewidywane jako sposób poprawiania efektywności przetwarzania. Na przykład wiele urządzeń zewnętrznych jest o wiele wolniejszych od procesora. Załóżmy, że procesor przenosi dane do drukarki, stosując schemat cyklu rozkazu przedstawiony na rys. 3.3. Po każdej operacji zapisu procesor musi pozostawać bezczynny, aż drukarka za nim nadaży. Długość tej pauzy może wynosić setki lub nawet tysiące cykli rozkazu, które nie angażują pamięci. Jest to oczywiście bardzo rozrzutne wykorzystanie procesora. Za pomocą przerwań procesor może być zaangażowany w wykonywanie innych rozkazów, podczas gdy ma miejsce operacja wejścia-wyjścia.

Na rysunku 3.7a widać ten właśnie stan w odniesieniu do zastosowania przedstawionego w poprzednim akapicie. Program użytkowy przewiduje szereg wezwań „zapisz” (*write*) na przemian z przetwarzaniem. Segmenty kodów 1, 2 i 3 odnoszą się do sekwencji rozkazów, które nie angażują wejścia-wyjścia. Wezwania „zapisz” są informacją dla programu wejścia-wyjścia, że system jest dostępny, więc może on przeprowadzić aktualną operację wejścia-wyjścia. Program wejścia-wyjścia składa się z trzech sekcji:

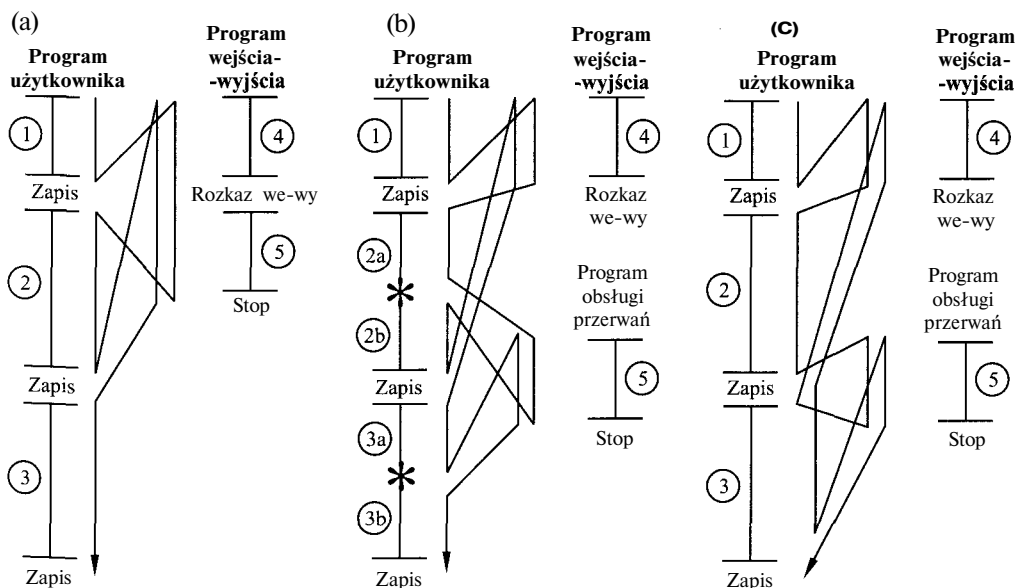
- ❑ Sekwencji rozkazów, oznaczonej na rysunku jako 4, mającej na celu przygotowanie do aktualnej operacji wejścia-wyjścia. Może ona zawierać kopiowanie danych do specjalnego bufora oraz przygotowywanie parametrów dla rozkazu.

- a Aktualnego rozkazu wejścia-wyjścia. Jeśli nie wykorzystuje się przerwań, to wydanie tego rozkazu powoduje, że program musi czekać, aż urządzenie wejścia-wyjścia zrealizuje wymaganą funkcję. Oczekiwanie to może polegać na prostym, powtarzającym się testowaniu w celu stwierdzenia, czy operacja wejścia-wyjścia jest zakończona.
- a Sekwencji rozkazów oznaczonej na rysunku jako 5, mającej na celu zamknięcie operacji. Może ona zawierać przekazanie znacznika wskazującego sukces lub niepowodzenie operacji.

Ponieważ operacja wejścia-wyjścia może zająć stosunkowo długi czas, program wejścia-wyjścia jest zawieszany do czasu zakończenia operacji. W wyniku tego program użytkowy jest zatrzymywany na wezwaniu „zapisz” przez dość długi czas.

Przerwania i cykl rozkazu

Przy wykorzystaniu przerwań procesor może być angażowany w wykonywanie innych rozkazów w czasie, gdy jest realizowana operacja wejścia-wyjścia. Rozważmy przebieg sterowania przedstawiony na rys. 3.7b. Jak poprzednio, program użytkowy osiąga punkt, w którym wysyła wywołanie systemowe w postaci wezwania „zapisz”. Program wejścia-wyjścia, który jest wywoływany w tym momencie, składa się tylko z kodu przygotowania i aktualnego rozkazu



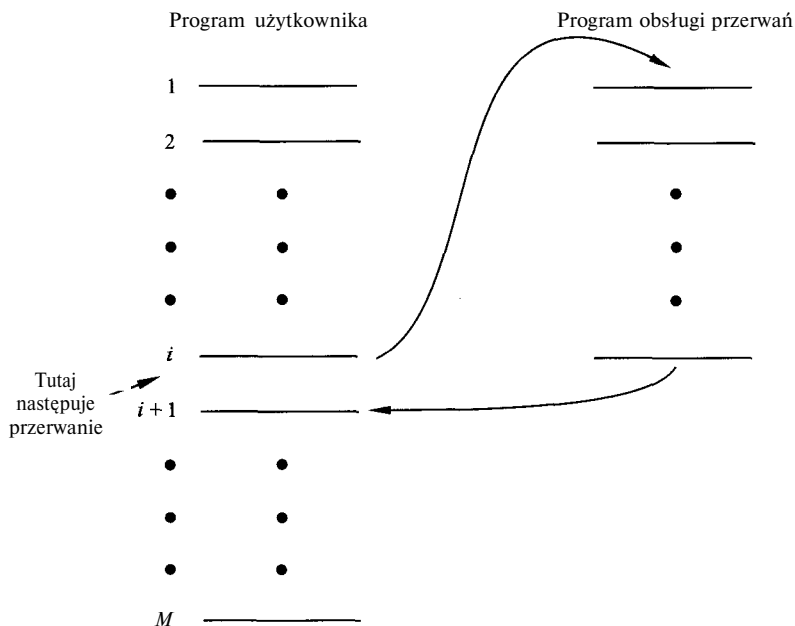
RYSUNEK 3.7. Programowy przepływ sterowania bez przerwań oraz z przerwaniami; (a) bez przerwań; (b) z przerwaniami; krótkie oczekiwanie na wejście-wyjście; (c) z przerwaniami; długie oczekiwanie na wejście-wyjście

wejścia-wyjścia. Po wykonaniu tych kilku instrukcji sterowanie wraca do programu użytkowego. Tymczasem urządzenie zewnętrzne zajmuje się pobieraniem danych z pamięci komputera i drukowaniem ich. Ta operacja wejścia-wyjścia jest przeprowadzana współbieżnie w stosunku do rozkazów zawartych w programie użytkowym.

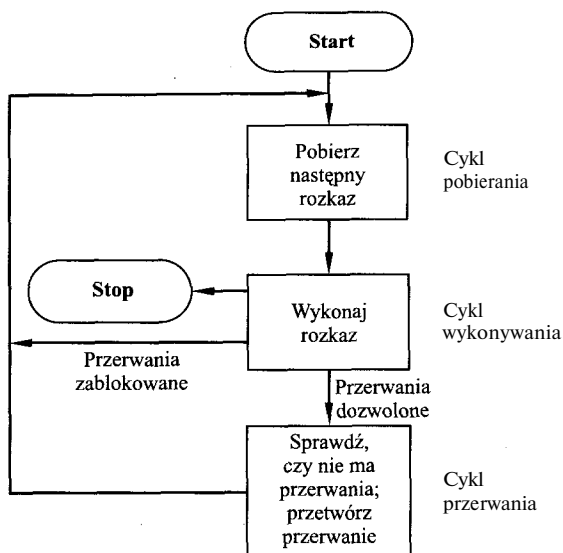
Gdy urządzenie zewnętrzne staje się gotowe do obsługi, to znaczy gdy jest gotowe do przyjmowania następnych danych z procesora, moduł wejścia-wyjścia związany z tym urządzeniem wysyła sygnał **żądania przerwania** do procesora. Procesor odpowiada, zawieszając działanie bieżącego programu i wykonując skok do programu obsługującego to urządzenie zewnętrzne, nazywanego **programem obsługi przerwania** (ang. *interrupt handler*). Po obsłudze urządzenia następuje powrót do programu bieżącego. Punkty, w których następują takie przerwy, oznaczono gwiazdkami (*) na rys. 3.7b.

Z punktu widzenia programu użytkowego przerwanie jest właśnie przerwaniem normalnej sekwencji pracy. Gdy przetwarzanie przerwania jest zakończone, następuje powrót do normalnej sekwencji (rys. 3.8). Program użytkowy nie musi więc zawierać jakiegoś specjalnego kodu w związku z przerwaniami; procesor i system operacyjny są odpowiedzialne za zawieszenie programu użytkowego, a następnie powodują powrót do tego samego punktu.

Aby dostosować się do przerw, do cyklu rozkazu jest dodawany **cykl przerwania** w sposób pokazany na rys. 3.9. Podczas cyklu przerwania procesor sprawdza, czy nie nastąpiły jakieś przerwy, czego świadectwem byłaby obec-



RYSUNEK 3.8. Przekazywanie sterowania za pomocą przerw



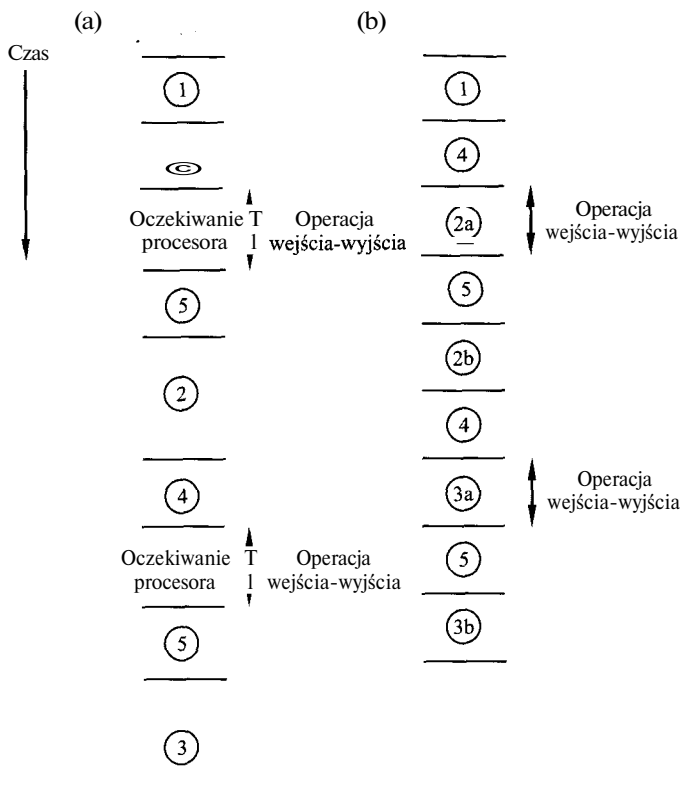
RYSUNEK 3.9. Cykl rozkazu z przerwaniem

ność sygnału przerwania. Jeśli przerwanie nie są realizowane, procesor przechodzi do cyklu pobrania i pobiera następny rozkaz z bieżącego programu. Jeśli natomiast następuje przerwanie, procesor wykonuje następujące czynności:

1. Zawiesza wykonywanie bieżącego programu i zachowuje jego kontekst. Polega to na pisaniu adresu następnego rozkazu przewidywanego do wykonania (tj. bieżącej zawartości licznika programu), a także innych danych związanych z bieżącym działaniem procesora.
2. Ustawia licznik programu na początkowy adres programu obsługi przerw.

Procesor przechodzi następnie do cyklu pobrania i pobiera pierwszy rozkaz z programu obsługi przerw. Program ten jest na ogół częścią systemu operacyjnego. Zwykle program obsługi określa naturę przerwania, po czym podejmuje niezbędne działania. W wykorzystywanym przez nas przykładzie program określa, który moduł wejścia-wyjścia wygenerował przerwanie, po czym może przeskoczyć do programu, który przekaże więcej danych temu modułowi. Po zakończeniu programu obsługi przerwania procesor może wznowić wykonywanie programu użytkowego w punkcie jego przerwania.

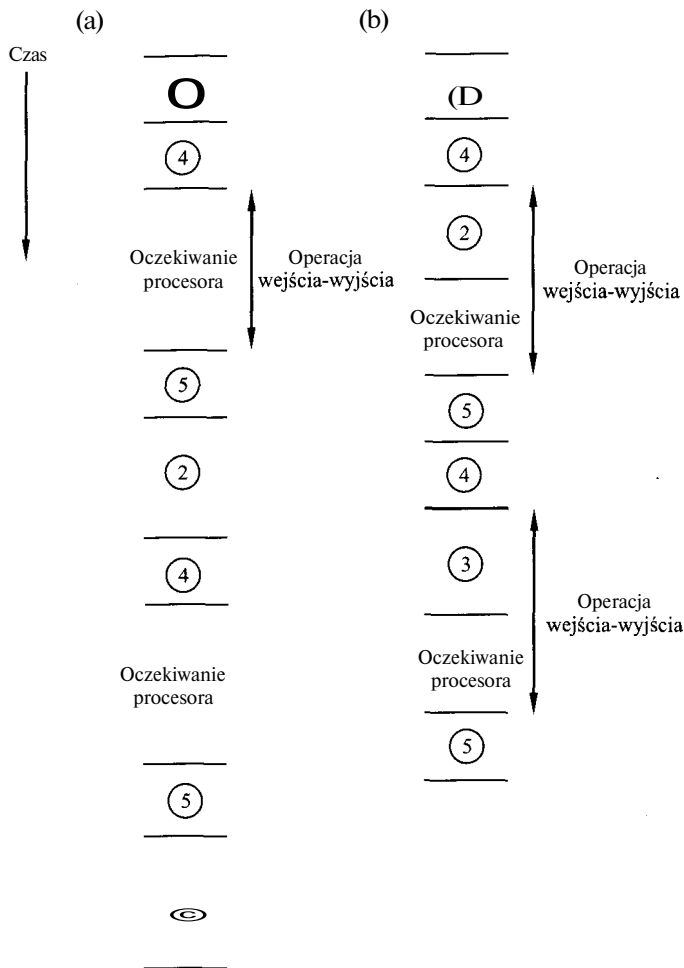
Jest jasne, że w tym procesie występują pewne dodatkowe czynności. Muszą być wykonane dodatkowe rozkazy (w ramach programu obsługi przerw) w celu określenia natury przerwania i podjęcia decyzji w sprawie niezbędnych działań. Ponieważ jednak stosunkowo długi czas byłby tracony na oczekiwanie na zakończenie operacji wejścia-wyjścia, wykorzystanie przerw umożliwia bardziej efektywną pracę procesora.



RYСУNEK 3.10. Taktowanie programu; krótkie oczekiwanie na wejście-wyjście: (a) bez przerw; (b) z przerwami

Żeby docenić poprawę efektywności, przyjrzyjmy się rys. 3.10, na którym jest pokazany wykres czasowy odnoszący się do przebiegów sterowania z rys. 3.7a i 3.7b.

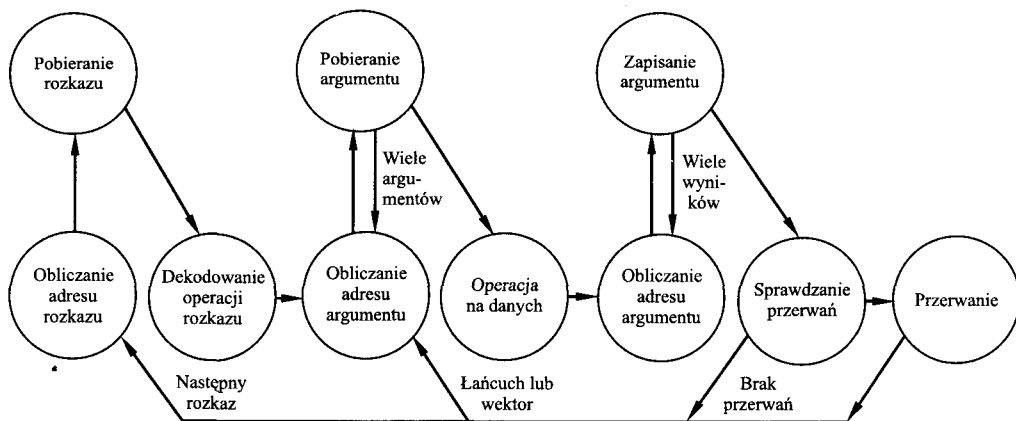
Na rysunkach 3.7b i 3.10 przyjęto, że czas wymagany na wykonanie operacji wejścia-wyjścia jest stosunkowo krótki: krótszy mianowicie od czasu wykonywania rozkazów przewidzianych między operacjami „zapisz” (*write*) w programie użytkowym. Częściej jednak operacja wejścia-wyjścia zabiera znacznie więcej czasu, niż wykonanie sekwencji rozkazów z programu użytkowego, co dotyczy zwłaszcza tak powolnego urządzenia, jak drukarka. Na rysunku 3.7c widać tę właśnie sytuację. W tym przypadku program użytkowy osiąga następne wezwanie „zapisz”, zanim jeszcze operacja wejścia-wyjścia wywołana przez poprzednie wezwanie została ukończona. W rezultacie program użytkowy jest w tym punkcie zawieszany. Po zakończeniu poprzedniej operacji wejścia-wyjścia może być przetwarzane nowe wezwanie „zapisz” i rozpoczyna się nowa operacja wejścia-wyjścia. Na rysunku 3.11 jest pokazany harmonogram dla tej wła-



RYSUNEK 3.11. Taktowanie programu; długie oczekiwanie na wejście-wyjście: (a) bez przerwań; (b) z przerwaniami

śnie sytuacji, z wykorzystaniem i bez wykorzystania przerwań. Widzimy, że wciąż jeszcze występuje poprawa efektywności, ponieważ część czasu wykonywania operacji wejścia-wyjścia nakłada się z czasem wykonywania rozkazów użytkownika.

Na rysunku 3.12 jest pokazany zrewidowany wykres stanów odnoszący się do cyklu rozkazu z uwzględnieniem przetwarzania cyklu przerwania.



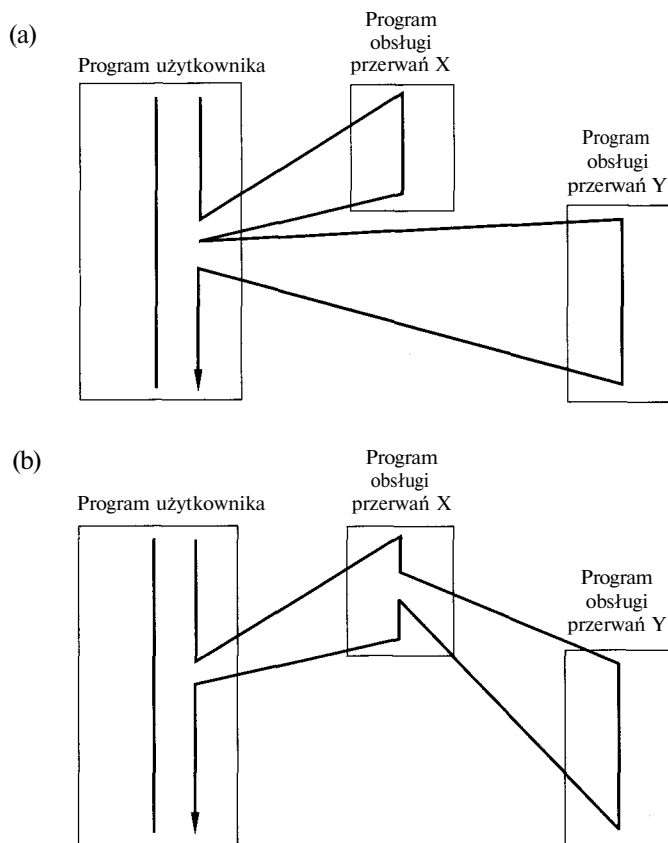
RYSUNEK 3.12. Graf stanów cyklu rozkazu z przerwaniami

Przerwania wielokrotne

W dotychczasowej dyskusji rozważaliśmy występowanie tylko pojedynczego przerwania. Załóżmy jednak, że mogą wystąpić przerwania wielokrotne. Program może, na przykład, otrzymywać dane z łącza komunikacyjnego a także sygnały dotyczące drukowania. Drukarka będzie generowała przerwanie każdorazowo po zakończeniu operacji drukowania. Sterownik łącza komunikacyjnego wygeneruje przerwanie każdorazowo po przybyciu jednostki danych. Jednostka ta może być pojedynczym znakiem lub blokiem, zależnie od przyjętych reguł komunikacji. W każdym przypadku możliwe jest, że przerwanie wywołane przez ten sterownik nastąpi w czasie, w którym jest przetwarzane przerwanie spowodowane przez drukarkę.

Problem wielokrotnych przerwań może być rozwiązany na dwa sposoby. Pierwszy to uniemożliwienie przerwań, jeśli jakiekolwiek przerwanie jest właśnie przetwarzane. **Przerwanie zablokowane** (ang. *disabled interrupt*) oznacza po prostu, że procesor zignoruje sygnał żądania przerwania. Jeśli żądanie przerwania nastąpi w tym właśnie czasie, na ogół pozostaje ono zawieszone i zostanie wykryte przez procesor, jeśli uzyska on zezwolenie na przerwanie. Jeżeli jest więc wykonywany program użytkowy i następuje przerwanie, to natychmiast uniemożliwiane są inne przerwania. Kiedy program obsługi przerwań kończy działanie, przerwania stają się dozwolone jeszcze przed wznowieniem programu użytkowego, a procesor sprawdza, czy wystąpiły dodatkowe przerwania. Rozwiązanie to jest przyjemne i proste, jeśli przerwania są realizowane w porządku ściśle sekwencyjnym (rys. 3.13a).

Ujemną stroną powyższego podejścia jest to, że nie bierze się pod uwagę względnej ważności przerwań oraz zadań, które krytycznie zależą od czasu. Jeśli

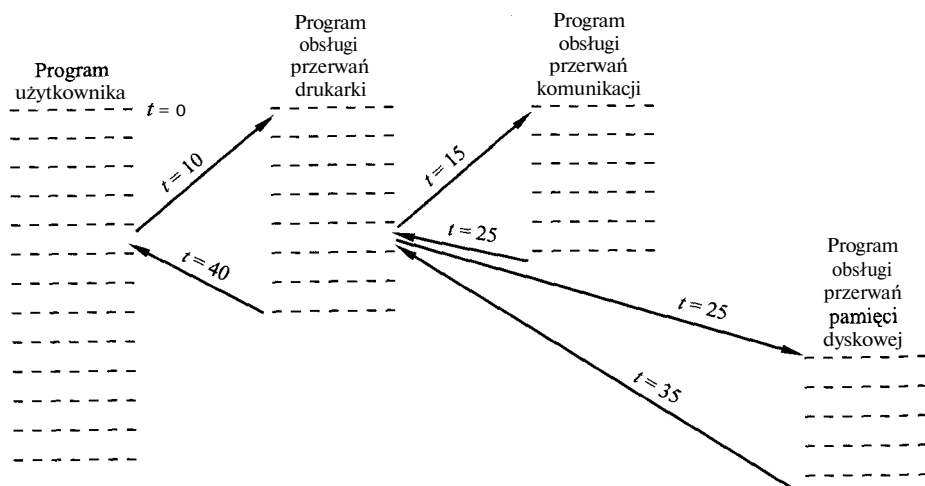


RYSUNEK 3.13. Przekazywanie sterowania przy wielu przerwań: (a) sekwencyjne przetwarzanie przerwań; (b) przetwarzanie zagnieżdżonych przerwań

na przykład przybywają dane z łącza komunikacyjnego, to może być konieczne ich szybkie przyjęcie w celu umożliwienia doprowadzenia następnych danych. Jeśli pierwsza porcja danych nie byłaby przetworzona przed przybyciem następnej, dane mogłyby zostać utracone.

Drugie podejście polega na określeniu priorytetów przerwań. Pozwala się na to, że przerwanie o wyższym priorytecie powoduje przerwanie programu obsługi przerwań o niższym priorytecie (rys. 3.13b).

Jako przykład tego drugiego rozwiązania, rozważmy system z trzema urządzeniami wejścia-wyjścia: drukarką, pamięcią dyskową i łączem komunikacyjnym, o rosnących kolejno priorytetach 2, 4 i 5. Na rysunku 3.14 jest pokazana jedna z możliwych sekwencji. Program użytkowy rozpoczyna się w chwili $t = 0$. W chwili $t = 10$ następuje przerwanie wywołane przez drukarkę; informacje użytkownika są umieszczane na stosie systemowym, dalsza praca odbywa się zgodnie z programem obsługi przerwań drukarki. W chwili $t = 15$, gdy program ten jest nadal realizowany, następuje przerwanie wywołane przez sterownik



RYСУNEK 3.14. Przykład sekwencji czasowej wielu przerwań

łącza komunikacyjnego. Ponieważ linia ta ma wyższy priorytet niż drukarka, honorowane jest nowe przerwanie. Program obsługi przerwań drukarki ulega przerwaniu, stan danych jest kierowany na stos, po czym następuje wykonanie programu obsługi przerwań łącza komunikacyjnego. Podczas realizacji tego programu następuje przerwanie wywołane przez pamięć dyskową ($t = 20$). Ponieważ to nowe przerwanie ma niższy priorytet, jest po prostu zatrzymywane, a program obsługi przerwań łącza komunikacyjnego jest nadal wykonywany, aż do zakończenia.

Po zakończeniu programu obsługi przerwań łącza komunikacyjnego ($t = 25$), odtwarzany jest poprzedni stan procesora, w którym wykonywany jest program obsługi przerwań drukarki. Zanim jednak może być wykonany pierwszy rozkaz z tego programu, procesor honoruje mające wyższy priorytet przerwanie pamięci dyskowej, a sterowanie ulega przeniesieniu do programu obsługi przerwań pamięci dyskowej. Dopiero po zakończeniu tego programu ($t = 35$) wznowiany jest program obsługi przerwań drukarki. Kiedy z kolei ten program jest zakończony ($t = 40$), sterowanie powraca w końcu do programu użytkownika.

Działanie urządzeń wejścia-wyjścia

Dotychczas rozważaliśmy działanie komputera jako sterowane przez jednostkę centralną, skupiając się przede wszystkim na współpracy procesora i pamięci. Czyniliśmy zaledwie aluzję do roli urządzeń wejścia-wyjścia. Zajmiemy się tym szczegółowo w rozdz. 6, tutaj przedstawimy je jedynie pokrótce.

Moduł wejścia-wyjścia może wymieniać dane bezpośrednio z procesorem. Podobnie jak procesor może inicjować odczyt lub zapis w pamięci, określając adres specyficznej lokacji. Procesor może też odczytywać dane pochodzące z modułu wejścia-wyjścia lub je w nim zapisywać. W tym ostatnim przypadku

procesor identyfikuje urządzenie, którym steruje określony moduł wejścia-wyjścia. Następnie może być realizowana sekwencja rozkazów o formie podobnej do tej z rys. 3.5, przy czym rozkazy odnoszą się do wejścia-wyjścia zamiast do pamięci.

W pewnych przypadkach pożądanym jest umożliwienie bezpośredniej wymiany danych między wejściem-wyjściem a pamięcią. W takim przypadku procesor przekazuje modułowi wejścia-wyjścia prawo do odczytywania lub zapisywania rozkazów w pamięci, dzięki czemu przesyłanie danych między wejściem-wyjściem a pamięcią może następować bez angażowania procesora. Podczas takiego przesyłania moduł wejścia-wyjścia odczytuje lub zapisuje rozkazy w pamięci, uwalniając procesor od odpowiedzialności za tę wymianę. Operacja taka jest znana jako **bezpośredni dostęp do pamięci** (ang. *direct memory access* - DMA). Zajmujemy się nią szczegółowo w rozdz. 6. Teraz musimy tylko wiedzieć, że struktura połączeń komputera powinna umożliwiać bezpośrednią współpracę pamięci z modułami wejścia-wyjścia.

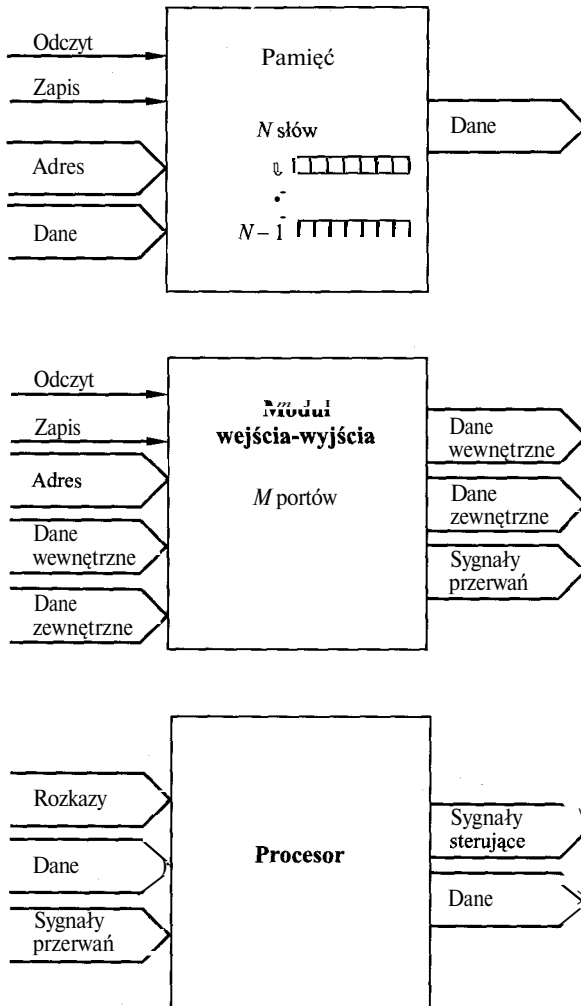
STRUKTURY POŁĄCZEŃ

Komputer jest zestawem zespołów lub modułów trzech podstawowych typów (procesor, pamięć, wejście-wyjście), które komunikują się wzajemnie. W rezultacie komputer jest siecią obejmującą podstawowe moduły. Muszą więc istnieć ścieżki łączące moduły.

Zbiór ścieżek łączących moduły jest nazywany **strukturą połączeń**. Projektowanie tej struktury zależy od wymiany, która musi zachodzić między modułami.

Na rysunku 3.15 są podane rodzaje wymiany, jakie mogą być potrzebne, z uwzględnieniem podstawowych form wejścia i wyjścia dla każdego typu modułu.

- **Pamięć.** Moduł pamięci składa się zwykle z N słów o jednakowej długości. Każde słowo ma przypisany jednoznaczny adres numeryczny (0, 1, ..., $N - 1$). Słowo może być odczytane z pamięci lub do niej zapisane. Rodzaj operacji jest wskazywany przez sygnały sterujące „czytaj” lub „zapisz”. Lokalizacja, której dotyczy operacja, jest wskazywana przez adres.
- **Moduł wejścia-wyjścia.** Z zewnętrznego (w stosunku do systemu komputerowego) punktu widzenia, moduł wejścia-wyjścia jest funkcjonalnie podobny do pamięci. Istnieją dwie operacje: zapisu i odczytu. Ponadto moduł wejścia-wyjścia może sterować więcej niż jednym urządzeniem zewnętrznym. Możemy określić każdy z interfejsów z urządzeniem zewnętrznym jako **port** i nadać każdemu z nich jednoznaczny adres (np. 0, 1, ..., $M - 1$).



RYSUNEK 3.15. Moduły komputera

Istnieją poza tym zewnętrzne ścieżki danych służące do wprowadzania i wyprowadzania danych z urządzenia zewnętrznego. Wreszcie moduł wejścia-wyjścia może wysyłać sygnały przerwania do procesora.

- *Procesor.* Procesor wczytuje rozkazy i dane, wysyła dane po przetworzeniu i posługuje się sygnałami sterującymi do sterowania całą pracą systemu. Otrzymuje też sygnały przerwania.

Powyższa lista określa dane podlegające wymianie. Struktura połączeń musi umożliwiać przesyłanie danych:

- *Z pamięci do procesora.* Procesor odczytuje z pamięci rozkazy lub jednostki danych.

- *Z procesora do pamięci.* Procesor zapisuje jednostki danych w pamięci.
- *Z urządzeń wejścia-wyjścia do procesora.* Procesor odczytuje dane z urządzenia wejścia-wyjścia za pośrednictwem modułu wejścia-wyjścia.
- a *Z procesora do wejścia-wyjścia.* Procesor wysyła dane do urządzenia wejścia-wyjścia.
- *Z urządzeń wejścia-wyjścia do pamięci lub na odwrót.* W tych dwóch przypadkach zezwala się modułowi wejścia-wyjścia na bezpośrednią wymianę danych z pamięcią, bez pośrednictwa procesora, przy wykorzystaniu bezpośredniego dostępu do pamięci (DMA).

W przeszłości wypróbowano wiele struktur połączeń. Zdecydowanie najpowszechniejsze są struktury magistralowe i wielomagistralowe. Pozostała część tego rozdziału jest poświęcona analizie struktur magistralowych.

3.4

POŁĄCZENIA MAGISTRALOWE

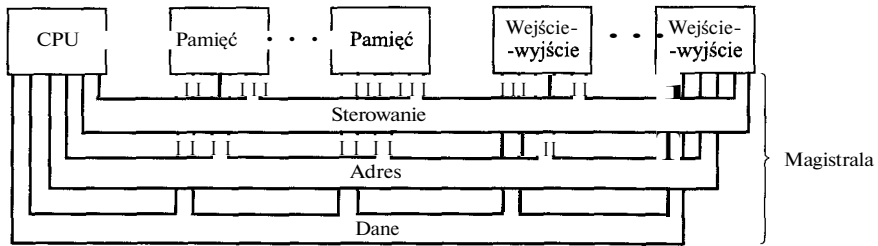
Magistrala jest drogą zapewniającą komunikację między urządzeniami. Główną cechą charakterystyczną magistrali jest to, że jest ona wspólnym nośnikiem transmisji (ang. *shared transmission medium*). Do magistrali dołącza się wiele urządzeń, a sygnały wysyłane przez którekolwiek z nich mogą być odbierane przez wszystkie pozostałe urządzenia. Jeśli dwa urządzenia nadawałyby w tym samym czasie, ich sygnały nakładałyby się i ulegały zakłócaniu. W określonym czasie może więc nadawać tylko jedno urządzenie.

Często magistrala składa się z wielu dróg (linii) komunikacyjnych. Każdą linią mogą być przesyłane sygnały reprezentujące binarne 0 i 1. W ciągu pewnego czasu przez pojedynczą linię może być przekazana sekwencja cyfr binarnych. Kilka linii zawartych w magistrali można wykorzystywać razem do jednoczesnego (równoległego) transmitowania cyfr binarnych. Na przykład 8-bitowa jednostka danych może być przesyłana przez 8 linii magistrali.

System komputerowy zawiera pewną liczbę różnych magistrali, które łączą zespoły komputera na różnych poziomach hierarchii. Magistrala łącząca główne zespoły komputera (procesor, pamięć, wejście-wyjście) nazywana jest **magistralą systemową**. Najczęściej spotykane struktury połączeń komputera wykorzystują jedną lub więcej magistrali systemowych.

Struktura magistrali

Magistrala systemowa składa się zwykle z 50 do 100 oddzielnych linii. Każdej linii jest przypisane określone znaczenie lub funkcja. Chociaż występuje wiele różnych rozwiązań magistrali systemowych, zawarte w nich linie można podzielić na trzy grupy funkcjonalne (rys. 3.16): linie danych, adresów i stereo-



RYSUNEK 3.16. Schemat połączenia magistralowego

wania. Ponadto mogą występować linie służące do zasilania dołączonych modułów.

Linie danych są ścieżkami służącymi do przenoszenia danych między modułami systemu. Wszystkie te linie łącznie są określane jako **szyna danych** (ang. *data bus*). Szyna danych składa się typowo z 8, 16 lub 32 oddzielnych linii, przy czym liczba linii określa **szerokość** tej szyny. Ponieważ w danym momencie każda linia może przenosić tylko 1 bit, z liczby linii wynika, ile bitów można jednocześnie przetransmitować. Szerokość szyny danych jest kluczowym czynnikiem określającym wydajność całego systemu. Jeśli na przykład szyna danych ma szerokość 8 bitów, a każdy rozkaz ma długość 16 bitów, to procesor musi łączyć się z modułem pamięci dwukrotnie w czasie każdego cyklu rozkazu.

Linie adresowe są wykorzystywane do określania źródła lub miejsca przeznaczenia danych przesyłanych magistralą. Jeśli na przykład procesor ma zamiar odczytać słowo (8, 16 lub 32 bity) danych z pamięci, umieszcza adres potrzebnego słowa na linii adresowej. Jest jasne, że szerokość szyny adresowej determinuje maksymalną możliwą pojemność pamięci systemu. Ponadto linie adresowe są również używane do adresowania portów wejścia-wyjścia. Najczęściej najbardziej znaczące bity służą do wybrania określonego modułu na magistrali, natomiast najmniej znaczące bity określają lokalizację w pamięci lub port wejścia-wyjścia wewnątrz modułu. W przypadku szyny 8-bitowej adres 01111111 i niższe mogą na przykład oznaczać lokalizację w module pamięci (moduł 0), a adres 10000000 i wyższe odnoszą się do urządzeń dołączonych do modułu wejścia-wyjścia (moduł 1).

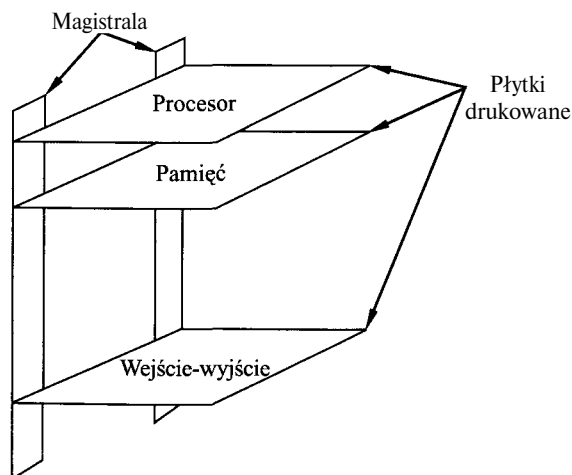
Linii sterowania używa się do sterowania dostępem do linii danych i linii adresowych, a także do sterowania ich wykorzystaniem. Ponieważ linie danych i adresowe służą wszystkim zespołom, musi istnieć sposób sterowania ich używaniem. Sygnały sterujące przekazywane między modułami systemu zawierają zarówno rozkazy, jak i informacje regulujące czas (taktujące). Sygnały czasowe określają ważność danych i adresów. Sygnały rozkazów precyzują operacje, które mają być przeprowadzone. Typowe linie sterowania to linie:

- ❑ **Zapis w pamięci.** Sprawia, że dane z magistrali zostają zapisane pod określonym adresem.

- ❑ **Odczyt z pamięci.** Sprawia, że dane spod określonego adresu są umieszczane w magistrali.
- a **Zapis do wejścia-wyjścia.** Sprawia, że dane z magistrali są kierowane do zaadresowanego portu wejścia-wyjścia.
- a **Odczyt z wejścia-wyjścia.** Sprawia, że dane z zaadresowanego portu wejścia-wyjścia są umieszczane na magistrali.
- a **Potwierdzenie przesyłania (*transfer ACK*).** Wskazuje, że dane zostały przyjęte z magistrali lub na niej umieszczone.
- ❑ **Zapotrzebowanie na magistralę (*bus request*).** Wskazuje, że moduł zgłasza zapotrzebowanie na przejęcie sterowania magistralą.
- a **Rezygnacja z magistrali (*bus grani*).** Wskazuje, że moduł rezygnuje ze sterowania magistralą.
- ❑ **Żądanie przerwania (*interrupt request*).** Wskazuje, że przerwanie jest zawieszone.
- a **Potwierdzenie przerwania (*interrupt ACK*).** Potwierdza, że zawieszone przerwanie zostało rozpoznane.
- ❑ **Zegar.** Wykorzystywany do synchronizowania operacji.
- a **Przywrócenie (*reset*).** Ustawia wszystkie moduły w stanie początkowym.

Działanie magistrali jest następujące. Jeśli jeden z modułów zamierza wysłać dane do drugiego, to musi wykonać dwie rzeczy: (1) uzyskać dostęp do magistrali i (2) przekazać dane za pośrednictwem magistrali. Jeśli natomiast zamierza uzyskać dane z innego modułu, to musi: (1) uzyskać dostęp do magistrali i (2) przekazać zapotrzebowanie do tego modułu przez odpowiednie linie sterowania i adresowe. Musi następnie czekać, aż drugi moduł wyśle dane.

Fizycznie magistrala systemowa jest zbiorem równoległych połączeń elektrycznych. Połączenia te są ścieżkami wytrawionymi w obwodzie drukowanym.



RYSUNEK 3.17. Typowa realizacja fizyczna architektury magistralowej

Magistrala rozciąga się przez cały system, a wszystkie jego zespoły są podłączone do pewnej liczby lub do wszystkich linii magistrali. Bardzo powszechne rozwiązanie fizyczne jest przedstawione na rys. 3.17. W przykładzie tym magistrala składa się z dwóch pionowych zespołów połączeń. Wzdłuż tych zespołów połączeń, w regularnych odstępach są rozmieszczone punkty mocowania w postaci poziomych gniazd podtrzymujących płytki drukowane. Każdy z głównych zespołów systemu zajmuje jedną lub więcej takich płytek i jest połączony z magistralą poprzez gniazda. Cały ten zestaw jest zamknięty w obudowie.

Przedstawione rozwiązanie jest najwygodniejsze. Można zakupić mały system komputerowy a następnie go rozszerzyć (zwiększyć pamięć, rozbudować wejście-wyjście) przez dołączenie dodatkowych płytek. Jeśli element na płycie ulega uszkodzeniu, można łatwo usunąć i wymienić płytkę.

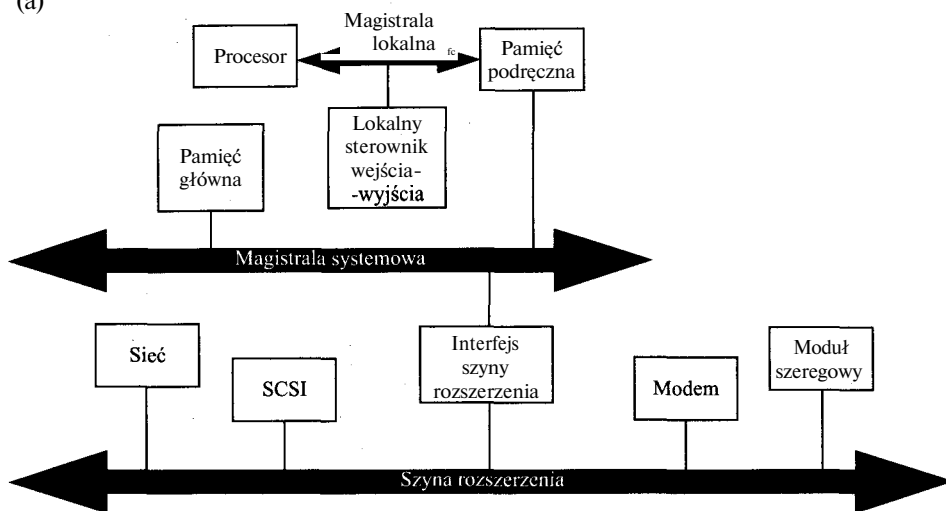
Hierarchiczne struktury wielomagistralowe

Jeśli do magistrali jest dołączona znaczna liczba przyrządów, to cierpi na tym wydajność. Sąto dwie główne przyczyny:

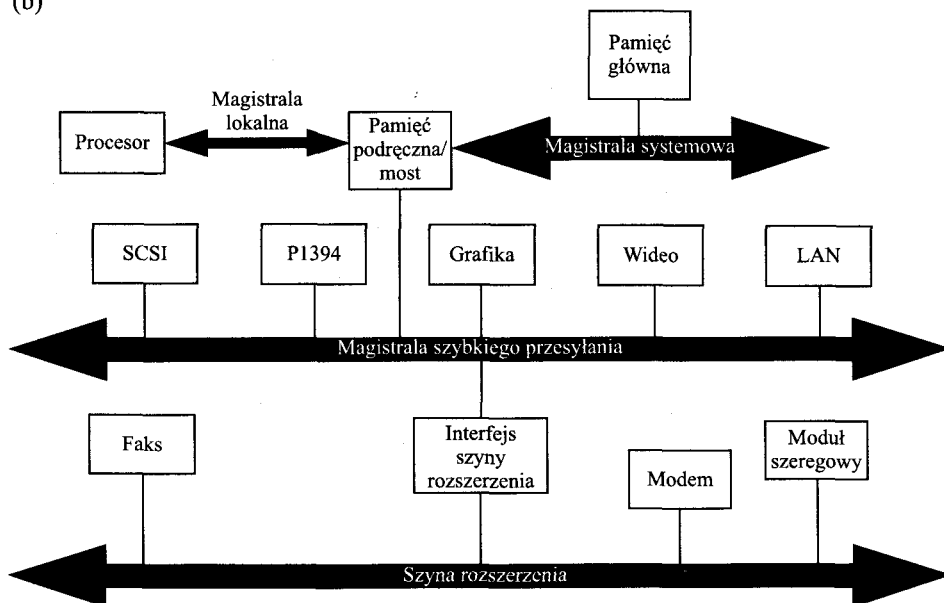
1. Na ogół, im więcej przyrządów dołączono do magistrali, tym większe jest opóźnienie propagacji. Opóźnienie to określa czas potrzebny do tego, aby skoordynować wykorzystanie magistrali. Jeśli sterowanie magistralą przenosi się często od zespołu do zespołu, to opóźnienia propagacji mogą zauważalnie obniżyć wydajność.
2. Magistrala może się stać wąskim gardłem, jeśli zapotrzebowanie na przesłanie zgromadzonych danych zbliża się do pojemności magistrali. Problemowi temu można do pewnego stopnia przeciwdziałać, zwiększając szybkość przenoszenia danych przez magistralę, a także stosując szersze magistrale (np. 64-bitowe zamiast 32-bitowych). Ponieważ jednak szybkości generowania danych przez dołączone urządzenia (np. sterowniki grafik i wideo, interfejsy sieciowe) wzrastają szybko, przesyłanie pojedynczą magistralą jest w tym wyjściu skazane na porażkę.

W tej sytuacji w większości systemów komputerowych wykorzystuje się struktury wielomagistralowe o określonej hierarchii. Typowa struktura tradycyjna jest pokazana na rys. 3.18a. Występuje tu lokalna magistrala łącząca procesor i pamięć podręczną. Może ona wspomagać jedno lub więcej urządzeń lokalnych. Sterownik pamięci podręcznej łączy tę pamięć nie tylko do magistrali lokalnej, ale również do magistrali systemowej, do której są dołączone wszystkie moduły pamięci głównej. Jak dowiemy się w rozdz. 4, wykorzystanie struktury pamięci podręcznej uwalnia procesor od potrzeby częstego dostępu do pamięci głównej. Dzięki temu pamięć główna może być przesunięta z magistrali lokalnej do systemowej. W ten sposób wejście-wyjście komunikuje się z pamięcią główną przez magistralę systemową, nie kolidując z działalnością procesora.

(a)



(b)



RYSUNEK 3.18. Przykładowe konfiguracje magistralowe: (a) tradycyjna architektura magistralowa; (b) architektura o dużej wydajności

Możliwe jest podłączenie sterowników wejścia-wyjścia bezpośrednio do magistrali systemowej. Bardziej efektywnym rozwiązaniem jest jednak wykorzystanie do tego celu jednej lub wielu szyn rozszerzenia (ang. *expansion buses*). Interfejs szyny rozszerzenia buforuje dane przesyłane między magistralą

systemową a sterownikami wejścia-wyjścia dołączonymi do szyny rozszerzenia. Rozwiązanie to umożliwia systemowi wykorzystywanie wielu urządzeń wejścia-wyjścia i jednocześnie izolowanie ruchu między pamięcią, a procesorem od ruchu związanego z wejściem-wyjściem.

Na rysunku 3.18a są pokazane typowe przykłady urządzeń wejścia-wyjścia, które mogą być podłączone do szyny rozszerzenia. Połączenia sieciowe obejmują sieci lokalne (ang. *local area networks* - LANs), takie jak Ethernet o przepustowości 10 Mbit/s, oraz sieci o dużym zasięgu, takie jak sieć komutacji pakietów (ang. *packet-switching network*). Interfejs SCSI (ang. *small computer system interface*) sam jest rodzajem magistrali, która jest wykorzystywana do współpracy z lokalnymi napędami dysków i innymi urządzeniami peryferyjnymi. Port szeregowy może być wykorzystywany do współpracy z drukarką lub skanerem.

Ta tradycyjna architektura magistralowa jest rozsądnie efektywna, jednak zaczyna się załamywać w miarę, jak rośnie wydajność urządzeń wejścia-wyjścia. W odpowiedzi na to zapotrzebowanie powszechnym rozwiązaniem przyjmowanym w przemyśle jest budowanie szybkich magistrali ściśle zintegrowanych z resztą systemu, wymagających tylko mostu między magistralą procesora a magistralą szybką. Rozwiązanie to jest czasem określane jako **architektura międzypiętrowa** (ang. *mezzanine architecture*).

Na rysunku 3.18b jest pokazana typowa realizacja powyższego rozwiązania. Znowu występuje tu magistrala lokalna, *łącząca* procesor ze sterownikiem pamięci podręcznej, który z kolei jest podłączony do magistrali systemowej współpracującej z pamięcią główną. Sterownik pamięci podręcznej jest zintegrowany z mostem (urządzeniem buforującym) łączącym z magistralą szybką. Magistrala ta obsługuje połączenia z szybkimi sieciami LAN, takimi jak sieci FDDI* o przepustowości 100 Mbit/s, sterowniki urządzeń graficznych i wideo oraz sterowniki interfejsów z lokalnymi magistralami peryferyjnymi, w tym SCSI i P1394. Ten ostatni jest urządzeniem z szybką magistralą, zaprojektowanym specjalnie do obsługi urządzeń wejścia-wyjścia o dużej przepustowości. Wolniejsze urządzenia nadal są obsługiwane przez szynę rozszerzenia z interfejsem buforującym ruch między szyną rozszerzenia a szybką magistralą.

Zaletą tego rozwiązania jest to, że szybka magistrala ściślej sprzęga procesor z urządzeniami wejścia-wyjścia o wysokich wymaganiach, a jednocześnie jest niezależna od procesora. Dzięki temu mogą być tolerowane różnice szybkości procesora i szybkiej magistrali, a także różne definicje linii sygnałowych. Zmiany architektury procesora nie wpływają na szybką magistralę i vice versa.

* Łączy dane w sieciach opartych na światłowodach o dużych przepustowościach; ang. *Fiber Distributed Data Interface* (przyp. tłum.).

Elementy projektowania magistrali

Chociaż występuje wiele różnych rozwiązań magistrali, istnieje kilka podstawowych parametrów i elementów projektowych, które służą do klasyfikowania i różnicowania magistrali. W tabeli 3.2 są wymienione główne z tych elementów.

TABELA 3.2. Elementy projektowania magistrali

Rodzaj	Szerokość magistrali
specjalistyczna	adres
multipleksowana	dane
Metoda arbitrażu	Rodzaj transferu danych
centralny	odczyt
rozproszony	zapis
Koordynacja czasowa	odczyt-modyfikacja-zapis
synchroniczna	odczyt po zapisie
asynchroniczna	blokowy

Rodzaje magistrali

Linie magistralowe mogą być podzielone na dwa rodzaje: specjalistyczne (ang. *dedicated*) i multipleksowane. Linia specjalistyczna jest trwale przypisana albo jednej funkcji, albo fizycznie określonym zespołom komputera.

Przykładem specjalizacji funkcjonalnej jest zastosowanie oddzielnych, specjalistycznych linii adresów i danych, co jest powszechne w wielu magistralach. Nie jest to jednak jedyne rozwiązanie. Na przykład, adresy i dane mogą być transmitowane przez ten sam zespół linii przy wykorzystaniu linii sterowania określającej ważność adresu (*Address Valid*). W tym przypadku każdy moduł ma do dyspozycji określony odcinek czasu na skopiowanie adresu i stwierdzenie, czy jest modułem adresowanym. Adres jest następnie usuwany z magistrali, a te same połączenia magistralowe są wykorzystywane do przenoszenia danych odczytywanych lub zapisywanych. Ta właśnie metoda używania tych samych linii do wielu celów jest znana jako **multipleksowanie czasowe** (ang. *time multiplexing*).

Zaletą multipleksowania czasowego jest stosowanie mniejszej liczby linii, co pozwala oszczędzić miejsce i (zwykle) koszt. Wadą jest to, że wewnątrz każdego modułu są potrzebne bardziej złożone układy. Potencjalnie może też nastąpić zmniejszenie wydajności, ponieważ niektóre zdarzenia wymagające tych samych linii nie mogą zachodzić równolegle.

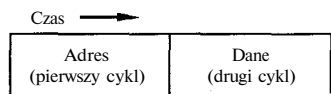
Specjalizacja fizyczna (ang. *physical dedication*) odnosi się do używania wielu magistrali, z których każda łączy tylko określoną grupę modułów. Typowym przykładem jest zastosowanie magistrali wejścia-wyjścia do łączenia wszystkich modułów wejścia-wyjścia. Magistrala ta jest następnie łączona

Szerokość magistrali

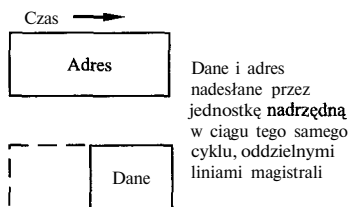
Wspomnieliśmy już o pojęciu szerokości magistrali. Szerokość szyny danych ma wpływ na wydajność systemu: im szersza jest szyna danych, tym większa jest liczba jednocześnie przesyłanych bitów. Szerokość szyny adresowej ma natomiast wpływ na pojemność systemu: im szersza jest szyna adresowa, tym większa jest ilość miejsc możliwych do określenia w pamięci.

Rodzaje transferu danych

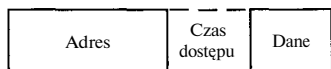
Magistrala służy do przesyłania różnych rodzajów danych, co widać na rys. 3.20. Wszystkie magistrale obsługują zarówno zapis (transfer od modułu nadrzędnego do podrzędnego), jak i odczyt (transfer w przeciwnym kierunku). W przypadku multipleksowanych szyn adresów i danych szyna jest najpierw używana do specyfikowania adresu, a następnie do przesyłania danych. W operacji odczytu występuje typowe oczekiwanie, podczas gdy dane są pobierane



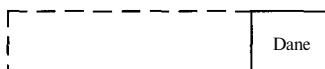
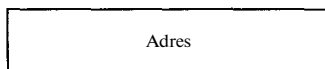
Operacja zapisu (multipleksowana)



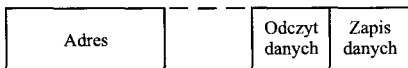
Operacja zapisu (nie multipleksowana)



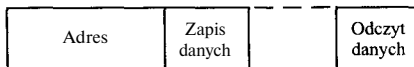
Operacja odczytu (multipleksowana)



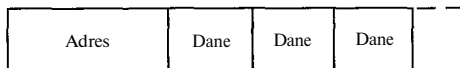
Operacja odczytu (nie multipleksowana)



Operacja odczyt-modyfikacja-zapis



Operacja odczyt po zapisie



Blokowe przesyłanie danych

RYSUNEK 3.20. Rodzaje magistralowego transferu danych [GOOR89]

z modułu podrzędnego i wprowadzane do magistrali. Zarówno przy zapisie, jak i odczycie może również występować opóźnienie, jeśli zaistnieje potrzeba arbitrażu, aby przejąć sterowanie linią w celu wykonania pozostałej części operacji (np. w celu przejścia magistrali przy żądaniu odczytu lub zapisu, a następnie ponownego przejścia magistrali w celu wykonania odczytu lub zapisu).

W przypadku specjalistycznych szyn adresów i danych adres jest lokowany na szynie adresowej i pozostaje tam w czasie, gdy dane są doprowadzane do szyny danych. Przy operacji zapisu moduł nadrzędny umieszcza dane na szynie danych natychmiast po ustabilizowaniu się adresu, gdy moduł podrzędny ma możliwość rozpoznania adresu. Przy operacji odczytu moduł podrzędny umieszcza dane na szynie danych tuż po rozpoznaniu adresu i pobraniu danych.

Na niektórych magistralach dopuszczalne są operacje kombinowane. Operacja odczyt-modyfikacja-zapis jest po prostu odczytem, po którym natychmiast następuje zapis pod tym samym adresem. Adres jest rozgłaszany tylko raz, na początku operacji. W typowym przypadku cała operacja jest niepodzielna w celu zapobieżenia dostępowi do danych przez inne potencjalne moduły nadrzędne. Zasadniczym celem tego rozwiązania jest ochrona zasobów pamięci w systemach wieloprogramowych (zob. rozdz. 7).

Odczyt-po-zapisie jest niepodzielną operacją składającą się z zapisu, po którym natychmiast następuje odczyt z tego samego adresu. Operacja odczytu może być wykonywana w celu kontroli.

Niektóre systemy magistralowe umożliwiają także blokowe przesyłanie danych. W tym przypadku po jednym cyklu adresu następuje n cykli danych. Pierwsza grupa danych jest przesyłana pod sprecyzowany adres (lub pobierana pod tym adresem), pozostałe dane są przesyłane pod następne adresy (lub pobierane pod tymi adresami).

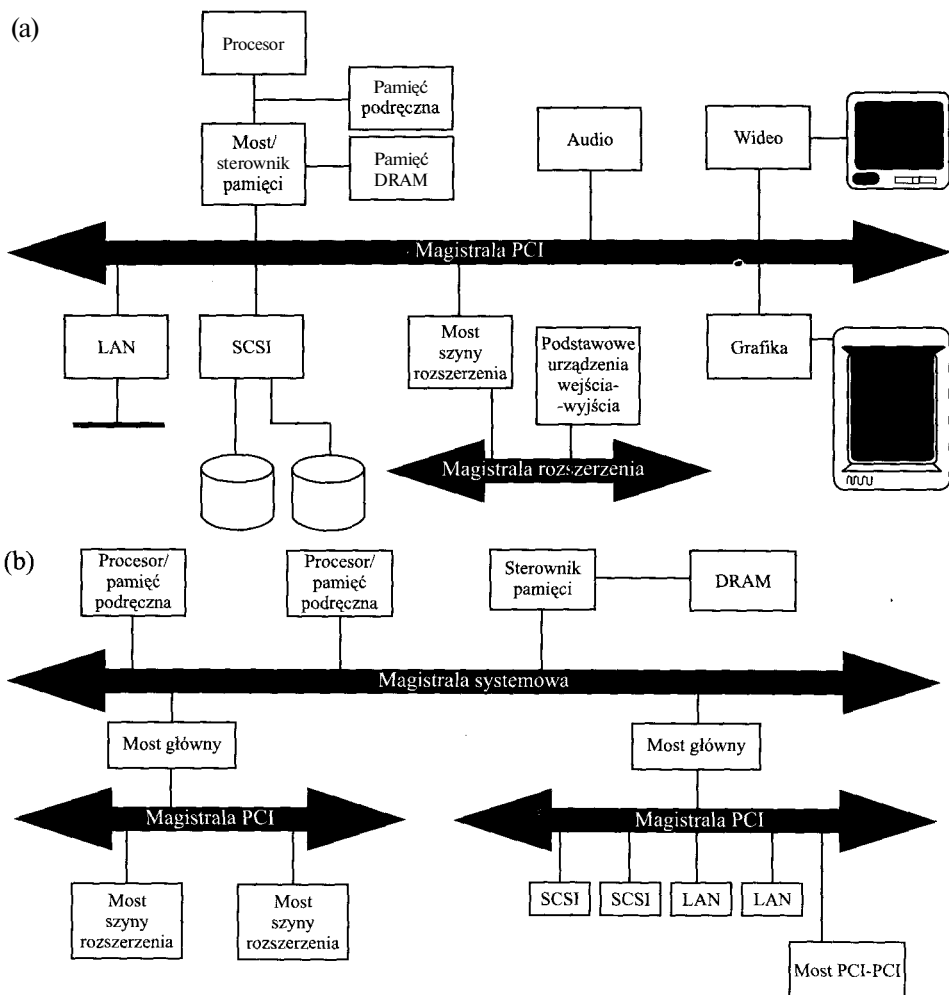
3.5

MAGISTRALA PCI

System połączeń urządzeń peryferyjnych PCI (skrót od ang. *Peripheral Component Interconnect*) jest współczesną, szerokopasmową magistralą niezależną od procesora, która może funkcjonować jako magistrala „międzypiętrowa” (*mezzanine*) lub peryferyjna. W porównaniu z innymi, powszechnie spotykanymi magistralami PCI umożliwia uzyskanie większej wydajności systemu, jeśli są wykorzystywane szybkie podsystemy wejścia-wyjścia (np. urządzenia graficzne, sterowniki interfejsów sieciowych, sterowniki dysków i inne). Aktualna norma pozwala na użycie do 64 linii danych przy 33 MHz, co daje szybkość przesyłania danych 264 MB/s lub 2,112 Gbit/s. Jednak nie tylko duża szybkość stanowi o atrakcyjności PCI. Magistrala PCI została zaprojektowana jako ekonomiczne rozwiązanie spełniające wymagania wejścia-wyjścia w nowoczesnych systemach; wymaga niewielu mikroukładów i wspomaga działanie innych magistrali, które są połączone z magistralą PCI.

Intel zaczął prace nad magistralą PCI w roku 1990, mając na uwadze systemy wykorzystujące mikroprocesor Pentium. Wkrótce Intel opatentował niezbędne rozwiązania i promował utworzenie przemysłowego stowarzyszenia pod nazwą PCI SIG, mającego na celu dalszy rozwój i zachowanie kompatybilności z magistralą PCI. W rezultacie magistrala PCI została szeroko zaakceptowana, a jej zastosowanie w komputerach osobistych, stacjonarnych i w systemach serwerów wzrasta. Aktualna wersja, PCI 2,0, została wprowadzona w roku 1993. Ponieważ specyfikacja jest dostępna publicznie i zyskała poparcie znacznej części przemysłu mikroprocesorów i urządzeń peryferyjnych, wyroby z magistralą PCI sprzedawane przez różnych dostawców są kompatybilne.

Magistralę PCI zaprojektowano dla szerokiego zakresu konfiguracji wykorzystujących mikroprocesory, łącznie z systemami jedno- i wieloprocessorowymi.



RYSUNEK 3.21. Przykładowe konfiguracje magistrali PCI: (a) typowy system stołowy; (b) typowy system serwera

Dlatego też PCI realizuje zespół funkcji o ogólnym przeznaczeniu. Wykorzystuje koordynację synchroniczną i arbitraż scentralizowany.

Na rysunku 3.21a jest pokazane typowe wykorzystanie magistrali PCI w systemie jednoprocessorowym. Zintegrowany układ sterownika DRAM i mostu do magistrali PCI zapewnia ściśle sprzężenie z procesorem i możliwość dostarczania danych z dużą szybkością. Most działa jako bufor danych, dzięki czemu szybkość magistrali PCI może się różnić od szybkości procesora. W systemie wieloprocessorowym (rys. 3.21b) jedną lub więcej konfiguracji magistrali PCI można połączyć za pomocą mostów z magistralą systemową procesorów. Magistrala systemowa obsługuje jedynie zespoły procesor/pamięć podręczna, pamięć główną i mosty PCI. Również i tutaj zastosowanie mostów uniezależnia magistralę PCI od szybkości procesora, umożliwiając jednocześnie szybkie otrzymywanie i dostarczanie danych.

Struktura magistrali

Magistrala PCI może być konfigurowana jako magistrala 32- lub 64-bitowa. W tabeli 3.3 są przedstawione obowiązujące linie sygnałowe magistrali PCI. Jest ich 50. Można je podzielić na następujące grupy funkcjonalne:

- ❑ **Wyprowadzenia systemowe.** Należą do nich wyprowadzenia zegara i przywracania (*reset*).
- a **Wyprowadzenia adresu i danych.** Należą do nich 32 linie multipleksowane adresów i danych. Pozostałe linie w tej grupie są wykorzystywane do interpretowania i określania ważności linii sygnałowych przenoszących adresy i dane.
- a **Wyprowadzenia sterowania interfejsu.** Służą do koordynowania transakcji i współdziałania między jednostkami inicjującymi a docelowymi.
- Q **Wyprowadzenia arbitrażowe.** W przeciwieństwie do pozostałych linii sygnałowych magistrali PCI nie są one liniami wspólnymi. Każda jednostka nadrzędna PCI ma własną parę linii arbitrażowych, które łączą ją bezpośrednio z arbitrem magistrali PCI.
- Q **Wyprowadzenia informujące o błędach.** Wykorzystywane do informowania o błędach parzystości i innych.

Ponadto specyfikacja PCI określa 50 opcjonalnych linii sygnałowych (tabela 3.4), podzielonych na następujące grupy:

- a **Wyprowadzenia przerwania.** Są one przewidziane dla urządzeń PCI, które muszą generować zapotrzebowanie na obsługę. Tak jak w przypadku wyprowadzeń arbitrażowych nie są one wspólne. Każde urządzenie PCI dysponuje własną linią lub liniami przerwania łączącymi ze sterownikiem przerwania.
- ❑ **Wyprowadzenia obsługi pamięci podręcznej.** Są one wykorzystywane do obsługi pamięci podręcznych zawartych w procesorach i innych urządze-

TABELA 3.3. Obowiązujące linie sygnałowe standardu PCI

Oznaczenie	Rodzaj	Opis
<i>Wyprowadzenia systemowe</i>		
CLK	we	Zapewnia koordynację czasową wszystkich transakcji i jest próbkowana przez wszystkie wejścia podczas zbocza rosnącego. Częstotliwość zegara do 33 MHz.
RST#	we	Wymusza inicjowanie wszystkich zgodnych ze standardem PCI rejestrów, sekwenserów i sygnałów.
<i>Wyprowadzenia adresów i danych</i>		
AD[31+0]	t/s	Multipleksowane linie do przenoszenia adresów i danych.
C/BE[3+0]#	t/s	Multipleksowane sygnały rozkazów magistralowych oraz zezwolenia bajtów. Podczas fazy danych linie te wskazują, które z czterech szyn bajtów niosą znaczące dane.
PAR	t/s	Dostarcza sygnały parzystości linii AD i C/BE z opóźnieniem jednego cyklu zegara. Jednostka nadrzędna aktywuje linie PAR w fazach adresu i zapisu danych; jednostka docelowa aktywuje linie PAR w fazach odczytu danych.
<i>Wyprowadzenia sterowania interfejsem</i>		
FRAME#	s/t/s	Aktywowana przez bieżącą jednostkę nadrzędną w celu wskazania początku i czasu trwania transakcji. Jest potwierdzana na początku i negowana, gdy inicjator jest gotowy do rozpoczęcia końcowej fazy danych.
IRDY#	s/t/s	Gotowość inicjatora. Aktywowana przez bieżącą jednostkę nadrzędną magistrali (inicjatora transakcji). Podczas odczytu wskazuje, że jednostka nadrzędna jest gotowa do przyjęcia danych; podczas zapisu wskazuje, że na AD obecne są ważne dane.
TRDY#	s/t/s	Gotowość jednostki docelowej. Aktywowana przez jednostkę docelową (wybrany przyrząd). Podczas odczytu wskazuje, że na AD są obecne ważne dane; podczas zapisu wskazuje, że jednostka docelowa jest gotowa do przyjęcia danych.
STOP#	s/t/s	Wskazuje, że bieżąca jednostka docelowa domaga się od inicjatora <i>zatrzymania</i> bieżącej transakcji.
LOCK#	s/t/s	Wskazuje operację elementarną, która może wymagać wielu transakcji.
IDSEL	we	Wybór przyrządu inicjowania. Używana do wyboru mikroukładu podczas konfigurowania transakcji odczytu i zapisu.
DEVSEL#	we	Wybór przyrządu. Potwierdzana przez jednostkę docelową, gdy rozpoznała ona swój adres. Wskazuje bieżącemu inicjatorowi, czy został wybrany jakikolwiek przyrząd.
<i>Wyprowadzenia arbitrażowe</i>		
REQ#	t/s	Wskazuje arbitrowi, że ten przyrząd wymaga użycia magistrali. Jest to linia doprowadzona do konkretnego przyrządu.
GNT#	t/s	Wskazuje przyrządowi, że arbiter przekazał mu dostęp do magistrali. Jest to linia doprowadzona do konkretnego przyrządu.
<i>Wyprowadzenia informowania o błędach</i>		
PERR#	s/t/s	Błąd parzystości. Wskazuje, że został wykryty błąd parzystości danych przez jednostkę docelową podczas fazy zapisu danych lub przez inicjatora podczas fazy odczytu danych.
SERR#	o/d	Błąd systemowy. Może być wzbudzona przez jakikolwiek przyrząd w celu poinformowania o błędach parzystości adresu oraz o błędach krytycznych innych niż błędy parzystości.

TABELA 3.4. Opcjonalne linie sygnałowe standardu PCI

Oznaczenie	Rodzaj	Opis
<i>Doprowadzenia przerwania</i>		
INTA#	o/d	Używana do zgłaszania zapotrzebowania na przerwanie.
INTB#	o/d	Używana do zgłaszania zapotrzebowania na przerwanie; ma znaczenie tylko w odniesieniu do przyrządu wielofunkcyjnego.
INTC#	o/d	Używana do zgłaszania zapotrzebowania na przerwanie; ma znaczenie tylko w odniesieniu do przyrządu wielofunkcyjnego.
INTD#	o/d	Używana do zgłaszania zapotrzebowania na przerwanie; ma znaczenie tylko w odniesieniu do przyrządu wielofunkcyjnego.
<i>Doprowadzenia wspierające pamięć podręczną</i>		
SBO#	we-wy	Wycofanie się z podglądania. Wskazuje na trafienie zmodyfikowanego wiersza.
SDONE	we-wy	Zakończenie podglądania. Wskazuje stan podglądania. Potwierdzana po zakończeniu podglądania.
<i>Wyprowadzenia rozszerzenia magistrali do 64 bitów</i>		
AD[63÷32]	t/s	Linie multipleksowane używane do adresów i danych w celu rozszerzenia magistrali do 64 bitów.
C/BE[7÷4]#	t/s	Multipleksowane sygnały rozkazów magistralowych i sygnałów zezwolenia bajtów. Podczas fazy adresu linie te dostarczają dodatkowych rozkazów magistralowych. Podczas fazy danych linie te wskazują, która z czterech rozszerzonych szyn bajtu przenosi znaczące dane.
REQ64#	s/t/s	Używana do zgłaszania zapotrzebowania na transfer w pakietach po 64 bity.
ACK64#	s/t/s	Wskazuje, że jednostka docelowa chce dokonać transferu 64-bitowego.
PAR64	t/s	Dostarcza sygnałów parzystości rozszerzonych linii AD i C/BE z opóźnieniem jednego cyklu zegara.
<i>Wyprowadzenia testowania granic JTAG</i>		
TCK	we	Zegar testowy. Używana do synchronizowania wprowadzania i wyprowadzania informacji o stanie i danych testowych do (i z) przyrządu podczas skanowania granic.
TDI	we	Wejście testowe. Używana do szeregowego wprowadzania danych testowych i rozkazów do przyrządu.
TDO	wy	Wyjście testowe. Używana do szeregowego wyprowadzania danych testowych i rozkazów z przyrządu.
TMS	we	Wybór trybu testowania. Używana do kontrolowania stanu sterownika portu testowego.
TRST#	we	Inicjowanie testu. Używana do inicjowania sterownika portu testowego.

we - tylko sygnał wejściowy,

wy - tylko sygnał wyjściowy,

t/s - dwukierunkowy, trójstanowy sygnał we-wy,

s/t/s - podtrzymywany sygnał trójstanowy, aktywowany w danej chwili tylko przez jeden przyrząd,

o/d - otwarty dren: pozwala wielu przyrządom na wspólne używanie w formie układowego LUB,

- aktywny stan sygnału ma miejsce przy niskim napięciu.

niach PCI. Umożliwiają stosowanie protokołu podglądania (*snoopy*) pamięci podręcznej. Protokoły te są omówione w rozdz. 16.

- **Wyprowadzenia rozszerzenia magistrali 64-bitowej.** Należą do nich 32 linie multipleksowane adresów i danych. W połączeniu z obowiązującymi liniami adresów i danych tworzą 64-bitową magistralę adresów i danych.

Inne linie z tej grupy są używane do interpretowania i określania ważności linii sygnałowych przenoszących dane i adresy. Wreszcie występują tu dwie linie umożliwiające dwóm urządzeniom PCI uzgodnienie użycia szerokości 64-bitowej.

- a **Wyprowadzenia testowania granic JTAG.** Linie te umożliwiają realizację procedur testowania określonych w normie IEEE 149.1.

Rozkazy PCI

Działanie magistrali odbywa się w formie transakcji między inicjatorem (modułem nadrzędnym) a celem (modułem podrzędnym). Gdy inicjator domaga się sterowania magistralą, określa typ transakcji, która ma być przeprowadzona. Podczas fazy adresu transakcji do sygnalizowania typu transakcji są używane linie C/BE. Rozkazy są następujące:

- a potwierdzenie przzerwania,
- a cykl specjalny,
- a odczyt wejścia-wyjścia,
- a zapis wejścia-wyjścia,
- ☐ odczyt pamięci,
- ☐ linia odczytu pamięci,
- a zwielokrotniony odczyt pamięci,
- ☐ zapis w pamięci,
- a zapis w pamięci i unieważnienie,
- a odczyt konfiguracji,
- a zapis konfiguracji,
- ☐ cykl podwójnego adresu.

„Potwierdzenie przzerwania” jest rozkazem odczytu przeznaczonym dla urządzenia, które działa jako sterownik przerwań na magistrali PCI. Linie adresowe nie są używane podczas fazy adresu, a linie zezwolenia bajtu wskazują rozmiar identyfikatora przzerwania, który musi być zwrócony.

„Cykl specjalny” jest rozkazem używanym przez inicjatora w celu rozgłoszenia wiadomości przeznaczonych dla jednego lub więcej celów.

Rozkazy „odczyt wejścia-wyjścia” i „zapis wejścia-wyjścia” są używane do przenoszenia danych między inicjatorem a sterownikiem wejścia-wyjścia. Każde urządzenie wejścia-wyjścia ma własną przestrzeń adresową. Linie adresowe służą do wskazywania określonego urządzenia oraz do sprecyzowania danych, które mają być wymienione z tym urządzeniem. Koncepcją adresów wejścia-wyjścia zajmiemy się w rozdz. 6.

Rozkazy odczytu i zapisu w pamięci są stosowane do precyzowania przesyłania porcji danych (*burst of data*) trwającego przez jeden lub więcej cykli zegarowych. Interpretacja tych rozkazów zależy od tego, czy sterownik pamięci na magistrali PCI realizuje protokół PCI przeznaczony do przesyłania danych mię-

dzy pamięcią główną a pamięcią podręczną. Jeśli tak, to przesyłanie danych do (i z) pamięci odbywa się typowo w postaci wierszy lub bloków². Trzy rozkazy odczytu pamięci mają zastosowania wymienione w tabeli 3.5. Rozkaz „zapis w pamięci” jest stosowany do przenoszenia danych do pamięci, w czasie jednego lub wielu cykli danych. Rozkaz „zapis w pamięci i unieważnienie” również służy do przenoszenia danych do pamięci, w czasie jednego lub wielu cykli. Ponadto gwarantuje on, że przynajmniej jedna linia pamięci podręcznej jest zapisana. Rozkaz ten obsługuje funkcję pamięci podręcznej polegającą na opóźnionym zapisie wiersza w pamięci.

TABELA 3.5. Interpretacja rozkazów odczytu standardu PCI

Rodzaj rozkazu odczytu	Pamięć współpracująca z pamięcią podręczną	Pamięć główna bez pamięci podręcznej
Odczyt z pamięci	pakietowanie co najwyżej 1/2 wiersza pamięci podręcznej	pakietowanie co najwyżej 2 cykli transferu
Odczyt wiersza z pamięci	pakietowanie od 1/2 do 3 wierszy pamięci podręcznej	pakietowanie od 3 do 12 cykli transferu danych
Wielokrotny odczyt z pamięci	pakietowanie ponad 3 wierszy pamięci podręcznej	pakietowanie ponad 12 cykli transferu danych

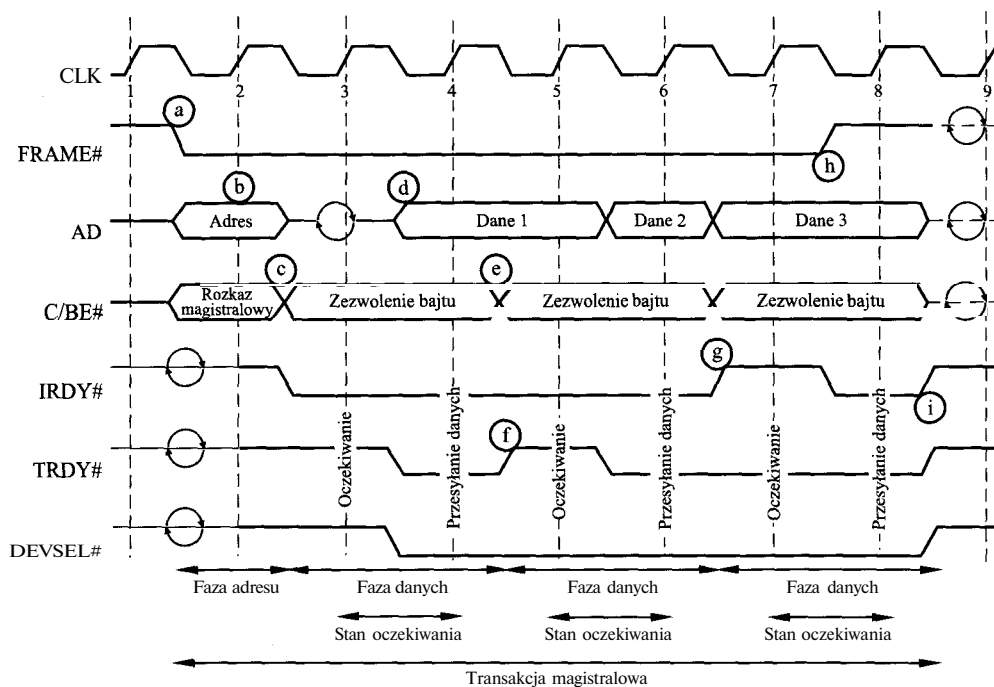
Dwa rozkazy konfiguracyjne umożliwiają inicjatorowi odczytanie i zaktualizowanie parametrów konfiguracji w urządzeniu dołączonym do magistrali PCI. Każde urządzenie PCI może zawierać do 256 rejestrów wewnętrznych, które są wykorzystywane do konfigurowania tego urządzenia podczas inicjowania systemu.

Rozkaz „cykl podwójnego adresu” jest wykorzystywany przez inicjatora do wskazania, że stosuje on adresowanie 64-bitowe.

Przesyłanie danych

Każde przesyłanie danych magistralą PCI jest pojedynczą transakcją składającą się z jednej fazy adresowej oraz jednej lub wielu faz danych. Przedstawimy obecnie typową operację odczytu; operacja zapisu przebiega podobnie. Na rysunku 3.22 jest pokazana koordynacja czasowa sygnałów podczas transakcji odczytu. Wszystkie zdarzenia są synchronizowane przez opadające części impulsów zegarowych, które występują w środku każdego cyklu zegara. Urządzenia magistrali badają linie magistrali w czasie narastania impulsów zegarowych, na początku cyklu magistrali.

² Podstawowe zasady dotyczące pamięci podręcznych są opisane w podrozdz. 4.3; protokoły pamięci podręcznych w systemach magistralowych są opisane w podrozdz. 16.2.



RYSUNEK 3.22. Przebieg operacji odczytu na magistrali PCI

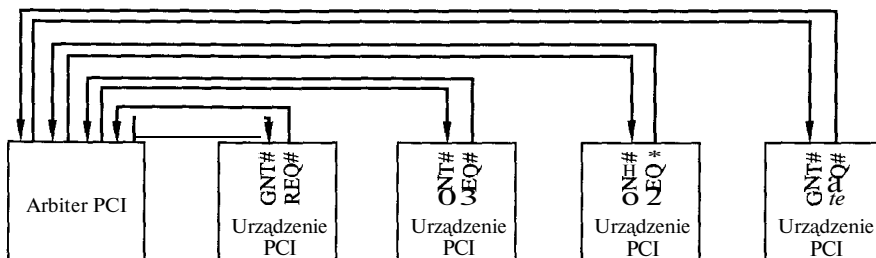
Oto znaczące zdarzenia pokazane na wykresie przebiegów czasowych:

- Moduł nadrzędny (*bus master*), gdy tylko uzyska sterowanie magistralą, może rozpocząć transakcję przez potwierdzenie FRAME. Linia ta pozostaje potwierdzona aż do chwili, w której inicjator jest gotowy do zakończenia ostatniej fazy danych. Inicjator umieszcza także adres startowy na szynie adresowej i odczytuje rozkaz na liniach C/BE.
- Na początku drugiego impulsu zegarowego urządzenie będące celem rozpoznaje swój adres na liniach AD.
- Inicjator wstrzymuje sterowanie magistrali AD. Na wszystkich liniach sygnałowych, które mogą być sterowane przez więcej niż jedno urządzenie, jest wymagany cykl obiegowy (*turnaround cycle*), pokazany na rysunku za pomocą kolistych strzałek. Dzięki usunięciu sygnału adresowego magistrala będzie przygotowana do wykorzystania przez urządzenie będące celem. Inicjator zmienia informację na liniach C/BE, aby określić, które linie AD mają być wykorzystane przy przesyłaniu aktualnie adresowanych danych (od 1 do 4 bajtów). Inicjator potwierdza także IRDY w celu pokazania, że jest przygotowany do przyjęcia pierwszych danych.
- Wybrany cel potwierdza DEVSEL, aby pokazać, że rozpoznał swój adres i udzieli odpowiedzi. Umieszcza wymagane dane na liniach AD i potwierdza TRDY, aby pokazać, że na magistrali są obecne ważne dane.

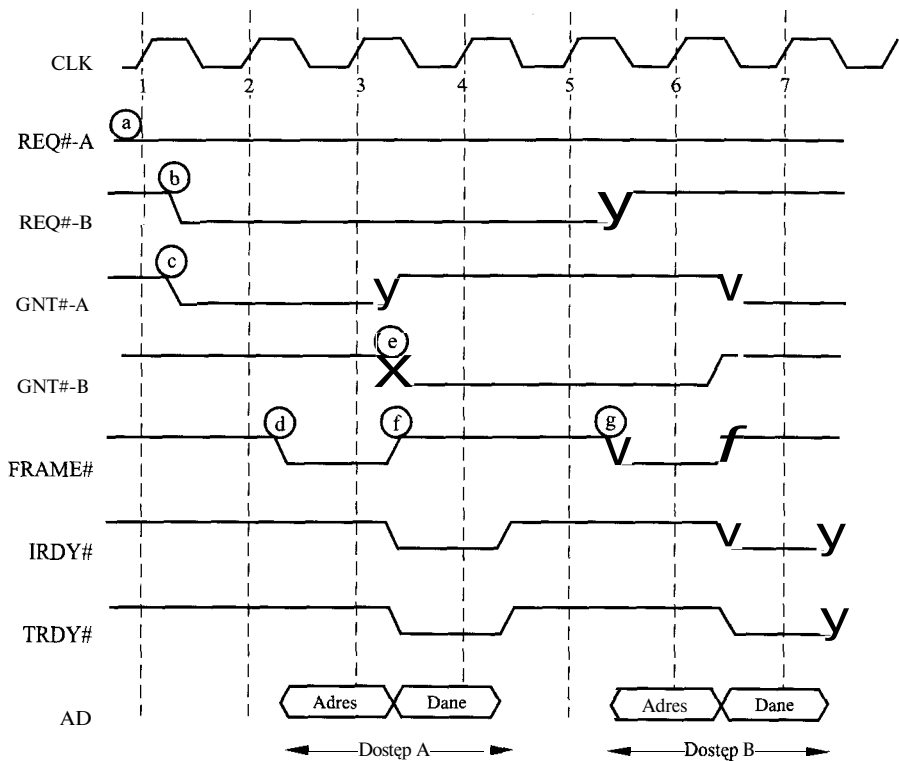
- (e) Inicjator odczytuje dane na początku czwartego impulsu zegarowego i zmienia stan linii zezwolenia bajtu (*byte enable lines*) w celu przygotowania do następnego odczytu.
- (f) W omawianym przykładzie cel potrzebuje pewnego czasu, aby przygotować się do transmisji drugiego bloku danych. Usuwa więc potwierdzenie TRDY, aby zasygnalizować inicjatorowi, że w ciągu nadchodzącego cyklu nie będzie nowych danych. Zgodnie z tym inicjator nie odczytuje linii danych na początku piątego cyklu zegara i nie musi zmieniać stanu linii zezwolenia bajtu podczas tego cyklu. Blok danych jest odczytywany na początku szóstego cyklu.
- (g) Podczas szóstego cyklu zegara cel umieszcza trzecią porcję danych na magistrali. Jednak w tym przykładzie inicjator nie jest jeszcze gotowy do odczytywania porcji danych (może być w stanie czasowo zapełnionych buforów). Odwołuje więc potwierdzenie IRDY, co powoduje, że cel będzie zachowywał trzecią porcję danych na magistrali w ciągu następnego cyklu zegara.
- (h) Inicjator wie, że trzecie przesłanie danych będzie ostatnim, odwołuje więc FRAME, aby w ten sposób zasygnalizować celowi, że jest to właśnie ostatnie przesłanie. Potwierdza także IRDY w celu pokazania, że jest gotów do zakończenia tego przesłania.
- (i) Inicjator odwołuje IRDY, powodując powrót magistrali do stanu jałowego, natomiast cel odwołuje TRDY i DEVSEL.

Arbitraż

Magistrala PCI wykorzystuje scentralizowany, synchroniczny rodzaj arbitrażu, w którym każdy moduł nadrzędny ma przypisane unikatowe sygnały zapotrzebowania (REQ) i udostępniania (*grant* - GNT). Odpowiednie linie sygnałowe są podłączone do centralnego arbitra (rys. 3.23), a dostęp do magistrali jest uzyskiwany przez zgodne zgłoszenie zapotrzebowania i udostępniania. Specyfikacja PCI nie dyktuje szczególnego algorytmu arbitrażu. Arbiter może wykorzystywać podejście „pierwszy zgłoszony - pierwszy obsłużony”, rozwiązanie cykliczne (*round-robin*) lub jakiś rodzaj układu priorytetów. Moduł nadrzędny PCI musi korzystać z arbitrażu przy każdej transakcji, którą chce przeprowadzić, ponieważ



RYСУNEK 3.23. Arbiter magistrali PCI



RYСУNEK 3.24. Arbitraż magistrali PCI między dwiema jednostkami nadrzędnymi

pojedyncza transakcja składa się z fazy adresu, po której następuje jedna lub wiele faz danych.

Na rysunku 3.24 jest pokazany przykład, w którym urządzenia A i B odwołują się do arbitrażu. Realizowana jest następująca sekwencja zdarzeń:

- W pewnym momencie, przed rozpoczęciem pierwszego cyklu zegara, A potwierdza swój sygnał REQ. Arbiter próbuje ten sygnał na początku pierwszego cyklu zegara.
- Podczas pierwszego cyklu zegara B zgłasza zapotrzebowanie na użycie magistrali, potwierdzając swój sygnał REQ.
- W tym samym czasie arbiter potwierdza sygnał GNT-A, przekazując tym samym urządzeniu A dostęp do magistrali.
- Urządzenie nadrzędne, którym stało się urządzenie A, próbuje sygnał GNT-A na początku drugiego cyklu zegara i stwierdza, że uzyskało dostęp do magistrali. Zauważa też, że IRDY i TRDY są odwołane, z czego wynika, że magistrala jest wolna. Potwierdza więc FRAME, umieszcza informację adresową na szynie adresu i rozkaz na magistrali C/BE (nie pokazane). Kontynuuje także potwierdzanie REQ-A, ponieważ ma jeszcze drugą transakcję do przeprowadzenia po pierwszej.

- (e) Arbiter magistrali próbkuje wszystkie linie GNT na początku trzeciego cyklu zegara i decyduje, że udostępnia magistralę urządzeniu B w celu dokonania następnej transakcji. Potwierdza więc GNT-B i odwołuje GNT-A. Urządzenie B nie będzie mogło użyć magistrali, dopóki nie powróci ona do stanu jałowego.
- (f) A odwołuje FRAME w celu pokazania, że realizowane jest ostatnie (i jedyne) przesłanie. Umieszcza dane na magistrali danych i sygnalizuje to celowi za pomocą IRDY. Cel odczytuje dane na początku następnego cyklu zegara.
- (g) Na początku piątego cyklu zegara B stwierdza odwołanie IRDY i FRAME, może więc przejąć sterowanie magistralą, potwierdzając FRAME. Odwołuje także swój sygnał REQ, ponieważ chce przeprowadzić tylko jedną transakcję.

W rezultacie A uzyskuje dostęp do magistrali i może przeprowadzić swoją drugą transakcję.

Zauważmy, że arbitraż może mieć miejsce w tym samym czasie, w którym aktualna jednostka nadrzędna linii przesyła dane. Dzięki temu arbitraż nie powoduje utraty cyklu magistrali. Takie rozwiązanie nosi nazwę **arbitrażu ukrytego** (*hidden arbitration*).

3.6

MAGISTRALA FUTUREBUS+

Futurebus+ jest standardem asynchronicznej magistrali o wysokiej wydajności, opracowanym przez IEEE. Początkowa wersja, określana mianem Futurebus, została ogłoszona w dokumencie ANSI/IEEE Std 896 w roku 1987. Wersja ta była magistralą 32-bitową, w intencji twórców niezależną od technologii. Po roku 1987 standard ten został zrewidowany i poszerzony. Dla zaakcentowania różnic jest obecnie określany jako Futurebus+. Znak „+” odzwierciedla rozszerzalny charakter tego standardu. Możliwa jest jego dalsza ewolucja w celu sprostanienia nie przewidywanym obecnie potrzebom wynikającym ze szczególnych zastosowań. Podrozdział ten dotyczy aktualnej wersji tego standardu, pochodzącej z roku 1994.

Komitet Futurebus+ zdefiniował osiem wymagań, na których opiera się projekt [ANDR90]. Magistrala powinna:

- a być niezależna od architektury, procesora i technologii,
- a posługiwać się protokołem przesyłania asynchronicznego jako podstawowym,
- umożliwiać opcjonalne wykorzystywanie protokołu opartego na synchronizacji przez źródło,
- a mieć górną granicę wydajności nie wynikającą z technologii,

- a stosować w pełni rozproszone protokoły równoległe i arbitrażowe, wykorzystując zarówno protokoły z komutacją łączy (*circuit-switched*), jak i oparte na transakcjach rozgałęzionych (*split-transaction*),
- a obsługiwać systemy tolerujące błędy, o wysokiej niezawodności,
- a bezpośrednio obsługiwać pamięci wspólne wykorzystujące strukturę pamięci podręcznych,
- a zapewniać kompatybilne definicje transportu wiadomości.

Doniosłość Futurebus+ polega na tym, że standard ten będzie stosowany w większości współczesnych magistral w systemach mikroprocesorowych. Zacytujmy następującą obserwację [ANDR91]:

Komitet Futurebus+ stworzył doskonałość w zakresie ulepszania magistrali. Kiedy to dzieło ukończono, prawdopodobnie niewiele pozostało do zmodyfikowania dla przyszłych generacji projektantów magistrali. Gdy po pięciu latach od zatwierdzenia standardu przystępuje się do obowiązkowego przeglądu, komitet może spojrzeć na Futurebus+ i stwierdzić „nie ma nic do dodania”.

Główne różnice między standardem Futurebus z roku 1987 a aktualną specyfikacją Futurebus+ są następujące:

- a Futurebus wykorzystywał 32-bitowe magistrale danych; Futurebus+ używa magistral danych o szerokościach 32, 64, 128 i 256 bitów.
- a Futurebus uwzględniał protokół arbitrażu rozproszonego. Futurebus+ umożliwia działanie zarówno modelu rozproszonego, jak i scentralizowanego.
- a Tylko Futurebus+ zawiera 3-bitowe pole umożliwiające modułowi zadeklarowanie zdolności do przystosowania się do podstawowych rodzajów transakcji magistralowych.

Futurebus+ jest złożoną specyfikacją magistrali. Wprowadza pewne innowacyjne koncepcje w budowie magistrali. W rezultacie standard zawiera szereg nowych i ponownie zdefiniowanych wyrażeń, które staną się powszechne w ciągu kilku najbliższych lat; w tabeli 3.6 są przedstawione najważniejsze z nich. Natomiast w tabeli 3.7 są wymienione linie magistralowe określone w normie Futurebus+.

TABELA 3.6. Wybrane terminy standardu Futurebus+

Arbitraż równoległy	Blokowanie
Proces, w którym moduły umieszczają na równoległej magistrali swoje jednoznaczne liczby arbitrażowe oraz uwalniają sygnały zgodnie z algorytmem, w wyniku którego po pewnym czasie na magistrali pojawia się numer zwycięzcy.	Możliwość blokowania dostępu do danych innym modułom, gdy jest wymagany wyłączny dostęp jednego modułu do adresowanych danych.

TABELA 3.6 (od.)

Cykl (beat)	Zdarzenie rozpoczynające się od zmiany stanu linii synchronizacji spowodowanej przez moduł nadrzędny, po której następuje uwolnienie linii potwierdzania przez jeden lub wiele modułów podrzędnych. W pierwszej połowie cyklu mogą być przekazane rozkaz i dane od modułu nadrzędnego do jednego lub wielu modułów podrzędnych. W drugiej połowie cyklu moduły podrzędne mogą przekazać modułowi nadrzędnemu dane oraz informacje o stanie i możliwościach.	Protokół wymuszonego przesyłania danych	Niezależny od technologii mechanizm przesyłania, zgodnie z którym moduł podrzędny jest zmuszony do udzielenia odpowiedzi, zanim moduł nadrzędny przystąpi do następnego przesyłania.
Kadencja magistrali (bus tenure)	• Czas trwania sterowania magistralą przez moduł nadrzędny; jest to okres, podczas którego moduł ma prawo inicjowania i wykonywania transakcji magistralowych.	Transakcja magistralowa	Zdarzenie rozpoczynające się od fazy połączenia i kończące się wraz z fazą rozłączenia. Podczas transakcji magistralowej może nastąpić lub może nie nastąpić przesłanie danych.
Komunikat arbitrażowy	Liczba rozgłaszana na liniach komunikatów arbitrażowych i skierowana do wszystkich modułów na magistrali.	Transakcja rozszczepiona	Transakcja systemowa, w której zapotrzebowanie jest zgłaszane podczas jednej transakcji magistralowej, a odpowiedź jest nadawana podczas następnej, oddzielnej transakcji magistralowej.
Oczekiwanie (starvation)	Warunek systemowy, który występuje, gdy jeden lub wiele modułów nie wykonuje użytecznej pracy przez nieokreślony czas z powodu braku dostępu do magistrali lub do innych zasobów systemu.	Transakcja systemowa	Kompletna operacja, taka jak odczyt lub zapis w pamięci, widziana przez moduł inicjujący. Transakcja systemowa może być przełożona przez interfejs Futurebus+ na jedną lub wiele transakcji magistralowych w celu zakończenia operacji.

TABELA 3.7. Linie sygnałowe w standardzie Futurebus+

Oznaczenie	Liczba linii	Opis
<i>Linie informacyjne</i>		
AD(63÷0)	32, 64, 64, 64	Przenoszą adres podczas przesyłania (adresu); przenoszą dane podczas przesyłania danych.
D(255÷64)	_, _, 64, 192	Przenoszą dane podczas przesyłania danych.
BP(31÷0)	4, 8, 16, 32	Po jednej linii parzystości dla każdego 8 linii adresów i danych.
TG(7÷0)	8	Przenosi dodatkowe informacje związane z liniami adresów i danych; norma nie określa zastosowania.
TP	1	Kod parzystości linii znacznika.
CM(7÷0)	8	Przenoszą rozkazy dotyczące bieżącej transakcji od modułu nadrzędnego do jednego lub wielu modułów podrzędnych.
CP	1	Kod parzystości linii rozkazów.
ST(7÷0)	8	Przenoszą informacje o stanie modułu podrzędnego i jego gotowości do wymaganego przesyłania.
CA(2÷0)	3	Aktywowane przez moduł w celu zadeklarowania zdolności do realizowania podstawowych trybów transakcji magistralowych.
<i>Linie synchronizacji</i>		
AS, AK, AI	3	Synchronizacja adresu (AS). Potwierdzana przez moduł nadrzędny w celu poinformowania modułów podrzędnych, że informacje dotyczące adresu i rozkazu są ważne.

TABELA 3.7 (cd.)

Oznaczenie	Liczba linii	Opis
<i>Linie synchronizacji cd.</i>		
DS, DK, DI	3	Potwierdzenie adresu (AK). Uwalniana przez wszystkie moduły w celu pokazania modułowi nadrzędnemu, że ich informacja o stanie jest ważna.
		Odwrotne potwierdzenie adresu (AI). Uwalniana przez wszystkie moduły w celu pokazania modułowi nadrzędnemu, że ich informacja o stanie i zdolnościach jest ważna.
		Synchronizacja danych (DS). Potwierdzana przez moduł nadrzędny w celu poinformowania modułów podrzędnych, że zapisywane dane i rozkazy są ważne lub że moduł nadrzędny jest gotowy do otrzymania odczytanych danych.
		Potwierdzenie danych (DK), odwrotne potwierdzenie danych (DI). Uwalniane przez uczestniczące moduły podrzędne w celu pokazania modułowi nadrzędnemu, że informacje o stanie i odczytane dane są ważne lub że zostały przyjęte zapisywane dane.
		Uwalniana przez moduł nadrzędny w celu zawiadomienia modułu-elekta, że jest nowym modułem nadrzędnym.
ET	1	
<i>Linie komunikatów arbitrażowych</i>		
AB(7+0)	8	Przenoszą liczbę, która oznacza pierwszeństwo konkurentów podczas procesu przekazywania komunikatów arbitrażowych oraz procesu rozproszonego arbitrażu.
ABP	1	Kod parzystości dla AB(7+0).
AP, AQ, AR	3	Linie uzgadniania, które służą do cyklicznego uzgadniania sterującego procesem komunikatów arbitrażowych i procesem rozproszonego arbitrażu.
AC(1+0)	2	Sterują procesem komunikatów arbitrażowych i procesem rozproszonego arbitrażu.
<i>Linia inicjowania magistrali</i>		
RE	1	Inicjuje układy logiczne interfejsu wszystkich modułów z magistralą.
<i>Arbitraż scentralizowany</i>		
RQ(1+0)	2	Jedna lub obydwie linie są potwierdzane w celu zgłoszenia zapotrzebowania na nadzorowanie magistrali.
GR	1	Potwierdzana przez centralnego arbitra w celu przyznania nadrzędności nad magistralą.
PE	1	Potwierdzana przez centralnego arbitra w celu wskazania bieżącej jednostce nadrzędnej, że zaistniał warunek zmiany nadrzędności i że jednostka nadrzędna powinna zrzec się nadrzędności nad magistralą.
GA(4+0)	5	Unikatowy identyfikator przypisany każdemu gniazdu modułu fizycznego na magistrali i przyjmowany przez dowolny moduł włączony do tego gniazda.

Adresowanie

Każdemu modułowi na magistrali przypisuje się jednoznaczny adres geograficzny (określający położenie), który jest instalowany na stałe przez wykonanie odpowiednich połączeń w gnieździe, w którym montuje się moduł. Jeśli moduł

zostanie umieszczony w gnieździe, odczytuje 5-bitową liczbę na liniach GA (4÷0) w celu stwierdzenia, w którym gnieździe jest zlokalizowany. Jest to wielka zaleta: moduły mogą być wkładane i wyjmowane bez konieczności wyłączania systemu i bez potrzeby konfigurowania adresu.

Arbitraż

Proces arbitrażu w Futurebus+ odbywa się równolegle do przesyłania danych magistralą. Futurebus+ umożliwia przeprowadzenie zarówno rozproszonego, jak i scentralizowanego sposobu arbitrażu. Największe możliwości ma model rozproszony i w związku z tym jego stosowanie będzie najbardziej prawdopodobne.

Arbitraż rozproszony

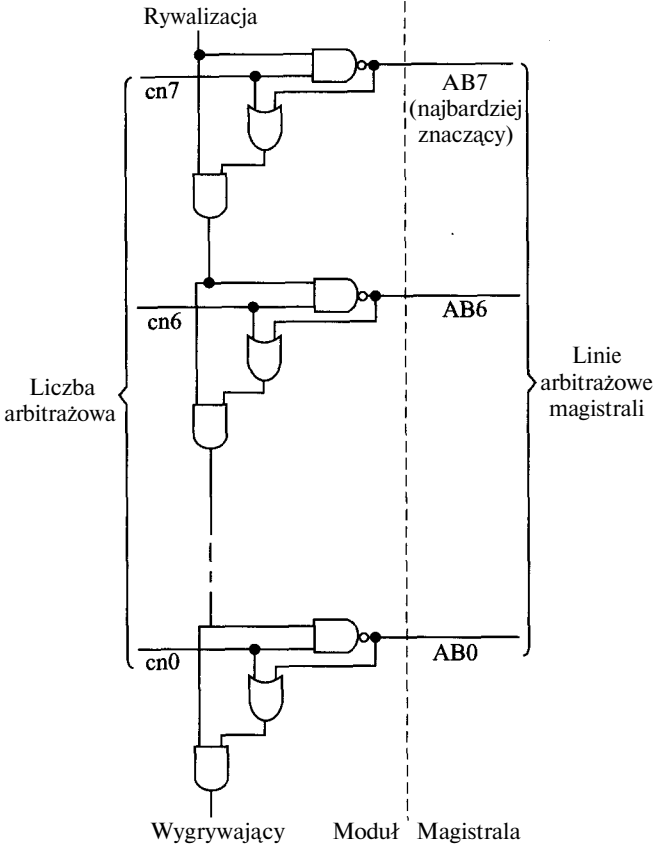
Każdy moduł na magistrali ma unikatową liczbę arbitrażową, wykorzystywaną w algorytmie równoległej rywalizacji o zarządzanie magistralą. Przed wyjaśnieniem procesu osiągania nadrzędności nad magistralą, musimy przeanalizować mechanizm rywalizacji.

W równoległej rywalizacji może wziąć udział dowolna liczba modułów, które wykorzystują do tego celu 8-bitową szynę komunikatów arbitrażowych. Wygrywa moduł, który doprowadzi do tej szyny największą liczbę. Każdy z rywali doprowadza swoją liczbę arbitrażową, $cn(7÷0)$, do linii $AB(7÷0)$. Sygnał na którejkolwiek linii jest realizacją operacji LUB (OR) na wszystkich sygnałach doprowadzonych do linii; linia ma więc stan logiczny 1, jeśli którykolwiek z rywalizujących modułów doprowadzi do niej 1. Układy logiczne arbitrażu w każdym module (rys. 3.25) odczytują wynikowe wartości szyny i modyfikują liczbę, którą moduł ten doprowadza do magistrali, posługując się przy tym następującą regułą. Jeśli którykolwiek z bitów liczby arbitrażowej modułu jest zerem, a odpowiedni bit linii arbitrażowej wykazuje 1, to wszystkie mniej znaczące bity liczby arbitrażowej modułu są odłączane od magistrali. Moduł o największej liczbie arbitrażowej stwierdzi, że jego liczba jest zgodna z liczbą pozostającą na magistrali, jest więc zwycięzcą.

Aby zrozumieć schemat logiczny³, zauważmy, że każdy z bitów modułu jest negowany przed doprowadzeniem do szyny, tak więc logiczna 1 jest reprezentowana przez poziom WYSOKI (HIGH) w samym module, natomiast przez poziom NISKI (LOW) w szynie i vice versa. W rezultacie bramka LUB (OR) wytworzy 0 (wymuszając przegraną) tylko wtedy, gdy bit cn (liczby arbitrażowej modułu) jest logicznym 0, a odpowiedni bit AB jest logiczną 1.

Proces arbitrażu wymaga, żeby każdy z konkurujących modułów (rozproszonych arbitrów) przechodził tę samą sekwencję stanów jednocześnie, w tempie najwolniejszego arbitra. Aby to osiągnąć, opracowano pomysłowy protokół

³ Patrz dodatek A, w którym są opisane cyfrowe układy logiczne.

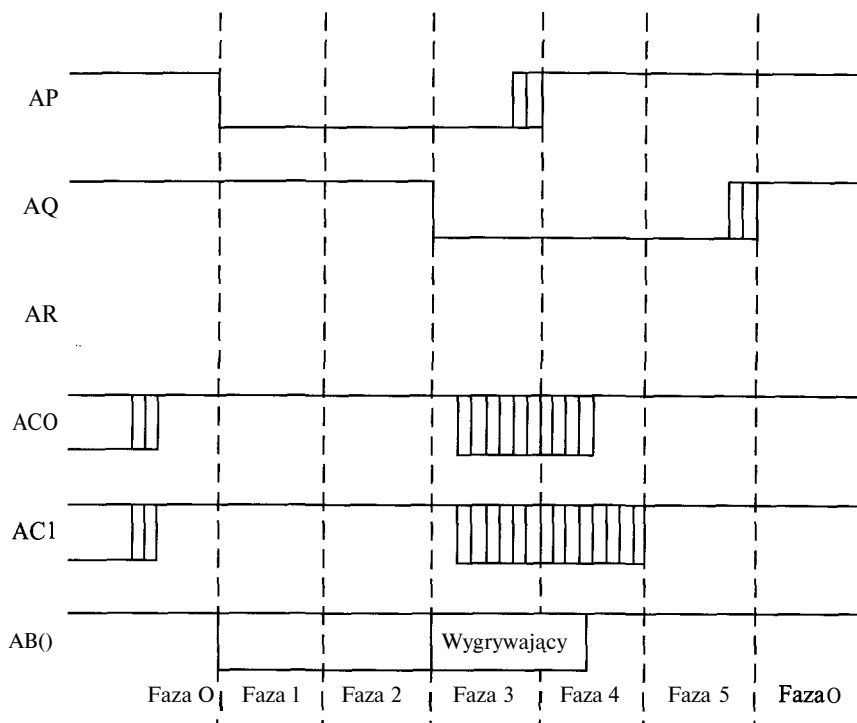


RYСУNEK 3.25. Układ logiczny arbitrażu Futurebus+

wykorzystujący trzy linie szyny (AP, AQ, AR). Protokół wykorzystuje inne linie magistralowe niż protokół przesyłania danych i dlatego może przebiegać równolegle w stosunku do transakcji przesyłania danych prowadzonych przez aktualną jednostkę nadrzędną magistrali.

Proces arbitrażu składa się z sześciu faz (rys. 3.26). Początek i koniec każdej fazy jest określany przez zmianę jednego z sygnałów synchronizujących arbitraż (AP, AQ, AR):

- ❑ *Faza 0 - bezczynność.* Identyfikowana przez zwolnione AP i AQ (HIGH) oraz potwierdzone AR (LOW); żadna rywalizacja nie ma miejsca.
- ❑ *Faza 1 - decyzja.* Jakikolwiek arbiter domagający się nadrzędności potwierdza AP. Każdy arbiter, który chce rywalizować, doprowadza swoją liczbę arbitrażową do AB (7÷0). Wszystkie moduły synchronizują się z procesem arbitrażu, uwalniając AR. Tylko wtedy, gdy wszystkie moduły uwolniły AR, jego stan jest WYSOKI (HIGH), co sygnalizuje koniec fazy 1.
- Q *Faza 2 - rywalizacja.* Podczas tej fazy rywale monitorują swój wewnętrzny sygnał „wygranej” przez czas, który pozwala układom logicznym rywaliza-

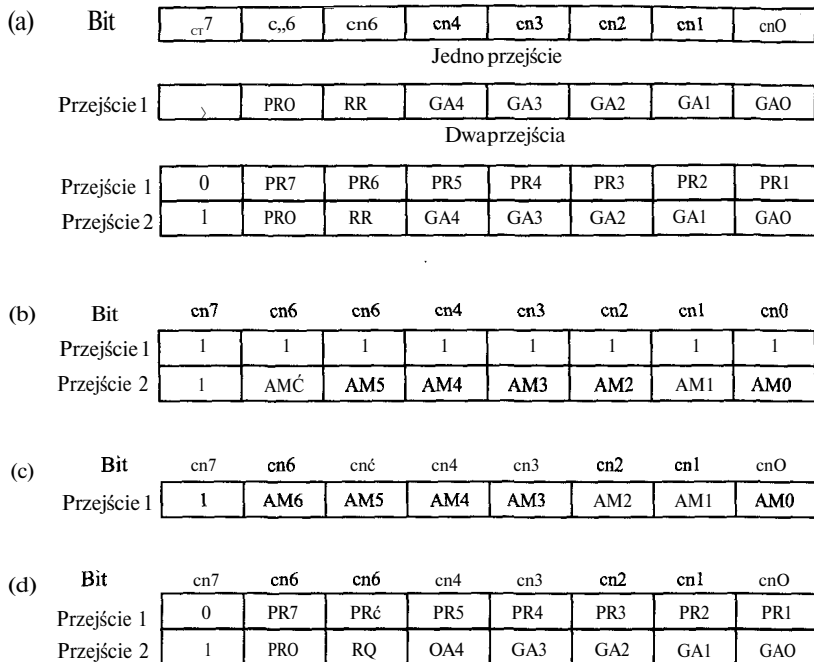


RYSUNEK 3.26. Fazy arbitrażu Futurebus+

cji (rys. 3.25) ustalić stan końcowy. Zwycięzca potwierdza AQ i staje się Jednostką nadrzędną – elektem".

- a *Faza 3 - sprawdzanie błędu.* Podczas tej fazy wszyscy przegrani sprawdzają, czy liczba wygrywająca współzawodnictwo jest większa niż ich własna. Wszystkie moduły wykrywające warunek błędu potwierdzają ACO i AC1. Faza ta trwa tak długo, jak długo aktualna jednostka nadrzędna kontroluje magistralę. Aktualna jednostka nadrzędna sygnalizuje koniec transakcji przez uwolnienie AS. Po wykryciu uwolnienia AS każdy moduł uwalnia AP.
- Q *Faza 4 - uwolnienie jednostki nadrzędnej.* Liczba wygrywająca jest usuwana z AB ($7 \div 0$). Jeśli jednostka nadrzędna chce zachować magistralę, potwierdza AC1; w przeciwnym razie potwierdza AR.
- *Faza 5 - ustanowienie jednostki nadrzędnej.* Wszystkie moduły zachowują stan ACO i AC1 w celu stwierdzenia, czy zachodzi przekazanie zarządzania, czy też stosowne jest jakieś inne działanie. Po określeniu stanu moduły uwalniają AQ. Gdy wszystkie moduły uwolnią AQ, faza ta jest zakończona i może się rozpocząć nowy cykl arbitrażu.

Jeśli powyższa sekwencja kroków zachodzi raz, a na końcu fazy 5 zarządcą-elekt przejmie zarządzanie, to sytuacja taka jest określana jako arbitraż jedno-



RYСУNEK 3.27. Komunikaty arbitrażowe Futurebus+: (a) pola zapotrzebowania arbitrażu rozproszonego (arbitraż rozproszony); (b) pola komunikatów arbitrażu rozproszonego (arbitraż rozproszony); (c) pola komunikatów arbitrażu centralnego (arbitraż centralny); (d) pola komunikatów arbitrażu centralnego (arbitraż centralny)

przejściowy (*single-pass arbitration*). W pozostałych przypadkach wymagane są dwa przejścia. Mamy wtedy do czynienia z większą liczbą zwycięzców i fazy od 0 do 5 są powtarzane.

Wartości liczb arbitrażowych stosowanych przez moduły podczas rywalizacji określają, czy wymagane jest jedno przejście, czy też dwa. Każde przejście cyklu arbitrażu polega na rywalizacji wykorzystującej 8-bitowe liczby arbitrażowe doprowadzone przez każdy moduł. Jak widać na rys. 3.27a, liczba arbitrażowa składa się z trzech pól: pola priorytetu, pola cyklicznego i pola adresu geograficznego. Liczba arbitrażowa modułu jest nakładana na jedną lub dwie liczby uzyskane podczas rywalizacji. Bit cn7 wskazuje, czy moduł wymaga następnego przejścia arbitrażu, tak więc jednoprzejściowa liczba arbitrażowa pokonuje liczbę dwuprzejściową.

Ośmiobitowe pole priorytetu PR(7÷0) służy do określenia jednego z 256 poziomów priorytetu (255 = najwyższy, 0 = najniższy). Na dwóch najwyższych poziomach priorytetu, przy których wszystkie bity poza PRO są jedynkami, wykorzystuje się jednoprzejściowy cykl rywalizacji⁴.

⁴ Jest to możliwe, jeśli bowiem wszystkie PR (7÷1) są ustawione na 1, to zgłaszający zapotrzebowanie zawsze wygra pierwsze przejście, można to więc w tym przypadku po-

W niezbyt obciążonym systemie mogą występować przypadki, w których nie następuje zawieszanie zapotrzebowania na magistralę. Aby zaoszczędzić czas, w takich warunkach aktualna jednostka nadrzędna zachowuje kontrolę i może rozpocząć nową transakcję bez potrzeby arbitrażu. Tylko wtedy, gdy jednostka nadrzędna stwierdzi, że miał miejsce arbitraż i istnieje jednostka nadrzędna-elekt, może ona zdecydować się na przekazanie władzy elektowi. Nawet wówczas, jeśli aktualna jednostka nadrzędna ma wyższy priorytet niż elekt, może ona zachować sterowanie magistralą.

Jednostka nadrzędna-elekt musi odczekać pewien czas, aż aktualna jednostka nadrzędna zakończy używanie magistrali. W tym czasie moduł o wyższym priorytecie może potrzebować magistrali. Jeśli tak jest, moduł ten może wymusić nowy cykl arbitrażu i nową rywalizację w celu określenia nowej jednostki nadrzędnej-elekta. Wtedy cykl rozpoczyna się od fazy 1.

Możliwość działania cyklicznego (*round-robin*) zapewnia sprawiedliwy i dający równe szansę proces przyznawania nadrzędności wśród modułów o takim samym priorytecie. Wartość bitu cyklicznego dla danego modułu jest dostawiana każdorazowo, gdy następuje przekazanie nadrzędności modułowi o takim samym priorytecie. Bit ten jest ustawiany, jeśli nadrzędność jest przekazywana modułowi o tym samym priorytecie, lecz o wyższym adresie geograficznym. Jest on natomiast zerowany w przypadku przejmowania nadrzędności przez moduł o tym samym priorytecie, lecz o niższym adresie geograficznym. Jeśli bit cykliczny jest zerowany, to moduł przegrywa w następnym cyklu arbitrażu z modułami o tym samym priorytecie, dla których ten bit jest 1. W rezultacie jest zapewnione cykliczne przekazywanie nadrzędności w warunkach dużego obciążenia magistrali.

Liczbę arbitrażową jednostki nadrzędnej-elekta widzą wszystkie moduły, może więc ona być też wykorzystywana do rozgłaszania informacji w całym systemie. Określamy to mianem **komunikatu arbitrażowego** (*arbitration message*). Można go użyć do przerwań obejmujących system lub do powiadomień nadzwyczajnych.

Komunikaty arbitrażowe wyglądają podobnie do innych żądań priorytetu, są zgodne z sekwencją przedstawioną powyżej i zilustrowaną na rys. 3.26. Komunikat arbitrażowy (rys. 3.27b) wykorzystuje same jedynki podczas pierwszego przejścia w celu pokonania innych komunikatów arbitrażowych o priorytecie niższym od 254. Aktualny komunikat jest przekazywany podczas drugiego przejścia; do systemu należy przypisanie znaczenia każdej liczbie składającej się na ten komunikat.

Jedyną zmianą w cyklu arbitrażu jest to, że wydawca komunikatu arbitrażowego nie zakłada przejścia nadrzędności po wygranu arbitrażu.

minąć. Dla usprawnienia, jeśli system ma tylko jeden lub dwa poziomy priorytetu, powinny być używane priorytety 255 i 254.

Arbitraż scentralizowany

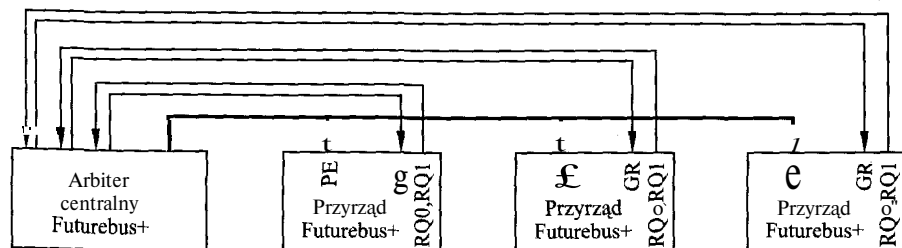
System Futurebus+ może być również zorganizowany na zasadzie arbitrażu centralnego (rys. 3.28). W modelu takiego arbitrażu wykorzystuje się linie RQ1, RQ0, GR i PE. Linie RQ i GR nie są aktualnie składnikami wspólnej magistrali; istnieją natomiast dwie oddzielne linie zapotrzebowania i jedna linia nadawania (*grant*) od centralnego arbitra do każdego modułu na magistrali. Konfiguracja taka zapobiega konieczności ustalania adresu w warunkach, gdy mamy do czynienia z wielomajędnoczesnymi żądaniami zapotrzebowania.

Moduł może potwierdzić RQ0 lub RQ1 w celu zgłoszenia zapotrzebowania na magistralę na jednym z dwóch poziomów priorytetu. Aktualny priorytet każdego modułu jest określany przez arbitra na podstawie zidentyfikowania modułu oraz poziomu priorytetu, na którym zgłosił on zapotrzebowanie. Arbitr wskazuje moduł, który jest nową jednostką nadrzędną-elektrem, przez potwierdzenie linii GR prowadzącej do tego modułu. Następnie moduł może przejąć nadrzędność nad magistralą, kiedy wykryje, że linia ET jest uwolniona przez aktualną jednostkę nadrzędną. Arbitr centralny może także potwierdzić PE w celu zasygnalizowania aktualnej jednostce nadrzędnej, że powinna przekazać sterowanie jednostce nadrzędnej-elektowi tak szybko, jak to jest możliwe.

Gdy jest wykorzystywany arbitr centralny, linie komunikatów arbitrażowych pozostają dostępne i są one używane wyłącznie do transmisji komunikatów arbitrażowych. Przewidziano dwa rodzaje komunikatów: ogólne i centralne komunikaty arbitrażowe (rys. 3.27c i d). Wykorzystany system numerowania zapewnia, że komunikaty ogólne mają pierwszeństwo w stosunku do centralnych.

Komunikaty ogólne są przeznaczone do rozgłaszania wiadomości innym modułom na magistrali. Podobnie jak w przypadku arbitrażu rozproszonego, mogą one być wykorzystywane do przekazywania przerwań lub wiadomości nadzwyczajnych.

Komunikaty centralne są kierowane do arbitra i stanowią środek przekazywania mu informacji o priorytecie. Pole RQ wskazuje, czy RQ0, czy też RQ1 jest przedmiotem tej wiadomości. Pole priorytetu PR(7÷0) służy do wybrania poziomu priorytetu, który ma być przypisany wybranej dla tego modułu linii RQ.



RYСУNEK 3.28. Arbiter centralny Futurebus+

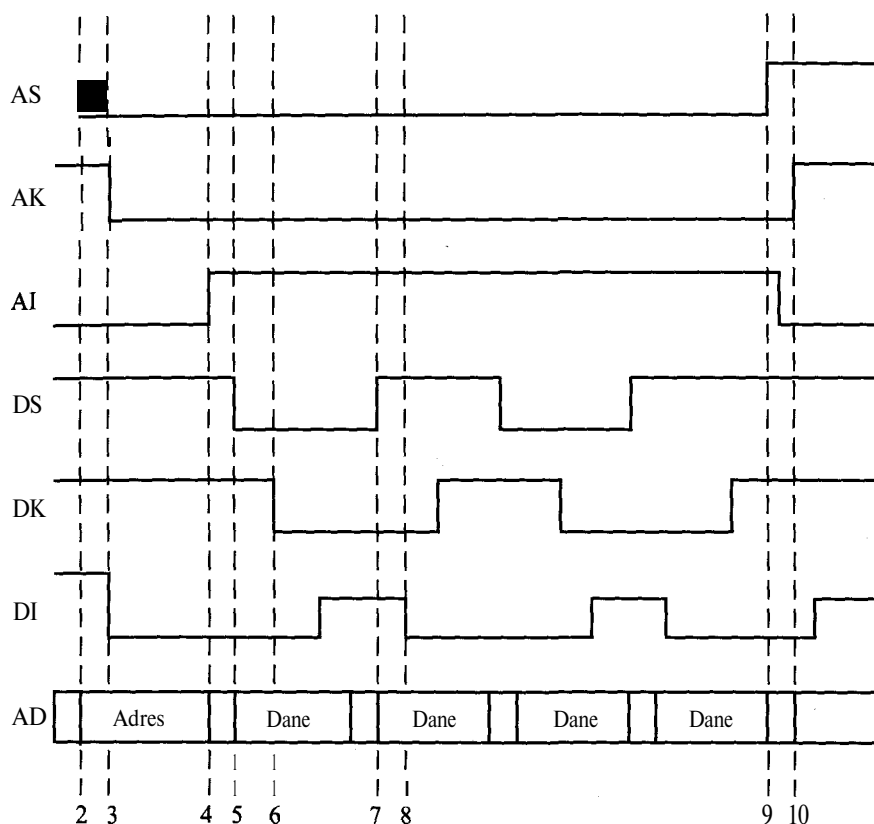
Przesyłanie danych

Futurebus+ zapewnia wiele różnych możliwości transferu danych. W tym punkcie przedstawiamy przykład zapisu wzięty z [JONE91], jako ilustrację podejścia stosowanego w Futurebus+.

Ogólnie rzecz biorąc, w skład transakcji Futurebus+ wchodzi: cykl adresu (*address beat*) oraz od zera do wielu cykli danych (*data beat*). W naszym przykładzie przesyłany jest blok danych złożony z 4 słów. Sekwencja zdarzeń jest następująca (rys. 3.29):

1. Nadawca przejmuje sterowanie magistralą, wykorzystując jedną z metod arbitrażu opisanych powyżej.
2. Nadawca ładuje adres i informację o rozkazie do magistrali. Wskazuje to, potwierdzając AS.
3. Pozostałe moduły odczytują adres. Następnie każdy moduł potwierdza AK w celu wskazania, że odczytał adres. Tak więc linia AK magistrali będzie w stanie potwierdzonym natychmiast po uzyskaniu odpowiedzi od najszybszego modułu.
4. Każdy moduł uwalnia także AI w tym samym czasie, w którym potwierdza AK. Linia AI nie jest uwolniona, aż do czasu uzyskania odpowiedzi od najwolniejszego modułu. Sygnalizuje to zakończenie fazy adresu. Jednostka nadrzędna może teraz bezpiecznie usunąć adres z magistrali.
5. Jednostka nadrzędna ładuje teraz dane do magistrali, sygnalizując to potwierdzeniem DS.
6. Odbiorca przyjmuje dane z magistrali i potwierdza DK, po czym uwalnia DI w celu zasygnalizowania udanego odbioru.
7. Jednostka nadrzędna ładuje następne słowo danych do magistrali i uwalnia DS, sygnalizując ważność danych.
8. Odbiorca przyjmuje dane z magistrali i potwierdza DI, po czym uwalnia DK, sygnalizując udany odbiór.
9. W ten sposób jest kontynuowane przesyłanie danych, za każdym razem po jednym słowie, z wykorzystaniem transmisji DS i DK/DI, aż jednostka nadrzędna nie będzie już miała danych do wysłania. Sygnalizuje ona zakończenie transakcji przez uwolnienie AS.
10. Odbiorca, a także wszystkie inne podległe moduły, potwierdzają uwolnienie AS przez potwierdzenie AI oraz uwolnienie AK. Kończy to transakcję.

Przykład ten ilustruje sposób uniezależnienia się standardu Futurebus+ od technologii. Każde zdarzenie jest powiązane przyczynowo tylko z poprzednim zdarzeniem, bez jakichkolwiek innych ograniczeń czasowych.



RYСУNEK 3.29. Przykład transferu danych magistralą Futurebus+

Podsumowanie

Specyfikacja Futurebus+ jest jednym z najbardziej złożonych technicznie standardów magistrali jakie kiedykolwiek opracowano. Standard obejmuje warstwę logiczną, warstwę fizyczną wraz z profilami, zalecane rozwiązania praktyczne oraz odniesienia do kilku innych standardów. Wynikiem jest specyfikacja magistrali, która może być stosowana jako lokalna magistrala procesor-pamięć lub może konkurować z magistralą PCI w celu obsłużenia szybkich urządzeń peryferyjnych. Zarówno PCI, jak i Futurebus+ umożliwiają bardzo szybkie przekazywanie danych. Podstawową różnicą jest to, że standard PCI zaprojektowano mając na uwadze niski koszt wdrażania oraz małe rozmiary, podczas gdy Futurebus+ ma zapewnić elastyczność i funkcjonalność w celu zaspokojenia zapotrzebowania na różnorodne systemy o wysokiej wydajności, zwłaszcza kosztowne.

POLECANA LITERATURA

Literatura na temat magistral i innych struktur połączeń jest zaskakująco skromna. W [ALEX93] zawarto głębsze omówienie struktur magistralowych i problemów przesyłania za pomocą magistrali, uwzględniając kilka szczególnych przypadków magistrali.

Najbardziej klarowny opis magistrali PCI zawarto w [SHAN94a]. Także [SOLA94] zawiera wiele solidnych informacji o PCI.

ALEX93 Alexandridis N.: *Design of Microprocessor-Based Systems*. Englewood Cliffs, Prentice Hall, 1993.

SHAN94a Shanley T.: Anderson D.: *PCI Systems Architecture*. Richardson, Mindshare Press, 1994.

SOLA94 Solari E., Willse G.: *PCI Hardware and Software: Architecture and Design*. San Diego, Annabooks, 1994.

PROBLEMY DO ROZWIĄZANIA

3.1. Dla każdego rozkazu IAS (zob. tab. 2.1) sprecyzuj sekwencję stanów (rys. 3.6) dla cyklu rozkazu.

3.2. Hipotetyczna maszyna z rys. 3.4 ma również dwa rozkazy wejścia-wyjścia:

0011 = Ładuj akumulator wejścia-wyjścia

0111 = Przenieś zawartość akumulatora do wejścia-wyjścia

W tych przypadkach 12-bitowy adres identyfikuje urządzenie wejścia-wyjścia. Wykorzystując format z rys. 3.5 pokaż realizację poniższego programu:

1. Ładuj akumulator z urządzenia 5.

2. Dodaj zawartość pamięci z pozycji 940.

3. Zawartość akumulatora przenieś do urządzenia 6.

3.3. Rozważ system komputerowy, który zawiera moduł wejścia-wyjścia sterujący prostym dalekopisem z klawiaturą. W CPU zawarte są następujące rejestry, podłączone bezpośrednio do magistrali systemowej:

INPR: Rejestr wejściowy - 8 bitów,

OUTR: Rejestr wyjściowy - 8 bitów,

FGI: Znacznik wejściowy - 1 bit,

FGO: Znacznik wyjściowy - 1 bit,

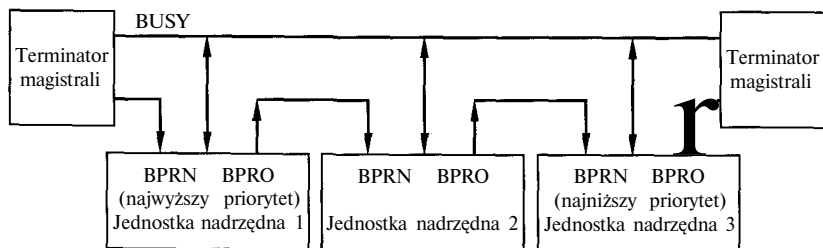
IEN: Zezwolenie przerywania - 1 bit.

Wejście z klawiatury dalekopisu i wyjście do drukarki dalekopisu są sterowane przez moduł wejścia-wyjścia. Dalekopis może kodować sym-

bole alfanumeryczne w postaci słów 8-bitowych oraz dekodować słowa 8-bitowe na symbole alfanumeryczne.

- (a) Opisz, jak procesor może uzyskać dostęp do wejścia-wyjścia z dalekopisem, posługując się pierwszymi czterema z wymienionych rejestrów.
- (b) Opisz, jak można tę funkcję zrealizować efektywniej, wykorzystując także ostatni rejestr.

3.4. Na rysunku 3.30 widać schemat układu arbitrażu rozproszonego, który może być stosowany z Multibus I. Moduły fizycznie są połączone łańcuchowo, według kolejności priorytetów. Moduł z lewej strony rysunku otrzymuje stały *priorytet na magistrali*, w postaci sygnału BPRN wskazującego, że żaden z modułów o wyższym priorytecie nie domaga się kontroli nad magistralą. Jeśli moduł nie żąda dostępu do magistrali, potwierdza swoją linię *wyłączenia priorytetu*, BPRO. Na początku cyklu zegara dowolny moduł może *życzyć* sobie sterowania magistralą przez ustalenie niskiego stanu linii BPRO. Obniża to stan linii BPRN następnego modułu w łańcuchu, przez co z kolei jest on zmuszany do obniżenia stanu swojej linii BPRO. Sygnał propaguje więc wzdłuż łańcucha. Na zakończenie tej „reakcji łańcuchowej” powinien być tylko jeden moduł, którego linia BPRN jest potwierdzona, a którego linia BPRO jest w stanie niskim. Ten właśnie moduł ma priorytet. Jeśli na początku cyklu magistrali nie jest ona zajęta (linia BUSY nieaktywna), to moduł mający priorytet może przejąć kontrolę nad magistralą, potwierdzając BUSY.



RYСУNEK 3.30. Arbitraż rozproszony Multibus I

Propagacja sygnału BPR od modułu o najwyższym priorytecie do modułu o najniższym priorytecie zabiera pewien czas. Wyjaśnij, czy ten czas musi być krótszy od cyklu zegara.

3.5. System pamięciowy składa się z modułów pamięci połączonych wspólną magistralą. Gdy zgłaszane jest żądanie zapisu, magistrala jest zajmowana na 100 ns przez sygnały danych, adresu i sterujące. Podczas tych samych 100 ns oraz w ciągu następnych 500 ns zaadresowany moduł pamięci wykonuje jeden cykl, przyjmując i magazynując dane. Działania modułów pamięci mogą się nakładać, ale w określonym czasie na magistrali może się znajdować tylko jedno żądanie.

- (a) Załóżmy, że do magistrali podłączono 8 takich modułów. Jaka jest maksymalna możliwa szybkość (w słowach na sekundę), z którą mogą być zapisywane dane?
- (b) Naskicuj wykres maksymalnej szybkości zapisu jako funkcji czasu cyklu modułu zakładając, że modułów jest 8 i że magistrala jest zajmowana przez 100 ns.

3.6. W celu zaoszczędzenia bramek magistrale są często multipleksowane czasowo. Rozważ maszynę o słowie 48-bitowym, z dyskiem 10^7 bit/s oraz z pamięcią główną o czasie cyklu równym 600 ns. Załóżmy, że każda transmisja przez magistralę wymaga 750 ns dla bitów danych i różnych operacji sterujących. Ile bitów danych można wysłać w każdym okresie 750 ns, żeby zachować wyprzedzenie w stosunku do dysku, i jaki typ magistrali należy wybrać? Jaka część szerokości pasma pamięci głównej jest zużywana na operację wejścia-wyjścia związaną z dyskiem?

Naskicuj czasową sekwencję zdarzeń zawartych w ciągłej transmisji z dysku do pamięci głównej; tzn. pokaż, jaką część szerokości pasma angażuje dysk, magistrala i pamięć główna.

3.7. Biorąc pod uwagę priorytety przy połączeniu łańcuchowym (*daisychain*), załóżmy, że urządzenia $d_1, \dots, d_k$ mają być połączone łańcuchowo i że każde z urządzeń, d_i , wykorzystuje część a_i pojemności magistrali:

$$0 < a_i < 1; \quad \sum_{i=1}^k a_i < 1$$

a pojemność magistrali ma znormalizowaną wartość 1. Jak powinny być urządzenia dołączone do łańcucha, żeby osiągnąć maksymalną przeciętną pojemność pozostającą? Pojemność pozostająca dla urządzenia w łańcuchu wynosi 1 minus suma pojemności wykorzystywanych przez wszystkie urządzenia o wyższym priorytecie znajdujące się w łańcuchu. Krótko skomentuj uzyskane wyniki. Daj przykład sytuacji, w której mogą one nie być odpowiednie.

3.8. Magistrale VAX SBI wykorzystują rozproszony arbitraż synchroniczny. Każde urządzenie SBI (np. procesor, pamięć, adapter Unibus) ma jednoznaczny priorytet i jest przypisane do unikalnej linii zapotrzebowania na transfer (TR). SBI ma 16 takich linii (TRO, TR1, ..., TR15), przy czym TRO ma najwyższy priorytet. Gdy urządzenie chce wykorzystać magistralę, rezerwuje przyszły przedział czasu (*time slot*) przez potwierdzenie swojej linii TR podczas bieżącego przedziału czasowego. Przy końcu bieżącego przedziału czasowego, każde urządzenie o zawieszonym rezerwacji bada linię TR; urządzenie o najwyższym priorytecie, posiadające rezerwację, wykorzystuje najbliższy przedział czasowy.

Do magistrali można dołączyć maksymalnie 17 urządzeń. Urządzenie o priorytecie 16 nie ma linii TR. Dlaczego?

- 3.9. Paradoksalnie, urządzenie o najniższym priorytecie ma zwykle najkrótszy średni czas oczekiwania. Z tej przyczyny CPU ma zwykle najniższy priorytet na magistrali SBI. Dlaczego urządzenie o priorytecie 16 ma zwykle najkrótszy średni czas oczekiwania? W jakich warunkach może to nie być prawdą?
- 3.10. Rozważmy hipotetyczny mikroprocesor 32-bitowy mający 32-bitowe rozkazy złożone z dwóch pól: pierwszy bajt zawiera kod operacji, a pozostałe argument lub adres argumentu.
- Jaka jest maksymalna pojemność pamięci adresowalnej bezpośrednio (w bajtach)?
 - Przedyskutuj wpływ na szybkość systemu, jeśli magistrala mikroprocesora ma:
 - 32-bitową lokalną szynę adresową i 16-bitową lokalną szynę danych, lub
 - 16-bitową lokalną szynę adresową i 16-bitową lokalną szynę danych.
 - Ilu bitów wymaga licznik programu, a ilu rejestr rozkazów?

Źródło: [ALEX93].

- 3.11. Rozpatrzmy hipotetyczny mikroprocesor generujący adresy 16-bitowe (załóżmy np. że licznik programu i rejestry adresu są 16-bitowe) i dysponujący 16-bitową szyną danych.
- Jaka jest maksymalna przestrzeń adresowa pamięci, do której procesor może mieć bezpośredni dostęp, jeśli jest połączony z pamięcią 16-bitową?
 - Jaka jest maksymalna przestrzeń adresowa pamięci, do której procesor może mieć bezpośredni dostęp, jeśli jest połączony z pamięcią 8-bitową?
 - Jaka cecha architektury pozwoli temu mikroprocesorowi na dostęp do oddzielnej przestrzeni wejścia-wyjścia?
 - Jeśli rozkaz wejścia i wyjścia może precyzować adres portu wejścia-wyjścia, ile 8-bitowych portów wejścia-wyjścia może obsługiwać mikroprocesor, a ile portów 16-bitowych? Odpowiedź uzasadnij.

Źródło: [ALEX93].

- 3.12. Rozważmy mikroprocesor 32-bitowy z 16-bitową zewnętrzną szyną danych, sterowany zegarem wejściowym 8 MHz. Załóżmy, że mikroprocesor ma cykl magistrali, którego maksymalny czas trwania jest równy 4 cyklom zegara wejściowego. Jaka jest maksymalna szybkość transferu danych, z którą ten mikroprocesor może pracować? Aby zwiększyć wydajność, czy byłoby lepiej zastosować 32-bitową szynę danych, czy też podwoić częstotliwość zewnętrznego zegara dołączonego do mikroprocesora? Sprecyzuj inne możliwe założenia i wyjaśnij.

Źródło: [ALEX93].

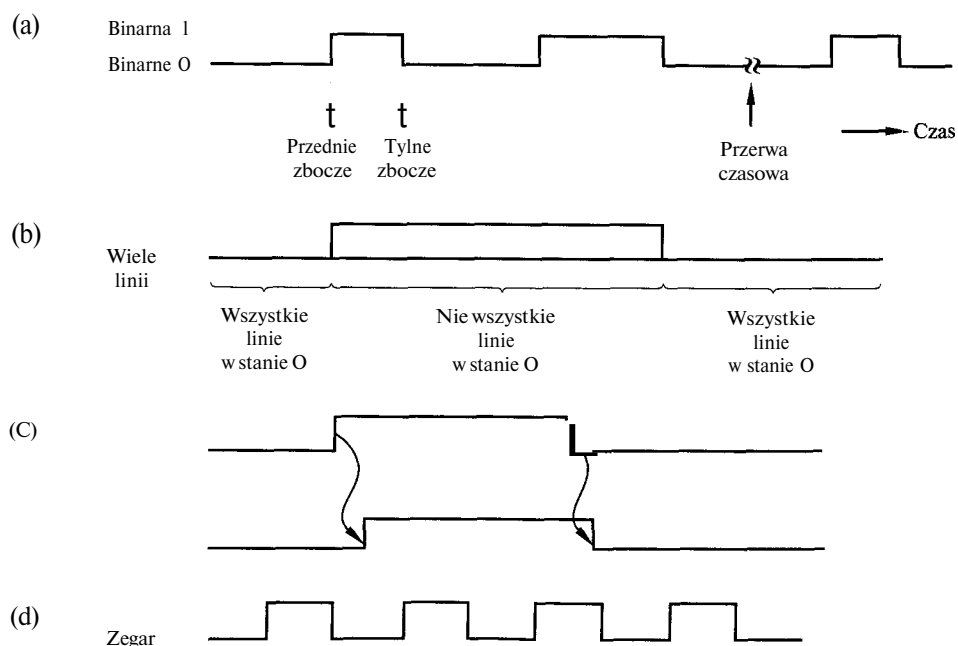
3.13. Nakreśl i wyjaśnij przebiegi czasowe dla operacji zapisu PCI (podobne do rys. 3.22).

DODATEK 3A

Wykresy przebiegów czasowych

W tym rozdziale wykorzystywaliśmy wykresy przebiegów czasowych w celu zilustrowania sekwencji zdarzeń i zależności między zdarzeniami. Czytelnikowi, któremu te wykresy nie są znane, dodatek ten dostarczy niezbędnych wyjaśnień.

Komunikacja między urządzeniami podłączonymi do magistrali zachodzi za pomocą zbioru linii służących do przenoszenia sygnałów. Transmitowane sygnały mogą mieć dwa różne poziomy napięcia, reprezentujące binarne 0 i binarną 1. Wykres czasowy pokazuje poziom sygnału na linii jako funkcję czasu (rys. 3.31a). Zgodnie z przyjętą konwencją poziom sygnału binarnej 1 jest przedstawiany jako wyższy niż binarnego 0. Zwykle binarne 0 ma wartość zaniedbywalną. Jeśli więc żadne dane ani żadne sygnały nie są transmitowane, poziom na linii reprezentuje binarne 0. Przejście sygnału od 0 do 1 jest zwykle określane jako zbocze narastające; przejście od 1 do 0 określa się jako zbocze opadające. Dla uproszczenia, zmiany sygnałów są często przedstawiane jako natychmiastowe. W rzeczywistości zajmują czas niezerowy, jednak jest on zwy-



RYSUNEK 3.31. Przebiegi czasowe: (a) sygnał jako funkcja czasu; (b) grupy linii; (c) zależności typu przyczyna-skutek; (d) sygnał zegarowy

kle krótki w porównaniu z czasem trwania sygnałów. Na wykresie przebiegu czasowego może się zdarzyć, że pomiędzy interesującymi zdarzeniami upływa zmienna lub przynajmniej nieistotna ilość czasu. Prezentujemy to, stosując przerwę w wykresie.

Sygnały są czasem rysowane w grupach (rys. 3.31b). Na przykład, jeśli dane są przesyłane bajtami, wymaganych jest 8 linii. Ogólnie nie musimy znać dokładnych wartości przesyłanych w takich grupach; musimy tylko wiedzieć, czy sygnały są obecne, czy nie. Zmiana sygnału na linii może spowodować, że dołączone urządzenie zmieni sygnały na innych liniach. Jeśli na przykład moduł pamięci wykryje sygnał sterujący odczytu (przejście do 0 lub 1), to umieści sygnały danych na liniach danych. Takie związki przyczynowo-skutkowe powodują sekwencje zdarzeń. W celu pokazania tych zależności na wykresach przebiegów czasowych wykorzystuje się strzałki (rys. 3.31c).

Częścią magistrali systemowej jest zwykle linia zegarowa. Do linii zegarowej jest dołączony zegar elektroniczny, wywołujący powtarzalne, regularne sekwencje zmian (rys. 3.31d). Za pomocą sygnału zegarowego mogą być synchronizowane inne zdarzenia.

Pamięć wewnętrzna

Pamięć komputera, pozornie prostą koncepcyjnie, cechuje najszerszy zakres rodzajów, technologii, organizacji, wydajności i kosztów w stosunku do pozostałych składników. Nie istnieje jedna technologia, która optymalnie spełniałaby wymagania systemu komputerowego dotyczące pamięci. W rezultacie typowy system komputerowy zawiera hierarchię podsystemów pamięci: niektóre z nich są wewnętrzne w stosunku do systemu (bezpośrednio dostępne dla procesora), a inne zewnętrzne (dostępne dla procesora poprzez moduł wejścia-wyjścia).

W tym rozdziale skoncentrujemy się na składnikach pamięci wewnętrznej, natomiast rozdz. 5 jest poświęcony pamięci zewnętrznej. W pierwszym podrozdziale zajmiemy się głównymi własnościami pamięci komputerowych. Następnie omówimy półprzewodnikowe podsystemy pamięci głównej, w tym pamięci ROM, DRAM i SRAM. W dalszej części opiszemy podstawowy element wszystkich nowoczesnych systemów komputerowych: pamięć podręczną. Na zakończenie powrócimy do pamięci DRAM i rozpatrzmy bardziej zaawansowane rozwiązania architektoniczne tych pamięci.

4.1

PRZEGLĄD SYSTEMÓW PAMIĘCI KOMPUTEROWYCH

Własności systemów pamięci

Złożone zagadnienia pamięci komputerowych można opanować łatwiej, jeśli podzieli się systemy pamięciowe według ich podstawowych własności. Najważniejsze z tych własności są wymienione w tabeli 4.1.

Rozpoczynamy od najłatwiej dostrzegalnego aspektu pamięci: jej *położenia*. Jak sugerują tytuły rozdziałów, istnieją pamięci zarówno wewnętrzne, jak i zewnętrzne w stosunku do komputera. Pamięć wewnętrzna jest często identyfikowana z pamięcią główną. Są jednak inne formy pamięci wewnętrznej. Procesor wymaga własnej pamięci lokalnej w postaci rejestrów (patrz np. rys. 2.3). Ponadto, jak zobaczymy, stanowiąca część procesora jednostka sterująca może również potrzebować własnej pamięci wewnętrznej. Dyskusję na temat tych

TABELA 4.1. Podstawowe własności komputerowych systemów pamięciowych

Położenie	Wydajność
procesor	czas dostępu
wewnętrzna (główna)	czas cyklu
zewnętrzna (pomocnicza)	szybkość transferu
Pojemność	Rodzaj fizyczny
rozmiar słowa	półprzewodnikowa
liczba słów	magnetyczna
Jednostka transferu	Własności fizyczne
słowo	ulotna/nieulotna
blok	wymazywalna/niewymazywalna
Sposób dostępu	Organizacja
sekwencyjny	
bezpośredni	
swobodny	
skojarzeniowy	

dwóch ostatnich typów pamięci wewnętrznej pozostawiamy do następnych rozdziałów. Pamięć zewnętrzna składa się z peryferyjnych urządzeń pamięciowych, takich jak pamięci dyskowe i taśmowe, które są dostępne dla procesora poprzez sterowniki wejścia-wyjścia.

Oczywistą własnością pamięci jest jej **pojemność**. W przypadku pamięci wewnętrznej jest ona zwykle wyrażana w bajtach (1 bajt = 8 bitów) lub w słowach. Powszechnymi długościami słów są: 8, 16 i 32 bity. Pojemność pamięci zewnętrznej jest zwykle wyrażana w bajtach.

Parametrem związanym z pojemnością jest **jednostka transferu** (ang. *transfer unit*). W przypadku pamięci wewnętrznej jednostka transferu jest równa liczbie linii danych doprowadzonych do modułu pamięci i wychodzących z niego. Jest ona często równa długości słowa, ale nie musi tak być. W celu wyjaśnienia tego problemu rozważmy trzy powiązane ze sobą pojęcia dotyczące pamięci wewnętrznej:

- **Słowo.** „Naturalna” jednostka organizacji pamięci. Zwykle rozmiar słowa jest równy liczbie bitów wykorzystywanych do reprezentowania liczby lub długości rozkazu. Niestety, jest wiele wyjątków. Na przykład CRAY-1 wykorzystuje słowa 64-bitowe, jednak liczba całkowita jest reprezentowana za pomocą 24 bitów. VAX ma zdumiewająco różne długości rozkazów, wyrażone jako wielokrotności bajtów, a rozmiar słowa wynosi 32 bity.
- a **Jednostka adresowalna.** W wielu systemach jednostką adresowalną jest słowo. Jednak niektóre systemy umożliwiają adresowanie na poziomie bajtów. W każdym przypadku zależność między długością adresu A a liczbą adresowalnych jednostek N jest następująca: $2^A = N$.
- **Jednostka transferu.** W przypadku pamięci głównej jest to liczba bitów jednocześnie odczytywanych z pamięci lub do niej zapisywanych. Jednostka transferu nie musi być równa słowu lub jednostce adresowalnej. W przypadku pamięci zewnętrznej dane są często przekazywane w jednostkach o wiele większych niż słowo, określanych jako **bloki**.

Jedną z najbardziej widocznych różnic między różnymi rodzajami pamięci dotyczy *sposobu dostępu* do jednostek danych. Można wyróżnić cztery rodzaje dostępu:

- a **Dostęp sekwencyjny.** Pamięć jest zorganizowana za pomocą jednostek danych zwanych rekordami. Dostęp jest możliwy w określonej sekwencji liniowej. Do oddzielania rekordów i do pomocy przy odczycie są wykorzystywane przechowywane informacje adresowe. Odczyt i zapis są wykonywane za pomocą tego samego mechanizmu, przy czym proces ten musi się przemieszczać z pozycji bieżącej do pozycji pożądaney, przepuszczając i odrzucając każdy rekord pośredni. W rezultacie czas dostępu do różnych rekordów może się bardzo różnić. Pamięci taśmowe, omawiane w rozdz. 5, charakteryzują się właśnie dostępem sekwencyjnym.
- a **Dostęp bezpośredni.** Podobnie jak w przypadku dostępu sekwencyjnego, proces odczytu i zapisu w pamięciach o dostępie bezpośrednim jest realizowany za pomocą tego samego mechanizmu. Jednak poszczególne bloki lub rekordy mają unikatowy adres oparty na lokalizacji fizycznej. Dostęp jest realizowany przez bezpośredni dostęp do najbliższego otoczenia, po którym następuje sekwencyjne poszukiwanie, liczenie lub oczekiwanie w celu osiągnięcia lokalizacji finalnej. Jak poprzednio czas dostępu jest zmienny. Pamięci dyskowe, które omówimy w rozdz. 5, charakteryzują się dostępem bezpośrednim.
- **Dostęp swobodny.** Każda adresowalna lokalizacja w pamięci ma unikatowy, fizycznie wbudowany mechanizm adresowania. Czas dostępu do danej lokalizacji nie zależy od sekwencji poprzednich operacji dostępu i jest stały. Dzięki temu dowolna lokalizacja może być wybierana swobodnie i jest adresowana i dostępna bezpośrednio. Systemy pamięci głównej wyróżniają się dostępem swobodnym.
- a **Dostęp skojarzeniowy.** Jest to rodzaj dostępu swobodnego, który umożliwia porównywanie i specyficzne badanie zgodności wybranych bitów wewnątrz słowa, przy czym jest to czynione dla wszystkich słów jednocześnie. Tak więc słowo jest wyprowadzane raczej na podstawie części swojej wartości niż na podstawie adresu. Podobnie jak w przypadku zwykłych pamięci o dostępie swobodnym, każda lokalizacja ma własny mechanizm adresowania, a czas dostępu jest stały i niezależny od poprzednich operacji dostępu. W pamięciach podręcznych, które omówimy w podrozdz. 4.3, może być używany dostęp skojarzeniowy.

Z punktu widzenia użytkownika dwiema najważniejszymi własnościami pamięci są pojemność i **wydajność**. Wykorzystywane są trzy parametry będące miarą wydajności:

- a **Czas dostępu.** W przypadku pamięci o dostępie swobodnym jest to czas niezbędny do zrealizowania operacji odczytu lub zapisu, to *znaczy czas* od chwili doprowadzenia adresu do chwili zmagazyinowania lub udostępnienia

danych. W przypadku pamięci o dostępie nieswobodnym czas dostępu jest czasem potrzebnym na umieszczenie mechanizmu odczytu-zapisu w pożądanym miejscu.

- a **Czas cyklu pamięci.** Pojęcie to było pierwotnie stosowane do pamięci o dostępie swobodnym. Czas cyklu składa się z czasu dostępu oraz z dodatkowego czasu, który musi upłynąć, zanim będzie mógł nastąpić kolejny dostęp. Ten dodatkowy czas może być potrzebny dla zaniku sygnałów przejściowych lub do regeneracji danych, jeśli odczyt jest niszczący.
- a **Szybkość przesyłania (transferu).** Jest to szybkość, z jaką dane mogą być wprowadzane do jednostki pamięci lub z niej wyprowadzane. W przypadku pamięci o dostępie swobodnym jest ona równa $1/(\text{czas cyklu})$. W przypadku pamięci o dostępie nieswobodnym zachodzi następująca zależność:

$$T_N = T_A + \frac{N}{R}$$

gdzie:

T_N - średni czas odczytu lub zapisu N bitów,

T_A - średni czas dostępu,

N - liczba bitów,

R - szybkość transferu w bitach na sekundę [bit/s].

Wykorzystywane są różne *fizyczne* rodzaje pamięci. Dwa najbardziej powszechne spośród nich obecnie to pamięć półprzewodnikowa, wykonana w technologii LSI lub VLSI, oraz powierzchniowa pamięć magnetyczna w postaci dysków i taśm.

Ważne są niektóre *własności fizyczne* urządzeń do przechowywania danych. W pamięci ulotnej informacja zanika naturalnie lub jest tracona po wyłączeniu zasilania. W pamięci nieulotnej informacja raz zapisana nie zmienia się aż do chwili zamierzonej zmiany; zasilanie elektryczne nie jest niezbędne do zachowania informacji. Powierzchniowe pamięci magnetyczne są nieulotne. Pamięć półprzewodnikowa może być albo ulotna, albo nieulotna. Zawartość pamięci niewymazywalnej nie może być zmieniana, z wyjątkiem wypadku zniszczenia jednostki pamięciowej. Pamięć półprzewodnikowa tego typu jest znana jako **pamięć stała** (ang. *read-only memory* - ROM). Z konieczności pamięć niewymazywalna musi być także nieulotna.

W przypadku pamięci o dostępie swobodnym podstawowym problemem przy projektowaniu jest organizacja. Przez **organizację** rozumiemy tutaj fizyczne uporządkowanie bitów w celu uformowania słów. Jak wyjaśnimy, nie zawsze wykorzystywane jest uporządkowanie, które wydaje się oczywiste.

Hierarchia pamięci

Ograniczenia przy projektowaniu pamięci komputera mogą być podsumowane za pomocą trzech pytań: Ile? Jak szybko? Za ile?

Pytanie „Ile?” jest wciąż w pewnym stopniu otwarte. Jeśli dysponujemy określoną pojemnością, to z pewnością pojawi się zastosowanie, w którym po-

jemność ta zostanie wykorzystana. Odpowiedź na pytanie „Jak szybko?” jest w pewnym sensie łatwiejsza. Aby osiągnąć największą wydajność, pamięć musi być w stanie nadążyć za procesorem. Znaczy to, że nie chcielibyśmy, aby procesor, wykonując rozkazy, czekał na rozkazy lub argumenty. Ostatnie z pytań również musi być rozważone. W przypadku praktycznego systemu koszt pamięci musi pozostawać w rozsądnej relacji do kosztu pozostałych składników.

Jak można było się spodziewać, istnieją wzajemne zależności między podstawowymi parametrami pamięci, to znaczy między kosztem, pojemnością i czasem dostępu. Jak zawsze przy konstruowaniu systemów pamięciowych są wykorzystywane różne technologie. Jeśli rozpatrujemy dostępne technologie, to zauważamy następujące zależności:

- mniejszy czas dostępu - większy koszt na bit,
- a większa pojemność - mniejszy koszt na bit,
- a większa pojemność - większy czas dostępu.

Dylemat, przed którym stoi projektant, jest jasny. Chciałby on wykorzystać technologie pamięci, które umożliwiają wytworzenie pamięci o dużej pojemności, ponieważ po pierwsze są one potrzebne, a po drugie z powodu mniejszego kosztu na bit. Jednak aby osiągnąć wymaganą wydajność, projektant wykorzystuje drogie i stosunkowo małej pojemności pamięci o krótkim czasie dostępu.

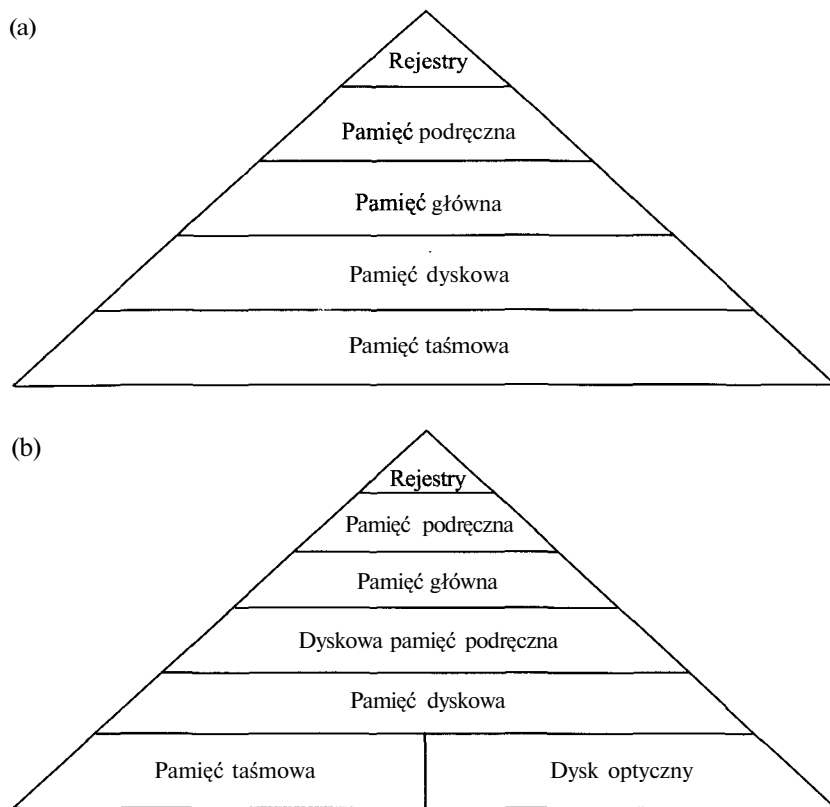
Osiągnięcie kompromisu nie polega na wyborze pojedynczego zespołu pamięci czy też określonej technologii, lecz na wykorzystaniu *hierarchii pamięci*. Typowa hierarchia jest przedstawiona na rys. 4.1a. Rozpatrując tę hierarchię w kierunku od góry do dołu, obserwujemy następujące zjawiska:

- (a) malejący koszt na bit,
- (b) rosnąca pojemność,
- (c) rosnący czas dostępu,
- (d) malejącą częstotliwość dostępu do pamięci przez procesor.

Tak więc mniejsze, droższe i szybsze pamięci są uzupełniane przez pamięci większe, tańsze i powolne. Kluczem do sukcesu przy projektowaniu organizacji jest ostatnie z wymienionych zjawisk, a mianowicie zmniejszanie częstotliwości dostępu. Przedyskutujemy tę koncepcję szczegółowo podczas omawiania pamięci podręcznej w dalszej części tego rozdziału oraz pamięci wirtualnej w rozdziale 7, tutaj jednak przedstawimy krótkie wyjaśnienie.

Jeśliby pamięć mogła być zorganizowana przy uwzględnieniu zjawisk (a), (b) oraz (c) i jeśli dane i rozkazy byłyby rozmieszczone w pamięci przy uwzględnieniu (d), to sądzimy intuicyjnie, że taki schemat zredukowałby łączne koszty, pozostawiając określony poziom wydajności. Podamy prosty przykład ilustrujący to stwierdzenie.

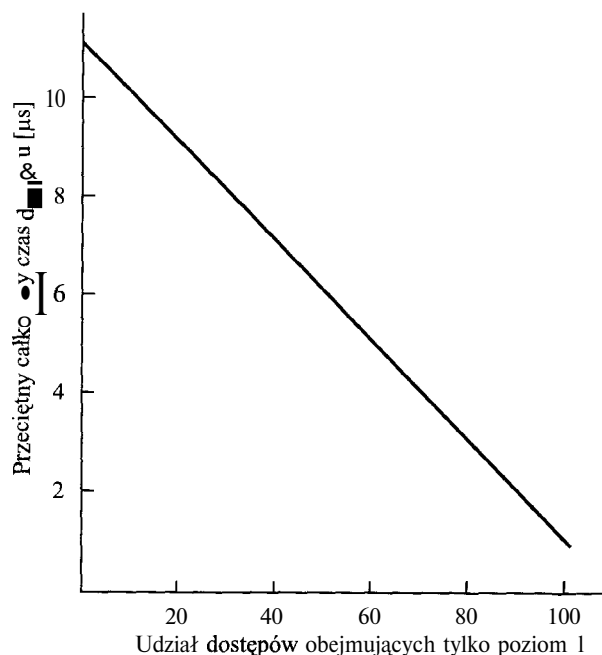
Założmy, że procesor ma dostęp do dwóch poziomów pamięci. Poziom pierwszy zawiera 1000 słów i ma czas dostępu 1 μ s. Poziom 2 zawiera 100 000 słów i ma czas dostępu 10 μ s. Założmy dalej, że jeśli poszukiwane słowo znaj-



RYСУNEK 4.1. Hierarchia pamięci: (a) tradycyjna hierarchia pamięci; (b) współczesna hierarchia pamięci

duje się na poziomie 1, to procesor ma do niego dostęp bezpośredni. Jeśli natomiast jest ono na poziomie 2, to słowo jest najpierw przenoszone na poziom 1, a następnie udostępniane procesorowi. Dla uproszczenia zaniedbajmy czas wymagany do tego, aby procesor określił, czy słowo znajduje się na pierwszym, czy też na drugim poziomie. Na rysunku 4.2 jest pokazany średni całkowity czas dostępu jako funkcja procentu czasu, w ciągu którego słowo jest znajdowane od razu na poziomie 1. Jak można zauważyć, przy wysokim udziale bezpośrednich dostępu do poziomu 1, średni całkowity czas dostępu jest o wiele bliższy temu czasowi charakterystycznemu dla poziomu 1 niż dla 2.

Przykład ten pokazuje, że strategia ta w zasadzie funkcjonuje. Będzie działała w praktyce, jeśli będą spełnione warunki od (a) do (d). Na rysunkach 4.3 i 4.4 są pokazane typowe własności współczesnych alternatywnych systemów pamięciowych. Jak widać na rys. 4.3, wykorzystując różne technologie, otrzymamy zbiór systemów pamięci, który satysfakcjonuje warunki (b) i (c), zaś rys. 4.4 potwierdza spełnienie warunku (a). Na szczęście warunek (d) jest również ogólnie słuszny.

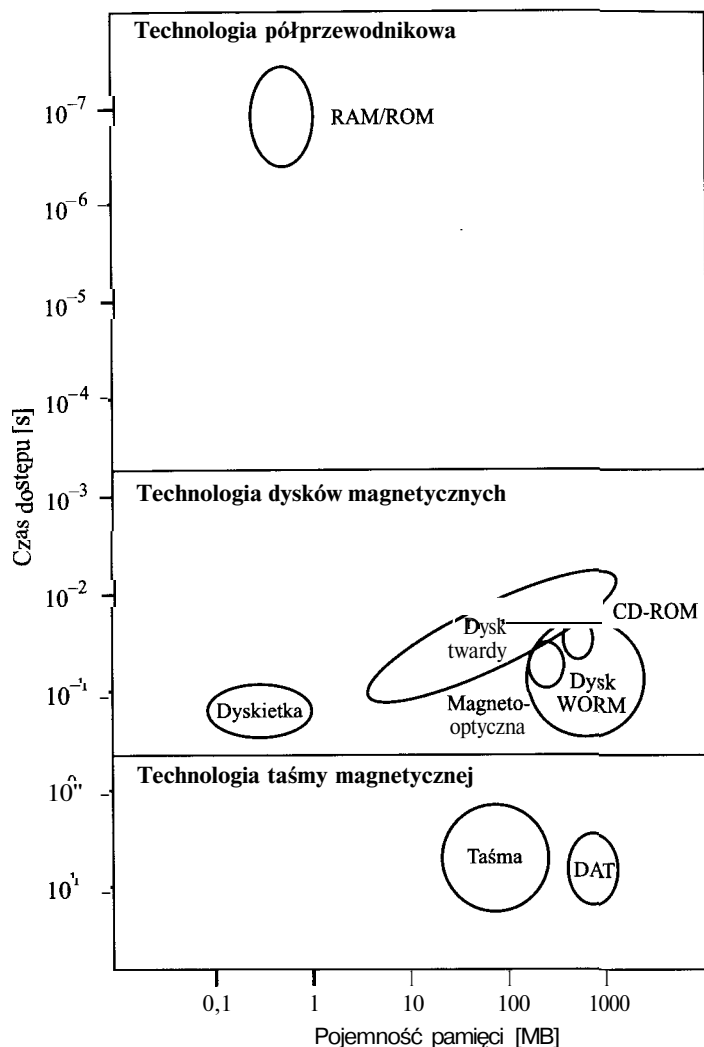


RYSUNEK 4.2. Własności prostej pamięci dwupoziomowej

Podstawą słuszności warunku (d) jest zasada **lokalności odniesień** (*locality of reference*) [DENN68]. Podczas wykonywania programu odniesienia do pamięci (zarówno dotyczące danych, jak i rozkazów) mają tendencję do gromadzenia się. Programy typowo zawierają pewną liczbę pętli iteracyjnych i podprogramów. Gdy jest uruchomiona pętla lub podprogram, występują powtarzające się odniesienia do niewielkiego zestawu rozkazów. Podobnie, operacje na tablicach lub ciągach danych polegają na dostępie do zgrupowanego zespołu słów. W ciągu długiego czasu wykorzystywane ugrupowania zmieniają się, jednak w krótkim czasie procesor pracuje z ustalonymi grupami odniesień do pamięci.

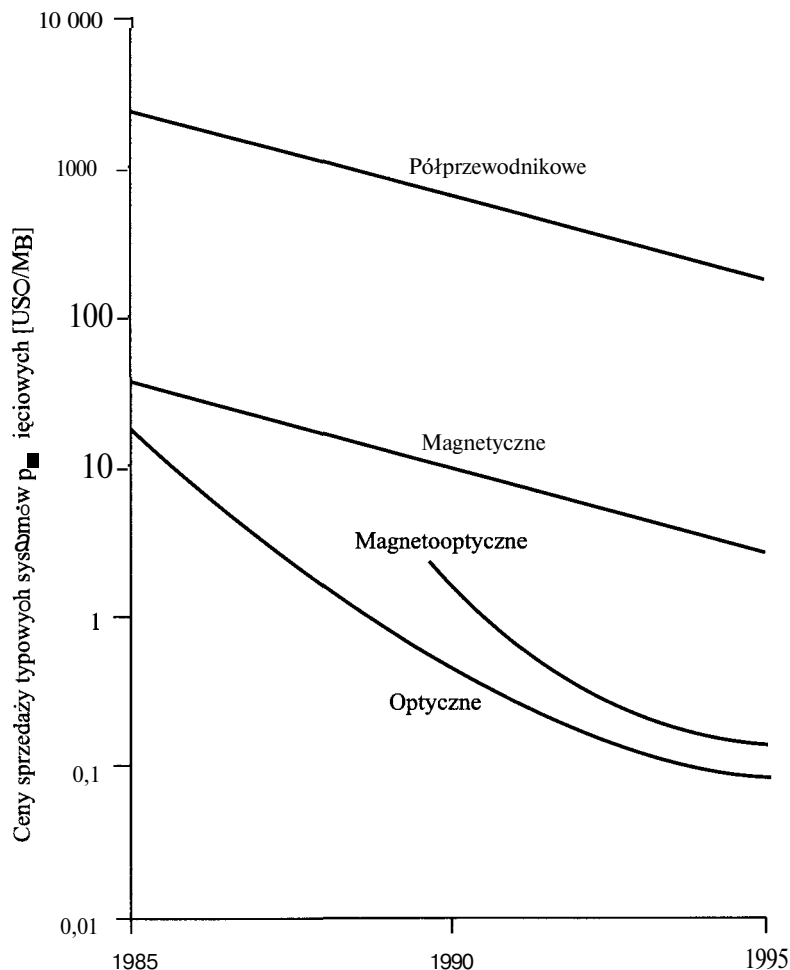
W tej sytuacji jest możliwa taka organizacja układu danych w hierarchii pamięci, żeby udział dostępu do każdego kolejnego niższego poziomu był istotnie mniejszy niż do poziomu wyższego. Rozpatrzmy dwupoziomowy przykład przedstawiony powyżej. Przyjmijmy, że poziom 2 pamięci zawiera wszystkie rozkazy i dane programu. Bieżące ugrupowania danych mogą być czasowo umieszczane na poziomie 1. Od czasu do czasu niektóre ugrupowania danych będą musiały być przenoszone na poziom 2, żeby zwolnić miejsce dla nowych zbiorów wprowadzanych do poziomu 1. Przeciętnie jednak większość odniesień będzie dotyczyła rozkazów i danych zawartych na poziomie 1.

Zasada ta może być stosowana do systemów zawierających więcej niż dwa poziomy. Rozważmy hierarchię pokazaną na rys. 4.1a. Najszybszy, najmniejszy



RYSUNEK 4.3. Porównanie pamięci

i najdroższy rodzaj pamięci obejmuje rejestry zawarte wewnątrz procesora. Typowo procesor zawiera kilka tuzinów takich rejestrów, chociaż niektóre maszyny mają ich setki. Dwa poziomy niżej znajduje się pamięć główna, będąca zasadniczym wewnętrznym systemem pamięciowym komputera. Każda lokacja w pamięci głównej ma swój unikatowy adres, a większość rozkazów maszynowych odnosi się do jednego lub wielu adresów w pamięci głównej. Pamięć główna jest zwykle poszerzana za pomocą szybszej i mniejszej pamięci podręcznej. Ta ostatnia nie jest zwykle widzialna dla programisty czy też, w istocie, dla procesora. Urządzenie to stopniuje przepływ danych między pamięcią główną a rejestrami procesora i ma na celu poprawienie wydajności.



RYSUNEK 4.4. Przewidywany koszt pamięci pomocniczych

Trzy opisane wyżej rodzaje pamięci są zwykle ulotne i są wykonane w technologii półprzewodnikowej. Zastosowanie układu trójpoziomowego pozwala na wykorzystanie dostępnych obecnie typów pamięci półprzewodnikowych różniących się szybkością i kosztem. Dane są przechowywane w sposób trwalszy w zewnętrznych, masowych urządzeniach pamięciowych, z których najpowszechniejsze są pamięci dyskowe i taśmowe. Zewnętrzne pamięci nieulotne są określane jako pamięci wtórne lub pomocnicze. Są używane do przechowywania programów i plików danych; programista widzi je tylko poprzez pliki i rekordy, a nie przez pojedyncze bajty czy słowa. Dyski są także wykorzystywane do rozszerzania pamięci głównej, jako pamięci wirtualne, którymi zajmujemy się w rozdz. 7.

W hierarchii mogą występować również inne rodzaje pamięci. Na przykład duże komputery IBM zawierają pewną postać pamięci wewnętrznej znaną jako

pamięć rozszerzona (*expanded storage*). Wykorzystuje się w niej wolniejszą i tańszą technologię półprzewodnikową niż w pamięci głównej. Ściśle mówiąc, pamięć ta nie tworzy hierarchii, lecz stanowi rodzaj bocznej gałęzi: dane mogą być przenoszone między pamięcią główną a pamięcią rozszerzoną, jednak nie między pamięcią rozszerzoną a pamięcią zewnętrzną. Do innych postaci pamięci wtórnych należą dyski optyczne i magnetyczne pamięci bąbelkowe (ang. *bubble memory devices*). Wreszcie, dodatkowe poziomy mogą być skutecznie dodane do hierarchii za pomocą oprogramowania. Część pamięci głównej może być wykorzystywana jako bufor do tymczasowego przechowywania danych, które następnie mają być odczytane i wyprowadzone na dysk. Taka technika, określana czasem jako dyskowa pamięć podręczna¹, poprawia wydajność na dwa sposoby:

- a Zapisy dyskowe są zgrupowane. W miejsce wielu transferów niewielkich ilości danych mamy do czynienia z niewieloma wielkimi transferami. Poprawia to wydajność dysku i minimalizuje zaangażowanie procesora.
- a Pewne dane przeznaczone do odczytania mogą być określone przez program jeszcze przed następnym sięgnięciem do dysku. W tym przypadku dane te mogą być wydobyte szybko z programowej pamięci podręcznej, a nie powoli z dysku.

Na rysunku 4.1b widać współczesną hierarchię pamięci, uwzględniającą dyskową pamięć podręczną i dysk optyczny jako dodatkowe rodzaje pamięci wtórnej.

W dodatku 4A jest przedyskutowany wpływ wielopoziomowych struktur pamięciowych na wydajność.

4.2

PÓŁPRZEWODNIKOWA PAMIĘĆ GŁÓWNA

We wcześniejszych komputerach najpowszechniejszą formą komputerowych pamięci głównych o swobodnym dostępie były zespoły pierścieni ferromagnetycznych nazywanych **rdzeniami**. Dlatego też pamięci główne tego rodzaju były nazywane **rdzeniowymi**. Termin ten funkcjonuje do dziś. Dość długo trwało, zanim nadejście mikroelektroniki i jej zalety spowodowały ustąpienie rdzeniowych pamięci magnetycznych. Dzisiaj zastosowanie mikroukładów półprzewodnikowych w pamięciach głównych jest prawie uniwersalne. Głównymi aspektami tej technologii zajmiamy się w następnym punkcie.

¹ Dyskowa pamięć podręczna jest na ogół rozwiązaniem czysto programowym i nie jest analizowana w tej książce. Omówienie tej pamięci można znaleźć w [STAL95].

Rodzaje pamięci półprzewodnikowych o dostępie swobodnym

Wszystkie rodzaje pamięci rozpatrywane w tym punkcie charakteryzują się dostępem swobodnym. Znaczy to, że pojedyncze słowa w pamięci są dostępne bezpośrednio za pomocą wbudowanych układów logicznych adresowania.

W tabeli 4.2 są wymienione główne rodzaje pamięci półprzewodnikowych. Najpowszechniejsza jest określana jako **pamięć o dostępie swobodnym (RAM)**. Oczywiście jest to niewłaściwe wykorzystanie tego określenia, ponieważ wszystkie rodzaje pamięci wymienione w tablicy są pamięciami o dostępie swobodnym. Cechą wyróżniającą pamięci RAM jest to, że możliwe jest zarówno odczytanie danych z pamięci, jak też łatwe i szybkie zapisanie do niej nowych danych. Zarówno odczyt, jak i zapis są dokonywane za pomocą sygnałów elektrycznych.

TABELA 4.2. Rodzaje pamięci półprzewodnikowych

Rodzaj pamięci	Kategoria	Wymazywanie	Sposób zapisu	Ulotność
Pamięć o dostępie swobodnym (RAM)	odczyt-zapis	elektryczne, na poziomie bajta	elektryczny	ulotna
Pamięć stała (ROM)	tylko odczyt	niemożliwe	maski	nieulotne
Programowalna pamięć stała (PROM)			elektryczny	
Wymazywalna PROM (EPROM)	głównie odczyt	światłem UV, na poziomie mikroukładu		
Pamięć błyskawiczna		elektryczne, na poziomie bloku		
Elektrycznie wymazywalna PROM (EEPROM)		elektryczne, na poziomie bajta		

Inną własnością wyróżniającą pamięci RAM jest ich ulotność. Pamięć RAM musi mieć źródło stałego zasilania. Jeśli zasilanie jest przerwane, dane są traczone. Pamięć RAM może więc być używana tylko do przechowywania czasowego.

Pamięci RAM można podzielić na statyczne i dynamiczne. **Dynamiczna** pamięć RAM jest wykonana z komórek, które przechowują dane podobnie, jak kondensatory przechowują ładunek elektryczny. Obecność lub brak ładunku w kondensatorze są interpretowane jako binarne 1 lub 0. Ponieważ kondensatory mają naturalną tendencję do rozładowywania się, dynamiczne pamięci RAM wymagają okresowego odświeżania ładunku w celu zachowania danych. W **statycznych** pamięciach RAM wartości binarne są przechowywane za pomocą przerzutnikowych konfiguracji bramek logicznych (patrz dodatek 4A zawierający opisy przerzutników). Statyczne pamięci RAM zachowują dane tak długo, jak długo są zasilane.

Zarówno statyczne, jak i dynamiczne pamięci RAM są ulotne. Dynamiczna komórka pamięciowa jest prostsza i dzięki temu mniejsza niż statyczna. W rezultacie dynamiczna pamięć RAM jest gęściej upakowana (mniejsze komórki = więcej komórek na jednostkę powierzchni) i tańsza niż odpowiadająca jej statyczna pamięć RAM. Z drugiej strony, dynamiczna pamięć RAM wymaga układów odświeżania. W przypadku dużych pamięci stały koszt układów odświeżania jest z nawiązką skompensowany przez mniejszy, zmienny koszt dynamicznych komórek pamięciowych. W rezultacie istnieje tendencja do faworyzowania dynamicznych pamięci RAM w dużych pamięciach. Zauważmy na koniec, że statyczne pamięci RAM są nieco szybsze niż dynamiczne.

Silny kontrast w stosunku do pamięci RAM stanowią **pamięci stałe** (*read-only memory* - ROM). Jak sugeruje ich nazwa, pamięci ROM zawierają trwały wzór danych, który nie może być zmieniony. Podczas gdy jest możliwe odczytywanie pamięci ROM, nie można zapisać w nich nowych danych. Ważnym zastosowaniem pamięci ROM jest mikroprogramowanie, którym zajmujemy się w części IV. Do innych potencjalnych zastosowań należą:

- a podprogramy biblioteczne dla często używanych funkcji,
- a programy systemowe,
- a tablice funkcji.

W przypadku umiarkowanych wymagań zaletą pamięci ROM jest to, że dane lub program pozostają na stałe w pamięci głównej i nigdy nie wymagają ładowania z urządzeń pamięci wtórnej.

Pamięć ROM powstaje podobnie, jak wszystkie mikroukłady scalone, przy czym dane są wbudowywane podczas procesu wytwarzania. Wynikają z tego dwa problemy.

- Q Umieszczaniu danych w pamięci towarzyszy stosunkowo duży koszt stały, niezależny od tego, czy wytwarza się 1000 kopii określonego układu, czy tylko jedną.
- a Nie może wystąpić błąd. Jeśli jeden bit jest niewłaściwy, cała partia układów ROM musi być wyrzucona.

Jeśli potrzebna jest tylko niewielka pamięć ROM o określonej zawartości, tańszym rozwiązaniem jest wykorzystanie **programowalnej pamięci ROM** (PROM). Podobnie jak ROM pamięć PROM jest nieulotna i można ją zapisać tylko raz. W przypadku pamięci PROM zapis jest realizowany elektrycznie i może być wykonany przez dostawcę lub przez klienta już po wyprodukowaniu mikroukładu. Do zapisu (lub „programowania”) wymagane są specjalne urządzenia. Pamięci PROM zapewniają elastyczność i wygodę. Pamięci ROM natomiast pozostają atrakcyjne w przypadku dużych ilości produkowanego sprzętu.

Inną odmianą pamięci stałych jest **pamięć głównie do odczytu** (*read-mostly memory*), która jest użyteczna wtedy, gdy operacje zapisu są znacznie częstsze

niż operacje zapisu, natomiast wymagana jest nieulotność. Powszechnie znane są trzy odmiany pamięci głównie do odczytu: EPROM, EEPROM i tzw. pamięć błyskawiczna (*flashmemory*).

Optycznie wymazywalna programowalna pamięć stała (EPROM) jest odczytywana i zapisywana elektrycznie, podobnie jak PROM. Jednak przed operacją zapisu wszystkie komórki pamięci muszą być wymazane przez naświetlenie znajdującego się już w obudowie układu promieniowaniem ultrafioletowym. Proces wymazywania może być wykonywany wielokrotnie; każde wymazanie trwa około 20 minut. Zawartość pamięci EPROM może więc być zmieniana wiele razy, poza tym pamięć ta przechowuje dane permanentnie, jak ROM i PROM. Przy porównywalnej pojemności pamięć EPROM jest droższa niż PROM, jednak ma zaletę możliwości wielokrotnej aktualizacji zawartości.

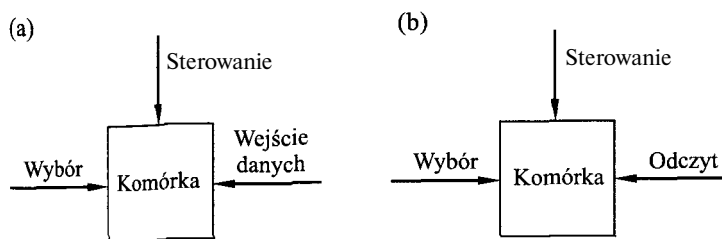
Bardziej atrakcyjną odmianą pamięci głównie do odczytu jest **elektrycznie wymazywalna programowalna pamięć stała** (EEPROM). Jest to pamięć głównie do odczytu, która może być zapisana bez wymazywania poprzedniej zawartości; aktualizowany jest tylko bajt (lub bajty) adresowane. Operacja zapisu trwa znacznie dłużej niż odczytu i zajmuje czas rzędu kilkuset mikrosekund na bajt. EEPROM łączy zaletę nieulotności z możliwością aktualizacji na miejscu, przy wykorzystaniu zwykłych magistralowych linii sterowania, adresów i danych. EEPROM jest droższa niż EPROM, a także mniej gęsto upakowana (zawiera mniej bitów w mikroukładzie).

Najnowszą postacią pamięci półprzewodnikowej jest **pamięć błyskawiczna** (nazwana tak z powodu szybkości, z jaką może być reprogramowana). Wprowadzono ją po raz pierwszy w połowie lat osiemdziesiątych. Zajmuje miejsce pośrednie między EPROM i EEPROM zarówno pod względem kosztu, jak i funkcjonalności. Podobnie jak w EEPROM w pamięci błyskawicznej wykorzystuje się metodę wymazywania elektrycznego. Cała pamięć błyskawiczna może być wymazana w ciągu kilku sekund, a więc o wiele szybciej niż EPROM. Ponadto możliwe jest wymazywanie bloków pamięci zamiast całego mikroukładu. Jednak pamięć błyskawiczna nie umożliwia wymazywania na poziomie bajtów. Podobnie jak w EPROM w pamięci błyskawicznej wykorzystuje się tylko jeden tranzystor na bit, osiągając przez to wysoką gęstość upakowania, podobną do EPROM.

Organizacja

Podstawowym elementem pamięci półprzewodnikowej jest komórka pamięci. Chociaż są wykorzystywane różne technologie, wszystkie komórki pamięci półprzewodnikowych mają pewne wspólne cechy:

- ❑ Mają dwa stabilne (lub półstabilne) stany, które mogą być użyte do reprezentowania binarnych 1 i 0.
- ❑ Umożliwiają zapis (przynajmniej jednokrotny).
- a Umożliwiają odczyt.



RYСУNEK 4.5. Działanie komórki pamięci: (a) zapis; (b) odczyt

Na rysunku 4.5, w sposób schematyczny, pokazano działanie komórki pamięci. Najczęściej komórka ma trzy końcówki funkcyjne służące do przenoszenia sygnału elektrycznego. Końcówka wyboru, jak sama nazwa wskazuje, służy do wybierania komórki pamięci w celu przeprowadzenia operacji odczytu lub zapisu. Końcówka sterowania umożliwia wskazanie rodzaju operacji (zapis lub odczyt). W przypadku zapisu przez inną końcówkę wprowadzany jest sygnał elektryczny, który ustala stan komórki na 1 lub 0. W przypadku odczytu ta sama końcówka służy do wyprowadzania sygnału o stanie komórki. Szczegóły organizacji wewnętrznej, funkcjonowania i przebiegów czasowych dotyczące komórki pamięciowej zależą od wykorzystanej technologii układów scalonych i wykraczają poza zakres tej książki. Dla naszych celów przyjmijmy jako dane to, że indywidualne komórki mogą być wybierane w celu przeprowadzenia operacji odczytu i zapisu.

Układy logiczne mikroukładu

Podobnie jak inne układy scalone, pamięć półprzewodnikowa jest w postaci obudowanego mikroukładu (rys. 2.7). Każdy mikroukład zawiera zestaw komórek pamięci. W przypadku aktualnie dostępnej technologii, powszechne są mikroukłady o pojemności 4 Mbit, natomiast wprowadzane są do użytku mikroukłady o pojemności 16 Mbit.

Jak widzieliśmy, w hierarchicznym układzie pamięci występują współzależności między szybkością, pojemnością a kosztem. Takie współzależności istnieją również, jeśli rozpatrujemy organizację komórek pamięci i funkcjonalne układy logiczne zawarte w mikroukładzie. W przypadku pamięci półprzewodnikowych jednym z kluczowych problemów projektowych jest liczba bitów danych, które mogą być jednocześnie odczytywane lub zapisywane. Na jednym biegunie znajduje się organizacja, w której fizyczne uporządkowanie komórek w układzie jest takie samo, jak uporządkowanie logiczne (postrzegane przez procesor) słów w pamięci. Komórki są wtedy zorganizowane w postaci *W* słów *S*-bitowych. Na przykład mikroukład 16 Mbit może być zorganizowany jako 1 M (milion) słów 16-bitowych. Na drugim biegunie znajduje się organizacja 1-bitowa, w przypadku której w określonym czasie odczytuje się lub zapisuje

tylko jeden bit. Przedstawimy organizację mikroukładów pamięciowych na przykładzie pamięci DRAM; organizacja pamięci ROM jest podobna, choć prostsza.

Na rysunku 4.6 jest pokazana typowa organizacja 16-megabitowej pamięci DRAM. W tym przypadku jednocześnie mogą być odczytywane lub zapisywane 4 bity. Logicznie rzecz biorąc, zespół pamięci jest zorganizowany w postaci 4 kwadratowych układów 2048 na 2048 elementów. Możliwe są różne organizacje fizyczne. W każdym przypadku elementy zespołu są połączone zarówno przez linie poziome (wiersze), jak i pionowe (kolumny). Każda linia pozioma jest połączona z końcówkami „wybór” każdej komórki w wierszu; każda linia pionowa jest połączona z końcówkami „zapis/odczyt” każdej komórki w kolumnie.

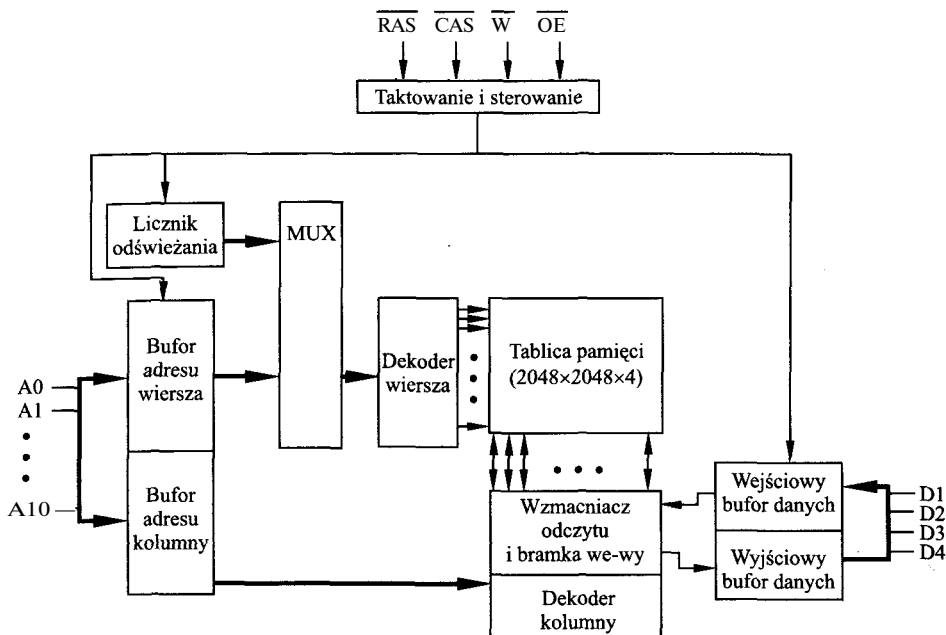
Linie adresowe dostarczają adres słowa, które ma być wybrane. Łącznie potrzeba $\log_2 W$ linii. W naszym przykładzie wymaganych jest 11 linii w celu wybrania 1 z 2048 wierszy. Linie te są doprowadzone do dekodera wiersza mającego 11 linii wejściowych i 2048 wyjściowych. Układ logiczny dekodera aktywuje jedną z 2048 linii zależnie od wzoru bitowego na 11 liniach wejściowych ($2^{11} = 2048$).

Dodatkowe 11 linii adresu wybiera 1 z 2048 kolumn, przy czym na 1 kolumnę przypadają 4 bity. W celu wprowadzania i wyprowadzania 4 bitów z bufora danych są wykorzystywane 4 linie danych. Na wejściu (zapis) sterownik bitowy każdej linii bitu jest aktywowany do 1 lub 0, zależnie od wartości odpowiedniej linii danych. Na wyjściu (odczyt) sygnał z każdej linii bitowej jest doprowadzany poprzez wzmacniacz odczytu do linii danych. Za pomocą linii wiersza dokonuje się wyboru, który wiersz komórek jest wykorzystywany przy odczycie lub zapisie.

Ponieważ do tej pamięci DRAM są zapisywane (lub odczytywane) jednocześnie tylko 4 bity, wiele takich pamięci musi być połączonych ze sterownikiem pamięci w celu umożliwienia odczytu lub zapisu słowa i przekazania go do magistrali.

Zauważmy, że występuje tylko 11 linii adresu ($A_0 \div A_{10}$), połowa liczby, której moglibyśmy się spodziewać dla zespołu 2048 x 2048. Czyni się tak, aby zmniejszyć liczbę doprowadzeń do układu. Wymagane linie adresu (22) przechodzą przez układy logiczne wyboru zewnętrzne w stosunku do mikroukładu, po czym są multipleksowane do 11 linii adresu. Najpierw do mikroukładu doprowadza się 11 sygnałów adresowych określających adres wiersza w zespole, po czym 11 pozostałych sygnałów adresowych określa adres kolumny. Sygnałom tym towarzyszą sygnały: wyboru adresu wiersza (RAS) i wyboru adresu kolumny (CAS) umożliwiające sterowanie czasowe mikroukładu.

Niezależnie od tego, adresowanie multipleksowane w połączeniu z kwadratową organizacją zespołów komórek umożliwiało 4-krotne zwiększenie pojemności pamięci w każdej nowej generacji mikroukładów pamięciowych. Jedno dodatkowe doprowadzenie przeznaczone do adresowania podwaja liczbę wier-



RYСУNEK 4.6. Typowa pamięć DRAM o pojemności 16 Mbit (4 M x 4)

szy i kolumn, przez co rozmiar mikroukładu pamięciowego wzrasta 4-krotnie. Dotychczas mieliśmy do czynienia z następującymi generacjami, występującymi mniej więcej co trzy lata: 1 K, 4 K, 16 K, 64 K, 256 K, 1 M, 4 M, 16 M.

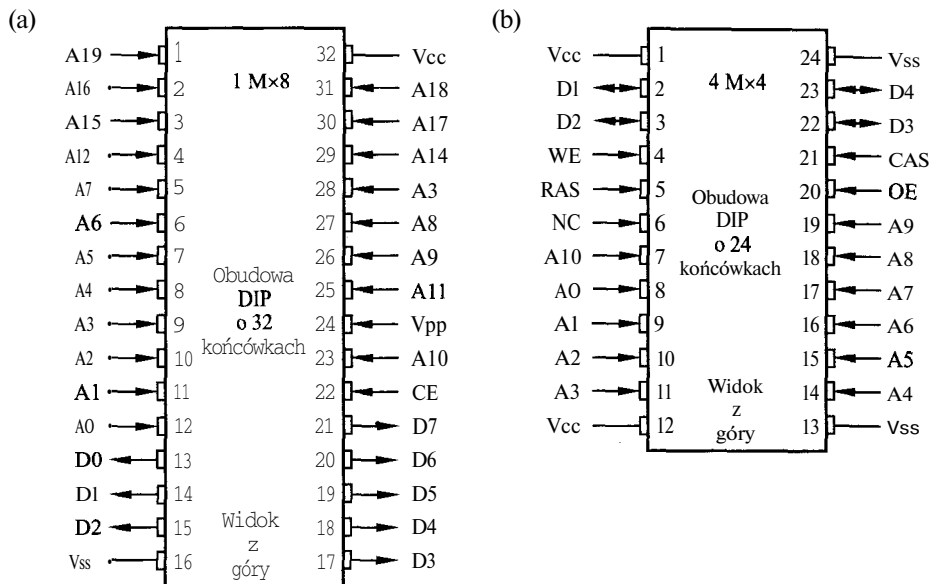
Na rysunku 4.6 są również pokazane układy odświeżania. Wszystkie pamięci DRAM wymagają operacji odświeżania. Prostim sposobem odświeżania jest uniemożliwienie używania mikroukładu DRAM w czasie, gdy wszystkie komórki danych są odświeżane. Licznik odświeżania przechodzi kolejno przez wszystkie wiersze. W przypadku każdego wiersza linie wyjściowe licznika odświeżania są łączone z dekodern wiersza i wzbudzana jest linia $\overline{\text{RAS}}$. Powoduje to odświeżenie każdej komórki w wierszu.

Montaż mikroukładów w obudowie

Jak stwierdziliśmy w rozdz. 2, układ scalony jest montowany w obudowie, która zawiera końcówki służące do łączenia ze światem zewnętrznym.

Na rysunku 4.7a jest pokazana przykładowa obudowa pamięci EPROM o pojemności 8 Mbit, zorganizowanej jako 1 M x 8. W tym przypadku organizacja ta może być traktowana jako oparta na słowach. Obudowa ma 32 końcówki, co jest jednym z rozwiązań standardowych. Końcówkom są przypisywane następujące linie sygnałowe:

- adres słowa; w przypadku 1 M słów wymagane jest 20 ($2^{20} = 1 \text{ M}$) końcówek ($\text{A0} \div \text{A19}$);



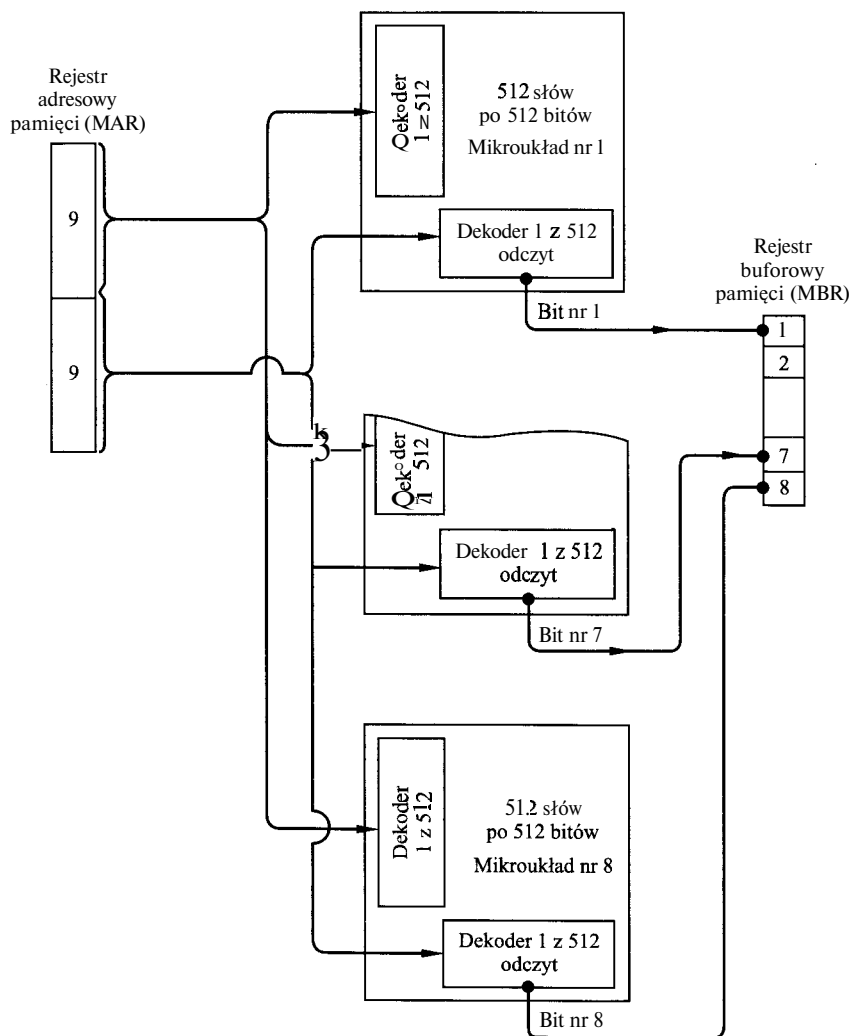
RYSunek 4.7. Rozkład wyprowadzeń i sygnały typowej obudowy mikroukładu pamięciowego: (a) pamięć EPROM 8 Mbit, (b) pamięć DRAM 16 Mbit

- odczytywane dane, wymagające 8 linii (D0÷D7);
- a zasilanie mikroukładu (V_{CC});
- a uziemienie (V_{SS});
- a uaktywnienie mikroukładu (CE); ponieważ może występować wiele mikroukładów pamięciowych, z których każdy jest dołączony do tej samej szyny adresowej, końcówka CE jest wykorzystywana do określania, czy adres dotyczy danego mikroukładu; końcówka CE jest pobudzana przez układy logiczne dołączane do linii najbardziej znaczących bitów szyny adresowej (tzn. do linii bitów powyżej A19).
- a napięcie programowania (V_{pp}), które jest doprowadzane podczas programowania (operacje zapisu).

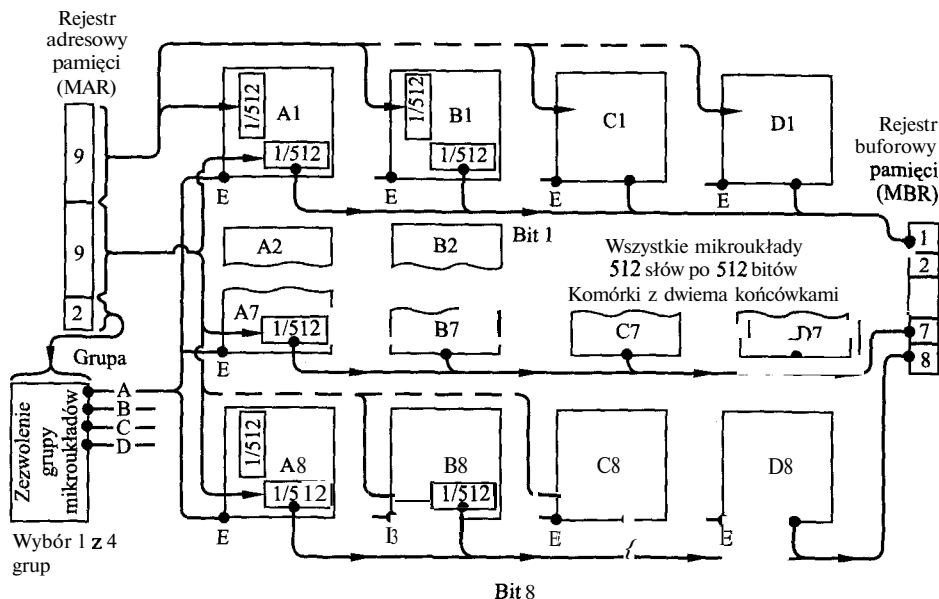
Typowy układ końcówek pamięci DRAM jest pokazany na rys. 4.7b dla mikroukładu 16 Mbit zorganizowanego jako 4 M x 4. Występuje tu kilka różnic w stosunku do układu ROM. Ponieważ pamięć RAM może być aktualizowana, końcówki danych są końcówkami wejścia/wyjścia. Końcówki zezwolenia zapisu (WE) i zezwolenia na wyprowadzanie (OE) wskazują na to, czy prowadzona jest operacja zapisu lub odczytu. Ponieważ pamięć DRAM jest adresowana za pomocą wierszy i kolumn, a adres jest multipleksowany, tylko 11 końcówek adresu potrzeba do określenia 4 M kombinacji wiersz/kolumna ($2^{11} \times 2^{11} = 2^{22} = 4 \text{ M}$). Funkcje wyboru adresu wiersza (RAS) i wyboru adresu kolumny (CAS) zostały już przedyskutowane uprzednio.

Organizacja modułu

Jeśli mikroukład RAM zawiera tylko 1 bit na słowo, będziemy oczywiście potrzebowali przynajmniej liczby mikroukładów równej liczbie bitów w słowie. Na rysunku 4.8 jest pokazane przykładowo, jak może być zorganizowany moduł pamięci zawierający 256 K słów 8-bitowych. W przypadku 256 K słów wymagany jest adres 18-bitowy; jest on dostarczany do modułu z pewnego źródła zewnętrznego (np. z linii adresowych magistrali, do której moduł jest dołączony). Adres jest doprowadzany do 8 mikroukładów 256 K x 1, z których każdy umożliwia wejście/wyjście 1 bita.



RYSUNEK 4.8. Organizacja pamięci 256 KB

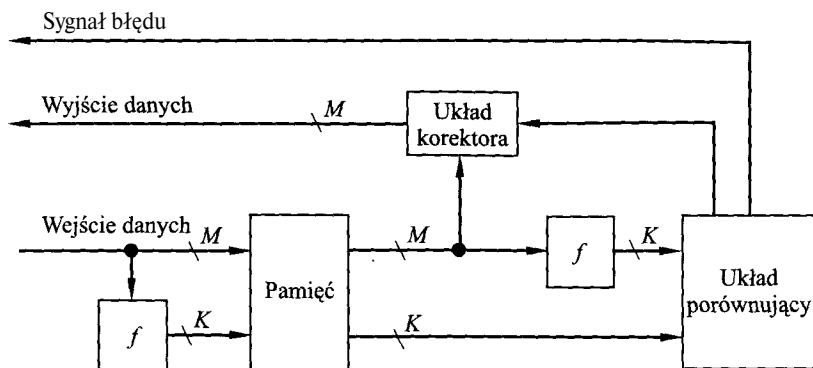


RYSUNEK 4.9. Organizacja pamięci i MB

Organizacja taka jest odpowiednia, dopóki rozmiar pamięci mierzony w słowach jest równy liczbie bitów w mikroukładzie. W przypadku gdy wymagana jest większa pamięć, potrzebna jest matryca mikroukładów. Na rysunku 4.9 jest pokazana możliwa organizacja pamięci składającej się z 1 M słów 8-bitowych. W tym przypadku występują 4 kolumny mikroukładów; każda kolumna zawiera 256 K słów uporządkowanych w sposób pokazany na rys. 4.9. W przypadku 1 M słów wymaganych jest 20 linii adresowych. Osiemnaście najmniej znaczących bitów doprowadza się do wszystkich 32 modułów. Dwanaście najbardziej znaczących bitów są doprowadzone do modułu logicznego wyboru grupy, który wysyła sygnał uaktywnienia mikroukładu do jednej z 4 kolumn modułów.

Korekcja błędów

W systemach pamięci półprzewodnikowych występują błędy. Można je podzielić na błędy stałe i przypadkowe. **Błąd** stały jest permanentnym defektem i zazwyczaj powodującym, że uszkodzona komórka lub komórki pamięciowe nie są w stanie niezawodnie przechowywać danych, lecz pozostają w stanie 0 lub błędnie przeskakują między 0 a 1. Błędy stałe mogą być wywołane przez działanie ostrych narażeń środowiskowych, defekty produkcyjne lub zużycie. Błąd przypadkowy jest losowym i nieniszczącym zjawiskiem, które zmniejsza zawartość jednej lub wielu komórek pamięciowych, bez uszkodzania samej pamięci. Błędy takie mogą być spowodowane przez problemy zasilania lub ciepłoty.



RYSUNEK 4.10. Działanie kodu korekcyjnego

steczki alfa. Cząsteczki te są rezultatem rozpadu promieniotwórczego i są niebezpiecznie powszechne, ponieważ jądra promieniotwórcze znajdują się w małych ilościach prawie we wszystkich materiałach. Zarówno błędy stałe, jak i przypadkowe są oczywiście niepożądane i większość współczesnych systemów pamięci głównych zawiera układy logiczne wykrywające i korygujące błędy.

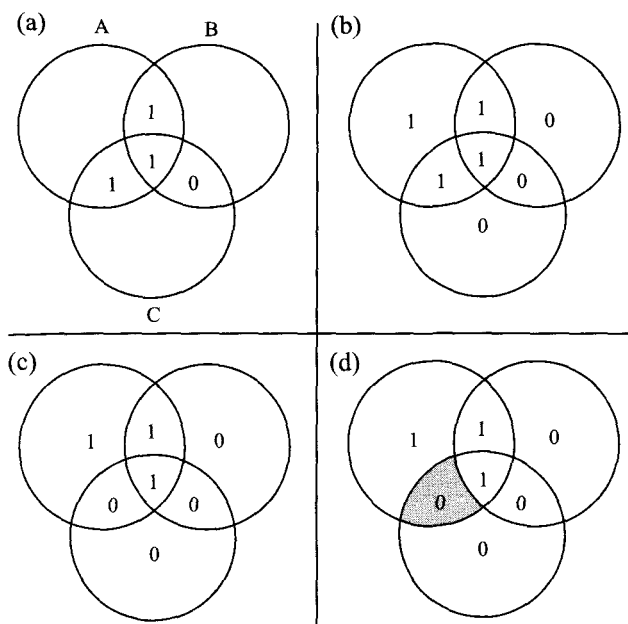
Na rysunku 4.10 jest pokazane w sposób ogólny, jak ten proces jest przeprowadzany. Gdy dane mają być wczytane do pamięci, przeprowadza się na tych danych obliczenia, określane jako funkcja f , w celu utworzenia kodu do korekty błędów. Zarówno kod, jak i dane są przechowywane. W rezultacie, jeśli ma być zapisane M -bitowe słowo danych, a kod ma długość K bitów, to aktualna długość przechowywanego słowa wynosi $M+K$ bitów.

Gdy uprzednio zmagazynowane słowo jest odczytywane, kod jest wykorzystywany do wykrywania i ewentualnej korekty błędów. Generowany jest nowy zestaw K bitów kodowych z M bitów danych, po czym porównuje się go z pobranymi bitami kodowymi. Porównanie prowadzi do jednego z trzech wyników:

- Q Nie wykryto żadnych błędów. Pobrane bity danych są wysyłane.
- Q Wykryto błąd, którego korekta jest możliwa. Bity danych i bity korekty błędu są doprowadzane do układu korektora, który tworzy poprawiony zestaw M bitów przeznaczony do wysłania.
- Wykryto błąd niemożliwy do poprawienia. Stan ten jest zgłaszany.

Kody funkcjonujące w ten sposób są określane jako **kody korekcyjne**. Kod jest charakteryzowany przez liczbę błędów bitowych w słowie, którą może on wykryć i poprawić.

Najprostszym kodem korekcyjnym jest **kod Hamminga**, opracowany przez Richarda Hamminga z Bell Laboratories. Na rysunku 4.11 do zilustrowania wykorzystania tego kodu w odniesieniu do słów 4-bitowych ($M = 4$) zostały wykorzystane wykresy Venna. W przypadku trzech krzyżujących się okręgów mamy do czynienia z siedmioma przedziałami. Przypisujemy 4 bity danych przedziałom wewnętrznym (rys. 4.11a). Pozostałe przedziały są wypełniane tzw. **bitami**



RYSUNEK 4.11. Kod korekcyjny Hamminga

parzystości. Bit parzystości jest wybierany tak, że całkowita liczba jedyne" w jego okręgu jest parzysta (rys. 4.11b). Wobec tego, ponieważ okrąg A zawiera 3 jedynki danych, bit parzystości w tym okręgu jest ustawiany na 1. Jeśli teraz błąd spowoduje zmianę jednego z bitów danych (rys. 4.11c), jest on łatwy do wykrycia. Sprawdzając bity parzystości, wykrywamy sprzeczności w okręgach A i C, ale nie w B. Tylko jeden z 7 przedziałów znajduje się w A i C, ale nie w B. Błąd może więc być poprawiony przez zmianę tego bitu.

W celu wyjaśnienia wykorzystanej koncepcji opracujemy kod, który może posłużyć do wykrycia i skorygowania 1-bitowego błędu w słowach 8-bitowych (przykład zaczerpnięty z [ALTN79]).

Na wstępie określimy wymaganą długość kodu. Zgodnie z rys. 4.10 logiczne układy porównujące otrzymują na wejściu dwie wartości K -bitowe. Porównanie bit po bicie jest przeprowadzane przy użyciu bramki LUB wyłączone (*exclusive-or-EXOR*) o 2 wejściach. Wynik jest określany jako słowo-syndrom. Tak więc każdy bit syndromu jest 0 lub 1 zależnie od tego, czy jest, czy też nie ma zgołności bitów na dwóch wejściach.

Słowo-syndrom ma więc K bitów i zakres wartości 2^K (pomiędzy 0 a $2^K - 1$). Wartość 0 wskazuje, że nie został wykryty żaden błąd, zaś $2^K - 1$ wartości służą do wskazania błędu i jego lokalizacji bitowej. Ponieważ błąd może wystąpić w każdym z M bitów danych i K bitów kontrolnych, musimy mieć

$$2^K - 1 > M + K$$

Równanie to określa liczbę bitów wymaganą do skorygowania błędu 1-bitowego w słowie zawierającym M bitów danych. W tabeli 4.3 jest podane, ile liczb bitów kontrolnych jest wymaganych w przypadku słów o różnej długości.

TABELA 4.3. Wzrost długości słowa po uwzględnieniu korekty błędu

Liczba bitów danych	Poprawianie pojedynczego błędu		Poprawianie pojedynczego błędu, wykrywanie podwójnego błędu	
	Bity kontrolne	% wzrostu	Bity kontrolne	% wzrostu
8	4	50	5	62,5
16	5	31,25	6	37,5
32	6	18,75	7	21,875
64	7	10,94	8	12,5
128	8	6,25	9	7,03
256	9	3,52	10	3,91

Widzimy w tej tabeli, że słowo zawierające 8 bitów danych wymaga 4 bitów kontrolnych. Dla wygody chcielibyśmy generować syndrom 4-bitowy o następujących właściwościach:

- Jeśli syndrom zawiera same 0, to znaczy, że nie został wykryty żaden błąd.
- Jeśli syndrom zawiera jedną i tylko jedną 1, błąd wystąpił w jednym z bitów kontrolnych. Wówczas korekta jest niepotrzebna.
- a Jeśli syndrom zawiera więcej niż jedną 1, to wartość numeryczna syndromu wskazuje pozycję błędnego bitu danych. Korekta polega na inwersji tego bitu.

W celu uzyskania takich właściwości bity danych i bity kontrolne są aranżowane w postaci słowa 12-bitowego w sposób pokazany na rys. 4.12. Pozycje bitowe są ponumerowane od 1 do 12. Pozycje bitowe, których numery są potęgą 2, są wyznaczone jako bity kontrolne. Bity kontrolne są obliczane następująco (symbol $\odot$ oznacza operację LUB wykluczające):

$$\begin{aligned} C1 &= M1 \odot M2 \odot M4 \odot M5 \odot M7 \\ C2 &= M1 \odot M3 \odot M4 \odot M6 \odot M7 \\ C4 &= M2 \odot M3 \odot M4 \odot M8 \\ C8 &= M5 \odot M6 \odot M7 \odot M8 \end{aligned}$$

Każdy bit kontrolny działa na każdej pozycji bitu danych, której numer zawiera 1 w odpowiedniej pozycji kolumny. Tak więc pozycje bitowe danych 3, 5, 7, 9 i 11 zawierają człon 2^0 ; pozycje bitowe 3, 6, 7 i 10 zawierają człon 2^1 ; pozycje bitowe 5, 6, 7 i 12 - człon 2^2 , a pozycje bitowe 9, 10, 11 i 12 - 2^3 . Inaczej mówiąc, pozycja bitowa n jest sprawdzana przez bity C_i takie, że $\sum i = n$. Na przykład pozycja 7 jest sprawdzana przez bity znajdujące się na pozycjach 4, 2 i 1; zachodzi więc $4 + 2 + 1 = 7$.

Pozycja bitowa	Numeryczni	Wzrost	Wzrost
12	1 1 0 0		M8
11	1 0 1 1		M7
10	1 0 1 0		M6
9	1 0 0 1		M5
8	1 0 0 0	C8	
7	0 1 1 1		M4
6	0 1 1 0		M3
5	0 1 0 1		M2
4	0 1 0 0	C4	
3	0 0 1 1		M1
2	0 0 1 0	C2	
1	0 0 0 1	C1	

RYSUNEK 4.12. Rozkład bitów danych i bitów kontrolnych

Sprawdźmy na przykładzie, jak funkcjonuje ten schemat. Załóżmy, że 8-bitowym słowem wejściowym jest 00111001, z bitem danych M1 na najbardziej znaczącej pozycji. Obliczenia są następujące:

$$C1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$

$$C2 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$

$$C4 = 0 \oplus 0 \oplus 1 \oplus 0 = 1$$

$$C8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

Założmy teraz, że bit danych 3 zawiera błąd i jest zmieniony z 0 na 1. Po ponownym przeliczeniu bitów kontrolnych mamy:

$$C1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$

$$C2 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$$

$$C4 = 0 \oplus 0 \oplus 1 \oplus 0 = 0$$

$$C8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

Gdy teraz nowe bity kontrolne porównamy ze starymi, tworzymy słowo-syndrom:

	C8	C4	C2	C1
	0	1	1	1
$\oplus$	0	0	0	1
	0	1	1	0

Pozycja bitowa	12	11	10	9	8	7	6	5	4	3	2	1
Bit danych	M8	M7	M6	M5		M4	M3	M2		M1		
Bit kontrolny					C8				C4		C2	C1
	1	1	1	1		0	0	0		0		C8 0
	1	0	1	0		1	1	1		0		C4 1
	0	1	1	0		1	1	0		1		C2 1
	0	1	0	1		1	0	1		1		C1 1
Słowo zapisane jako:	0	0	1	1	0	1	0	0	1	1	1	
Słowo pobrane jako:	0	0	1	1	0	1	1	0	1	1	1	
	1	1	1	1		0	0	0		0		C8 0
	1	0	0	0		1	1	1		0		C4 0
	0	1	1	0		1	1	0		1		C2 0
	0	1	0	1		1	0	1		1		C1 1

RYSUNEK 4.13. Generowanie bitów kontrolnych

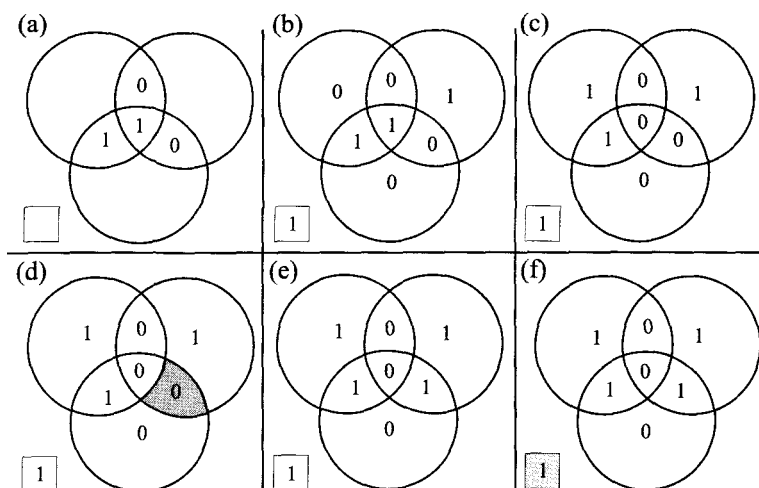
Wynikiem jest 0110, co wskazuje, że pozycja bitowa 6, zawierająca bit danych 3, jest błędna.

Na rysunku 4.13 są zilustrowane powyższe obliczenia. Bity danych i bity kontrolne zostały odpowiednio umieszczone w słowie 12-bitowym. Dzięki ułożeniu numerów każdego z bitów danych w kolumny, jedynki w każdym wierszu wskazują bity danych sprawdzane przez bit kontrolny przypisany temu wierszowi. Ponieważ wynik jest zmieniany tylko przez jedynki, tylko kolumny zawierające jedynki zostały zakreślone dla identyfikacji. Bity kontrolne mogą być następnie obliczone wzdłuż wierszy. Pokazano wyniki dla pierwotnych bitów danych oraz dla bitów danych zawierających błąd.

Opisany właśnie kod jest znany jako **kod poprawiania pojedynczego błędu** (ang. *single-error-correcting* - SEC). Częściej pamięć półprzewodnikowa jest wyposażana w kod poprawiania pojedynczego i podwójnego błędu (SEC-DED). Jak widać w tabeli 4.3, takie kody wymagają jednego dodatkowego bitu w porównaniu z kodami SEC.

Na rysunku 4.14 jest pokazane, jak działa taki kod, znów dla przypadku 4-bitowego słowa danych. Przedstawiona sekwencja ujawnia, że jeśli występują dwa błędy (rys. 4.14c), procedura kontrolna jest błędna i pogarsza problem, tworząc trzeci błąd (rys. 4.14e). W celu pokonania tej trudności, dodany jest ósmy bit, taki że całkowita liczba jedynek na wykresie jest parzysta. Dodatkowy bit parzystości umożliwia wychwycenie błędu (rys. 4.14f).

Kod korygowania błędów umożliwia poprawienie niezawodności pamięci drogą zwiększenia jej złożoności. W przypadku jednobitowej organizacji mikroukładów kod SEC-DED jest na ogół uważany za odpowiedni. Na przykład w maszynach IBM 30xx dla każdego 64 bitów danych w pamięci głównej jest wykorzystywany 8-bitowy kod SEC-DED. W rezultacie rozmiar pamięci głównej jest o około 12% większy niż obserwowany przez użytkownika. W komputerach VAX jest wykorzystywany 7-bitowy kod SEC-DED dla każdego 32 bitów pamięci, co oznacza nadwyżkę 22%.



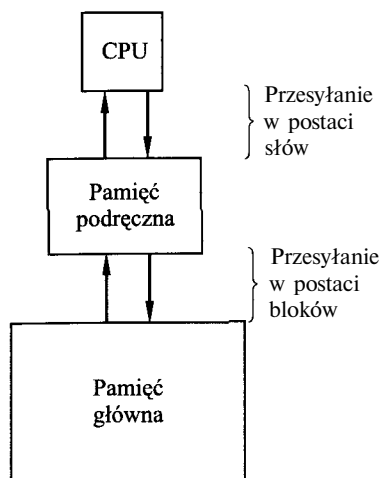
RYSUNEK 4.14. Kod Hamminga SEC-DEC

4.3

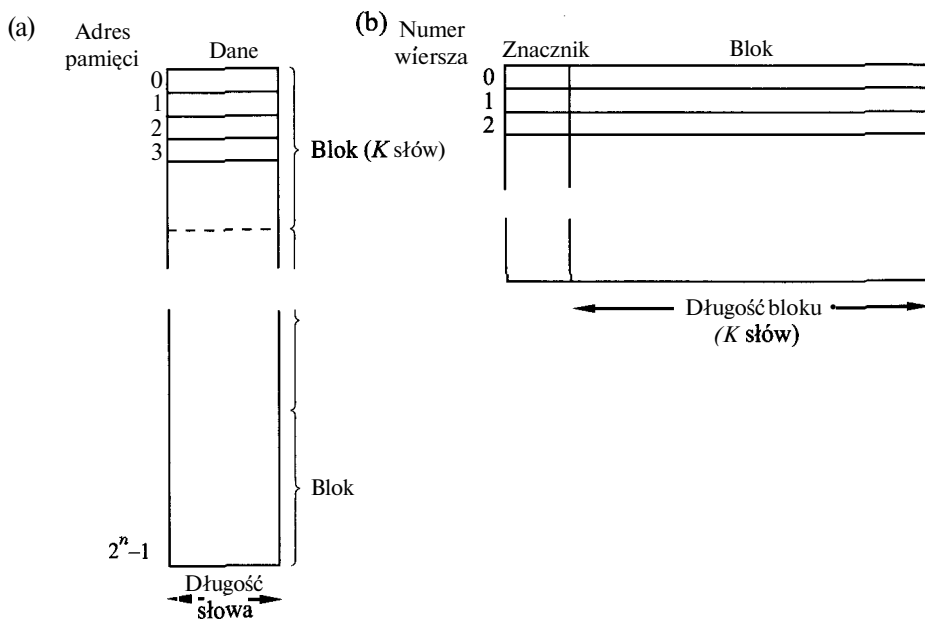
PAMIĘĆ PODRĘCZNA**Zasady**

Pamięci podręczne stosuje się w celu uzyskania pamięci o takiej szybkości, jaką mają najszybsze osiągalne pamięci, z jednoczesnym uzyskaniem dużego rozmiaru pamięci w cenie tańszych rodzajów pamięci półprzewodnikowych. Koncepcja ta jest zilustrowana na rys. 4.15. Występuje tu względnie duża i wolniejsza pamięć główna obok mniejszej i szybszej pamięci podręcznej. Pamięć podręczna zawiera kopię części zawartości pamięci głównej. Gdy procesor zamierza odczytać słowo w pamięci, najpierw następuje sprawdzenie, czy słowo to nie znajduje się w pamięci podręcznej. Jeśli tak, to słowo jest dostarczane do procesora. Jeśli nie, to blok pamięci głównej zawierający ustaloną liczbę słów jest wczytywany do pamięci podręcznej, a następnie słowo jest dostarczane do procesora. Ze względu na zjawisko lokalności odniesień, jeśli blok danych został pobrany do pamięci podręcznej w celu zaspokojenia pojedynczego odniesienia do pamięci, jest prawdopodobne, że przyszłe odniesienia będą dotyczyły innych słów zawartych w tym samym bloku.

Na rysunku 4.16 jest pokazana struktura systemu pamięć podręczna/pamięć główna. Pamięć główna składa się z 2^n adresowalnych słów, przy czym każde słowo ma jednoznaczny adres n -bitowy. Aby było możliwe odwzorowywanie pamięć ta składa się z pewnej liczby bloków o stałej długości, zawierających K słów każdy. Tak więc występuje $M = T/K$ bloków. Pamięć podręczna zawiera C wierszy zawierających K słów każdy, a liczba wierszy jest znacząco mniejsza

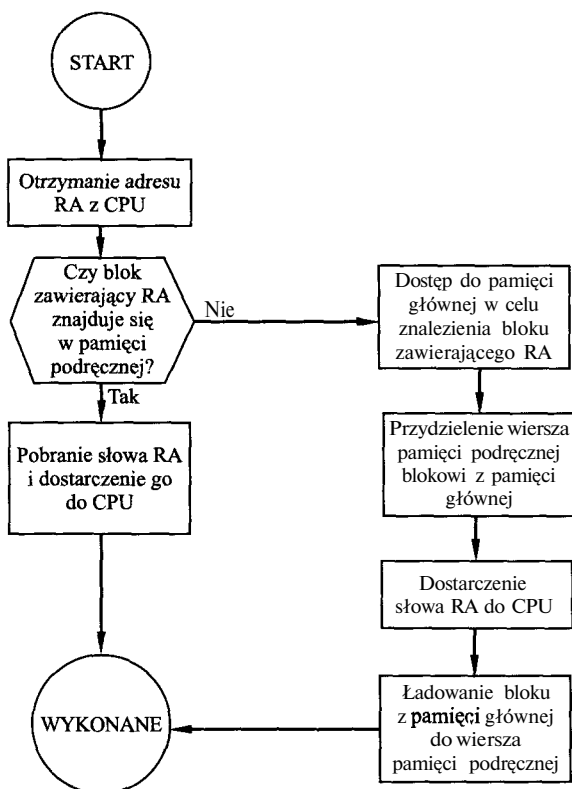


RYSUNEK 4.15. Pamięć podręczna i pamięć główna



RYSUNEK 4.16. Struktura pamięci podręcznej i pamięci głównej: (a) pamięć główna; (b) pamięć podręczna

od liczby bloków w pamięci głównej ($C \ll M$). W każdej chwili, pewien zespół bloków pamięci pozostaje w wierszach pamięci podręcznej. Jeśli słowo w bloku pamięci jest odczytywane, to odpowiedni blok jest przenoszony do jednego z wierszy pamięci podręcznej. Ponieważ bloków jest więcej niż wierszy, określony wiersz nie może być jednoznacznie i trwale przypisany określonemu blo-



RYSUNEK 4.17. Operacja odczytu z pamięci podręcznej

kowi. Każdy wiersz zawiera w związku z tym znacznik określający, który blok jest właśnie przechowywany. Znacznik jest zwykle częścią adresu pamięci głównej, o czym powiemy w dalszej części podrozdziału.

Na rysunku 4.17 jest pokazana operacja odczytu. Procesor generuje adres słowa, które ma być odczytane, RA. Jeśli słowo jest zawarte w pamięci podręcznej, jest dostarczane do procesora. W przeciwnym razie blok zawierający to słowo jest ładowany do pamięci podręcznej, po czym słowo jest dostarczane do procesora.

Dyskusja parametrów wydajności związanych z pamięcią podręczną znajduje się w dodatku 4A.

Elementy projektowania pamięci podręcznych

Chociaż istnieje już wielka liczba implementacji pamięci podręcznych, niewiele jest podstawowych elementów projektowania, które służą do klasyfikowania i różnicowania architektury pamięci podręcznych. W tabeli 4.4 są wymienione takie główne elementy.

TABELA 4.4. Elementy projektowania pamięci podręcznych

Pojemność pamięci	Sposób zapisu
Rodzaj odwzorowywania	jednoczesny
bezpośrednie	opóźniony
skojarzeniowe	jednokrotny
sekcyjno-skojarzeniowe	Rozmiar bloku
Algorytm wymiany	Liczba pamięci podręcznych
najmniej ostatnio używany (LRU)	jedno- lub dwupoziomowa
pierwszy wszedł - pierwszy wychodzi (FIFO)	jednolita lub rozszczepiona
najrzadziej używany (LFU)	
swobodny	

Rozmiar pamięci podręcznej

Pierwszy element, rozmiar pamięci podręcznej, już omówiliśmy. Chcielibyśmy, żeby rozmiar pamięci podręcznej był na tyle mały, żeby ogólny przeciętny koszt na bit był zbliżony do tego kosztu samej pamięci głównej, oraz na tyle duży, żeby ogólny przeciętny czas dostępu był zbliżony do czasu dostępu samej pamięci podręcznej. Występuje jeszcze parę innych powodów minimalizowania rozmiarów pamięci podręcznej. Im większa jest pamięć podręczna, tym większa jest też liczba bramek służących do adresowania tej pamięci. W rezultacie duże pamięci podręczne są nieco wolniejsze niż małe - nawet jeśli są wykonane za pomocą tej samej technologii układów scalonych i umieszczone w tym samym miejscu w mikroukładzie i w obwodzie drukowanym. Rozmiar pamięci podręcznej jest również limitowany dostępną powierzchnią mikroukładu i obwodu drukowanego. W dodatku 4A wskazujemy, że zgodnie z wieloma badaniami rozmiar pamięci podręcznej między 1 K a 512 K słów wydaje się optymalny. Ponieważ wydajność pamięci podręcznej jest bardzo wrażliwa na naturę realizowanych zadań, nie jest możliwe ściśle określenie optymalnego rozmiaru tej pamięci.

Funkcja odwzorowywania

Ponieważ wierszy w pamięci podręcznej jest mniej niż bloków pamięci głównej, wymagany jest algorytm odwzorowywania bloków pamięci głównej w wierszach pamięci podręcznej. Ponadto potrzebne są środki do określania, który blok pamięci głównej aktualnie zajmuje wiersz pamięci podręcznej. Wybór funkcji odwzorowywania dyktuje organizację pamięci podręcznej. Mogą być wykorzystywane trzy metody: bezpośrednia, skojarzeniowa i sekcyjno-skojarzeniowa. Omówimy je kolejno. W każdym przypadku rozpatrzmy strukturę ogólną, a następnie szczególny przykład. Dla wszystkich trzech przypadków przykład zawiera następujące elementy:

1. Pamięć podręczna może przechowywać 64 KB.
2. Dane są przenoszone między pamięcią główną a pamięcią podręczną w blokach po 4 bajty. Oznacza to, że pamięć podręczna jest zorganizowana w postaci $16\text{ K} = 2^{14}$ wierszy po 4 bajty każdy.
3. Pamięć główna składa się z 16 MB, przy czym każdy bajt jest bezpośrednio adresowalny za pomocą 24-bitowego adresu ($2^{24} = 16\text{ M}$). Tak więc, aby umożliwić odwzorowywanie, możemy traktować pamięć główną jako składającą się z 4 M bloków po 4 bajty każdy.

Odwzorowanie bezpośrednie

Najprostsza metoda, znana jako odwzorowanie bezpośrednie, polega na odwzorowywaniu każdego bloku pamięci głównej na tylko jeden możliwy wiersz pamięci podręcznej. Schemat tej metody jest pokazany na rys. 4.18. Odwzorowanie jest wyrażane jako

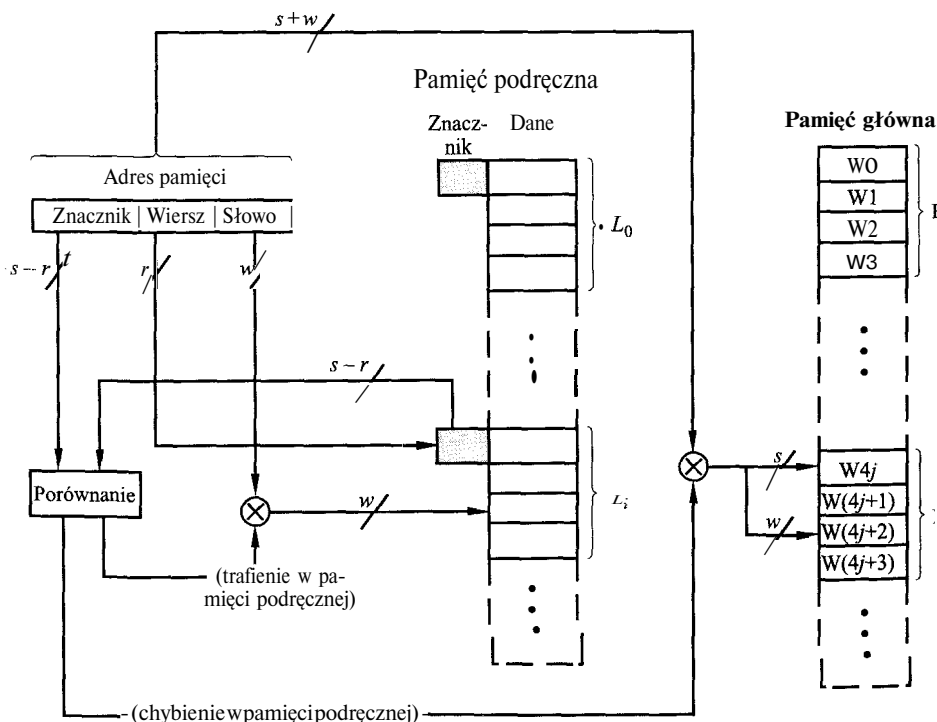
$$i = j \text{ modulo } m$$

gdzie:

i - numer wiersza pamięci podręcznej,

j - numer bloku pamięci głównej,

m - liczba wierszy w pamięci podręcznej.



RYSUNEK 4.18. Organizacja pamięci podręcznej o bezpośrednim odwzorowaniu

Funkcja odwzorowywania może być z łatwością realizowana za pomocą adresu. W celu uzyskania dostępu do pamięci podręcznej każdy adres pamięci głównej może być widziany jako składający się z 3 pól. Najmniej znaczące bity w określającym jednoznacznie słowo lub bajt w bloku pamięci głównej; w najnowocześniejszych maszynach adres jest formułowany na poziomie bajtów. Pozostałych s bitów określa jeden z 2^s bloków pamięci głównej. Układy logiczne pamięci podręcznej interpretują te s bitów jako znaczniki w postaci $s - r$ bitów (najbardziej znacząca część) oraz pole linii złożone z r bitów. To ostatnie pole identyfikuje jeden z $m = 2^r$ wierszy pamięci podręcznej. Wynikiem tego odwzorowywania jest to, że bloki pamięci głównej są przypisane do wiersza pamięci podręcznej następująco:

Wiersz pamięci podręcznej	Przypisane bloki pamięci głównej
0	$0, m, \dots, 2^s - m$
1	$1, m + 1, \dots, 2^s - m + 1$
.	.
.	.
$m - 1$	$m - 1, 2m - 1, \dots, 2^s - 1$

Tak więc, wykorzystanie części adresu jako numeru wiersza pozwala na jednoznaczne odwzorowanie każdego bloku pamięci głównej w pamięci podręcznej. Gdy blok jest właśnie wczytywany do przypisanego wiersza, konieczne jest zaznaczenie danych, aby odróżnić je od innych bloków, które mogą pasować do tego wiersza. Do tego celu służą najbardziej znaczące bity $s - r$.

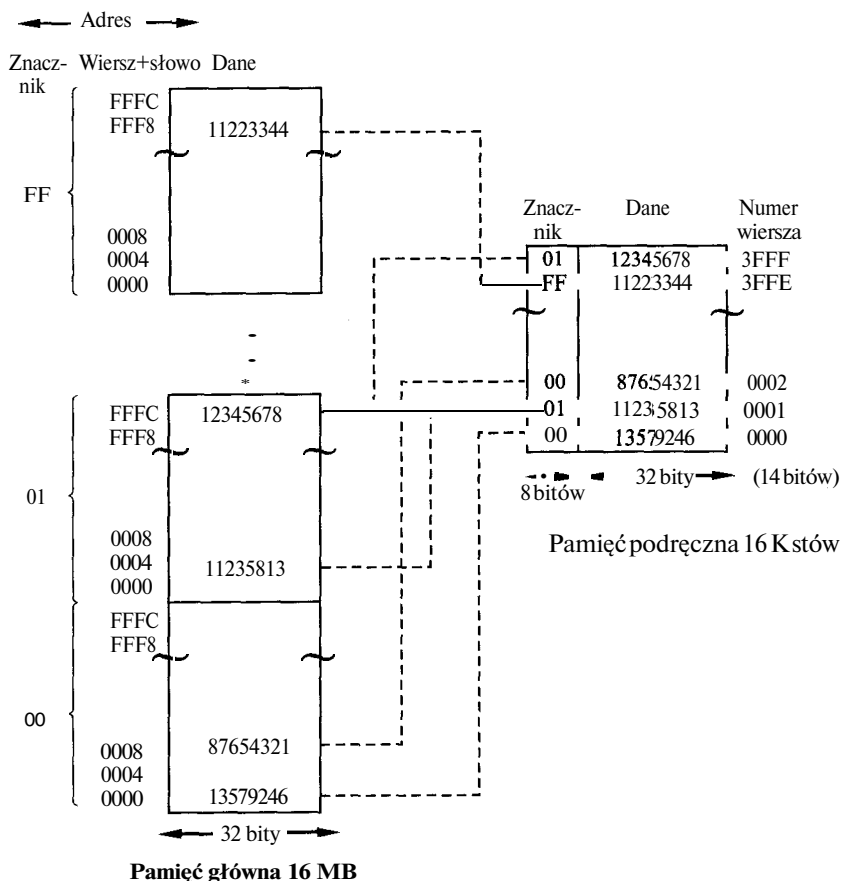
Na rysunku 4.19 jest pokazany nasz przykładowy system stosujący odwzorowanie bezpośrednie². W tym przykładzie $m = 16 \text{ K} = 2^{14}$ oraz $i = j$ modulo 2^{14} . Odwzorowanie jest następujące:

Wiersz w pamięci podręcznej	Przypisane bloki pamięci głównej
0	000000, 010000, ..., FF0000
1	000001, 010001, ..., FF0001
.	.
.	.
3FFF	00FFFC, 01FFFC, ..., FFFFFC

Zauważmy, że żadne dwa bloki odwzorowane w tym samym wierszu nie mają takiego samego znacznika. Bloki 000000, 010000, ..., FF0000 mają znaczniki odpowiednio 00, 01, ..., FF.

Wracając do rys. 4.17, operację odczytu możemy przedstawić następująco. System pamięci podręcznej jest prezentowany za pomocą adresów 24-bitowych. Czternastobitowy numer wiersza jest wykorzystywany jako indeks dostępu do

² Na rysunku 4.19 adresy i wartości w pamięci są reprezentowane za pomocą wygodnej notacji szesnastkowej. Czytelnik, któremu ta notacja jest obca, może się z nią zapoznać w rozdz. 8.



	Znacznik	Wiersz	Słowo
Adres pamięci głównej =	8	14	2

RYСУNEK 4.19. Przykład odwzorowania bezpośredniego

określonego wiersza w pamięci podręcznej. Jeśli 8-bitowy znacznik jest zgodny ze znacznikiem aktualnie przechowywanym w tym wierszu, to 2-bitowy numer słowa służy do wyboru jednego z 4 bajtów w tym wierszu. W przeciwnym razie 22-bitowe pole obejmujące znacznik i wiersz jest wykorzystywane do pobrania bloku z pamięci głównej. Adres użyty do tego pobierania składa się z tych 22 bitów powiązanych z dwoma bitami zerowymi, tak więc na granicy bloku są pobierane 4 bajty.

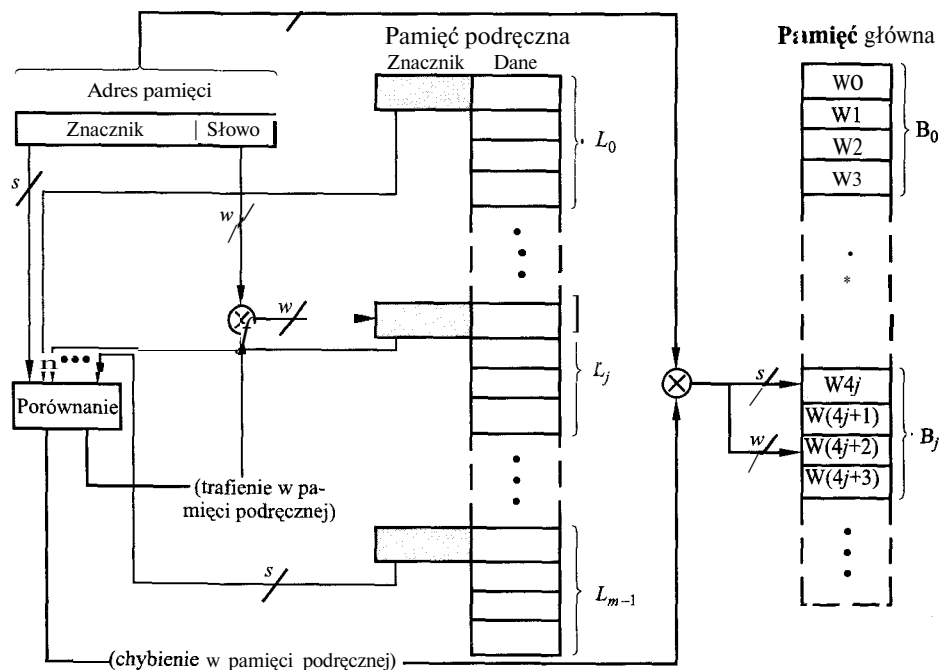
Metoda odwzorowywania bezpośredniego jest prosta i tania przy wdrażaniu. Jej główną wadą jest to, że dla danego bloku istnieje stała lokalizacja w pamięci podręcznej. W rezultacie, jeśli program będzie się często odwoływał do słów z dwóch różnych bloków przypisanych do tego samego wiersza, to bloki te będą ciągle przenoszone do pamięci podręcznej, co obniży ogólną szybkość.

Odzworowywanie skojarzeniowe

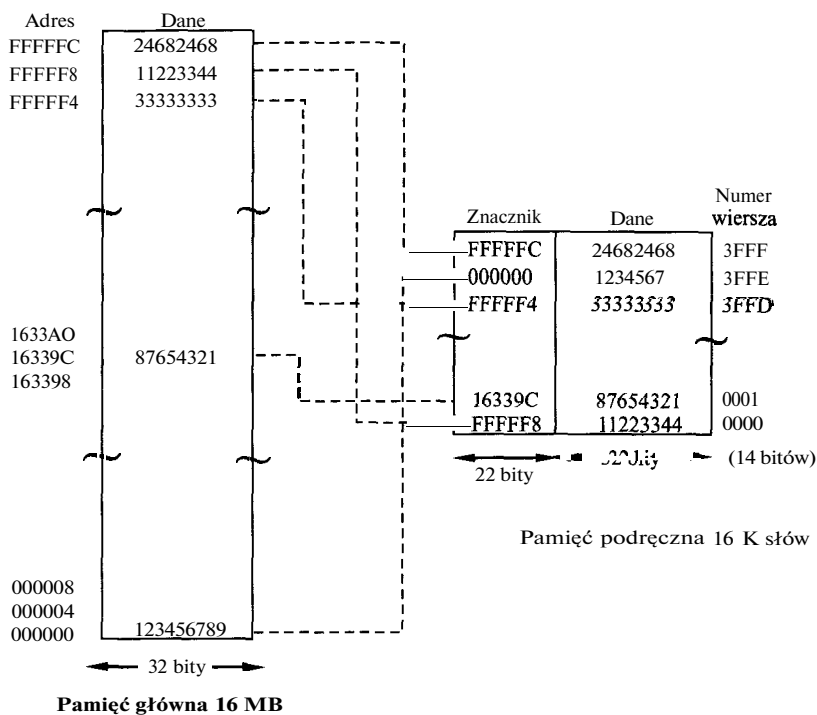
Odzworowywanie skojarzeniowe eliminuje wady odzworowywania bezpośredniego, gdyż umożliwia ładowanie każdego bloku pamięci głównej do dowolnego wiersza pamięci podręcznej. W tym przypadku sterujące układy logiczne pamięci podręcznej interpretują adres pamięci po prostu jako znacznik i pole słowa. Pole znacznika jednoznacznie określa blok pamięci głównej. W celu stwierdzenia, czy blok znajduje się w pamięci podręcznej, sterujące układy logiczne pamięci podręcznej muszą jednocześnie zbadać zgodność znacznika każdego wiersza. Na rysunku 4.20 jest pokazany taki właśnie schemat logiczny.

Na rysunku 4.21 widać naszą przykładową pamięć w wersji odzworowywania skojarzeniowego. Adres pamięci głównej składa się z 22-bitowego znacznika i 2-bitowego numeru bajtu. W przypadku każdego wiersza pamięci podręcznej 22-bitowy znacznik musi być przechowywany razem z 32-bitowym blokiem danych.

W przypadku odzworowywania skojarzeniowego blok do zastąpienia jest wybierany elastycznie, kiedy nowy blok jest wczytywany do pamięci podręcznej. Opracowano algorytmy zastępowania służące do maksymalizacji współczynnika trafień, które omówimy w dalszej części tego podrozdziału. Główną wadą odzworowywania skojarzeniowego są złożone układy wymagane do równoległego badania znaczników wszystkich wierszy pamięci podręcznej.



RYСУNEK 4.20. Organizacja w pełni skojarzeniowej pamięci podręcznej



	Znacznik	Słowo
Adres pamięci głównej =	22	2

RYSUNEK 4.21. Przykład odwzorowania skojarzeniowego

Odwzorowywanie sekcyjno-skojarzeniowe

Odwzorowywanie sekcyjno-skojarzeniowe stanowi kompromis łączący zalety odwzorowywania bezpośredniego i skojarzeniowego; jest ono pozbawione wad charakteryzujących te metody. W tym przypadku pamięć podręczna jest dzielona na v sekcji, z których każda składa się z k wierszy. Zależności są następujące

$$m = v \times k$$

$$i - j \text{ modulo } v$$

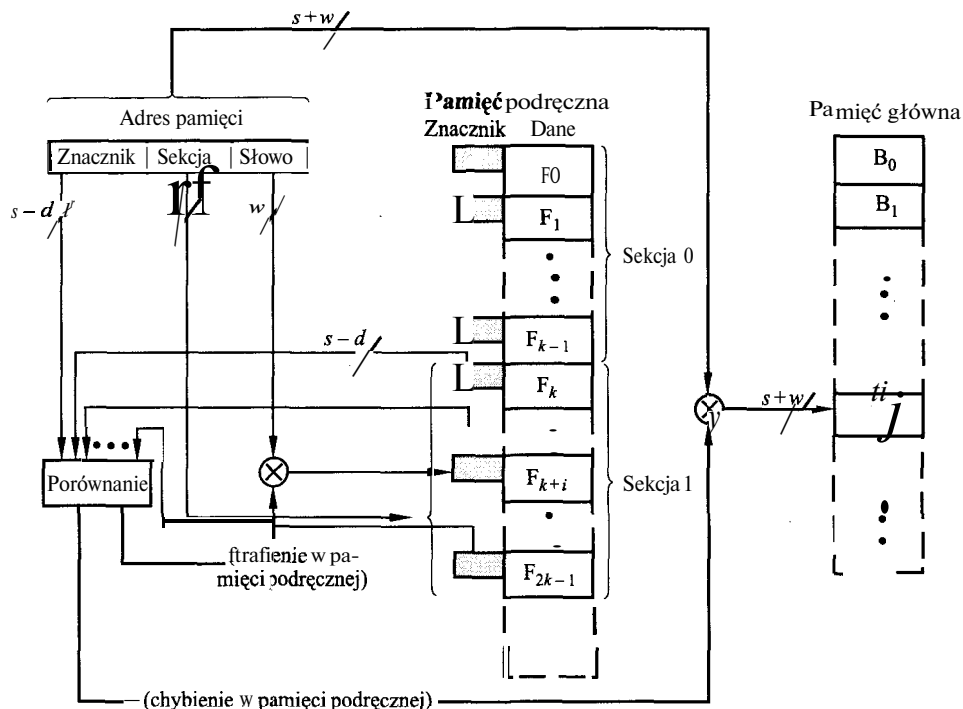
gdzie:

i - numer sekcji pamięci podręcznej,

j - numer bloku pamięci głównej,

m - liczba wierszy pamięci podręcznej.

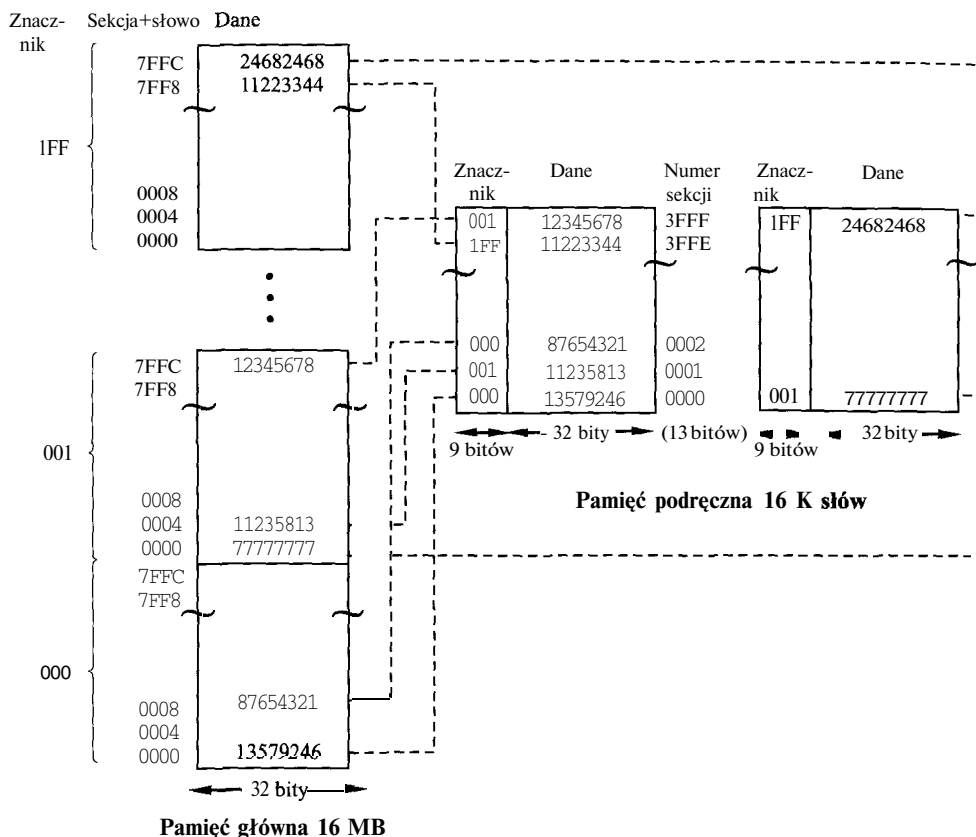
W przypadku odwzorowywania sekcyjno-skojarzeniowego blok B_j może być odwzorowywany na dowolny wiersz sekcji i . W tym przypadku sterujący układ logiczny pamięci podręcznej interpretuje adres pamięci jako trzy pola: znacznik



RYСУNEK 4.22. Dwudrożna sekcyjno-skojarzeniowa organizacja pamięci podręcznej [HWAN93]

sekcja i słowo. Za pomocą d bitów sekcji precyzuje się jedną z $v = 2^d$ sekcji, a s bitów znacznika w połączeniu z polem sekcji określa jeden z 2^s bloków pamięci głównej. Na rysunku 4.22 są przedstawione sterujące układy logiczne pamięci podręcznej.

Na rysunku 4.23 jest pokazana nasza przykładowa pamięć w wersji odwzorowywania sekcyjno-skojarzeniowego z dwoma wierszami w każdej sekcji; układ można określić jako dwudrożny sekcyjno-skojarzeniowy. Za pomocą 13-bitowego numeru sekcji identyfikuje się jednoznacznie określoną sekcję złożoną z 2 wierszy wewnątrz pamięci skojarzeniowej. Określa się także numer bloku w pamięci głównej, modulo 2^{13} , co umożliwia odwzorowanie bloków na wierszach. Tak więc bloki 000000, 00A000, ..., FF1000 pamięci głównej są przypisane sekcji 0 pamięci podręcznej. Dowolny z tych bloków może być załadowany do jednego z dwóch wierszy tej sekcji. Zauważmy, że żaden z dwóch bloków przypisanych tej samej sekcji pamięci podręcznej nie ma takiego samego numeru znacznika. W przypadku operacji odczytu 13-bitowy numer sekcji jest wykorzystywany do określenia, która sekcja ma być badana. Następnie jest sprawdzana zgodność obu wierszy sekcji z numerem znacznika adresu, do którego chcemy uzyskać dostęp.



	Znacznik	Sekcja	Słowo
Adres pamięci głównej =	9	13	2

RYSUNEK 4.23. Dwudrożne odwzorowanie sekcji-skojarzeniowe

W krańcowym przypadku, gdy $v = m$, $k = 1$, metoda sekcji-skojarzeniowa redukuje się do odwzorowywania bezpośredniego, a przy $v = 1$, $k = m$ - redukuje się do odwzorowywania skojarzeniowego. Wykorzystywanie dwóch wierszy sekcję ($v = m/2$, $k = 2$) jest najbardziej powszechną organizacją sekcji-skojarzeniową. Poprawia ona znacząco współczynnik trafienia w stosunku do odwzorowywania bezpośredniego. Czterodrożne odwzorowywanie sekcji-skojarzeniowe ($v = m/4$, $k = 2$) umożliwia uzyskanie umiarkowanej poprawy w zamian za względnie mały koszt dodatkowy [MAYB84, HILL89]. Dalej zwiększanie liczby wierszy w sekcji przynosi małe efekty.

Algorytmy zastępowania

Gdy do pamięci podręcznej jest wprowadzany nowy blok, jeden z istniejących bloków musi być zastąpiony. W przypadku odwzorowywania bezpośredniego istnieje tylko jeden możliwy wiersz dla każdego określonego bloku i wybór nie jest możliwy. W przypadku metody skojarzeniowej i sekcyjno-skojarzeniowej wymagany jest algorytm zastępowania. Aby uzyskać dużą szybkość, algorytm taki musi być wbudowany w postaci sprzętowej. Wypróbowano wiele algorytmów. Wspomnimy o czterech najbardziej powszechnych. Prawdopodobnie najbardziej efektywny jest algorytm „**najmniej ostatnio używany**” (ang. *least-recently used* - LRU). Algorytm określa, że należy zastąpić ten blok w sekcji, który pozostawał w pamięci podręcznej najdłużej bez odwoływania się do niego. W przypadku dwudroźnego odwzorowywania sekcyjno-skojarzeniowego jest to łatwe do wdrożenia. Każdy wiersz zawiera bit wykorzystania (USE). Gdy wiersz jest adresowany, jego bit USE jest ustawiany na 1, a bit USE pozostałego wiersza w tej sekcji jest ustawiany na 0. Jeśli blok ma być wczytany do sekcji, wykorzystuje się wiersz, którego bit USE ma wartość 0. Ponieważ zakładamy, że niedawno wykorzystywane lokacje pamięci są bardziej prawdopodobne do wykorzystania w przyszłości, LRU powinien dawać najlepszy współczynnik trafień. Inną możliwość stwarza algorytm „**pierwszy wchodzi - pierwszy wychodzi**” (FIFO). Polega on na zastępowaniu tego bloku w sekcji, który najdłużej pozostawał w pamięci podręcznej. Algorytm FIFO jest łatwy do wdrożenia w postaci metody cyklicznego buforowania. Jeszcze inną możliwość stanowi algorytm „**najrzadziej używany**” (LFU). Określa on, że zastępowany jest ten blok w sekcji, którego dotyczyło najmniej odniesień. Algorytm LFU można wdrożyć przez skojarzenie licznika z każdym wierszem. Metodą abstrahującą od ilości odniesień jest **przypadkowy** wybór wśród kandydujących wierszy. Badania symulacyjne wykazały, że losowe zastępowanie prowadzi do niewiele tylko mniejszej wydajności w stosunku do algorytmów opartych na wykorzystywaniu [SMIT82].

Algorytm zapisu

Zanim blok pozostający w pamięci podręcznej będzie mógł być zastąpiony, konieczne jest rozważenie, czy musi on być zmieniony w pamięci podręcznej, a nie w pamięci głównej. Jeśli nie musi, to stary blok w pamięci podręcznej może być nadpisany. Jeśli musi, to znaczy, że na słowie zawartym w tej linii pamięci podręcznej musi być przeprowadzona przynajmniej jedna operacja zapisu, a pamięć główna musi być odpowiednio zaktualizowana. Możliwe są różne algorytmy zapisu wykorzystujące współzależności dotyczące wydajności i ekonomii. Istnieją dwa problemy, z którymi musimy się borykać. Po pierwsze, do pamięci głównej może mieć dostęp więcej niż jedno urządzenie. Na przykład, moduł wejścia-wyjścia może być zdolny do odczytu/zapisu bezpośrednio w pamięci. Jeśli sło-

wo byłoby zmienione tylko w pamięci podręcznej, to odpowiednie słowo w pamięci głównej byłoby nieaktualne. Ponadto, jeśli moduł wejścia-wyjścia zmieniłby słowo w pamięci głównej, to odpowiednie słowo w pamięci podręcznej byłoby nieważne. Bardziej złożony problem występuje, gdy mamy do czynienia z wieloma procesorami podłączonymi do tej samej magistrali i wyposażonymi we własne, lokalne pamięci podręczne. Jeśli w takiej sytuacji byłoby zmienione słowo w jednej pamięci podręcznej, spowodowałoby to unieważnienie słowa w innych pamięciach podręcznych.

Najprostsza metoda jest określana jako **zapis jednoczesny** (ang. *write through*). Przy jej zastosowaniu wszystkie operacje zapisu są prowadzone jednocześnie zarówno do pamięci głównej, jak i do pamięci podręcznej, co zapewnia stałą aktualność danych w pamięci głównej. Jakikolwiek inny moduł procesor - pamięć podręczna może monitorować przesyłanie do pamięci głównej w celu zachowania spójności (*consistency*) swojej pamięci podręcznej. Główną wadą tej metody jest to, że generuje ona znaczny przepływ danych do pamięci i może przez to spowodować występowanie wąskich gardeł. Metoda alternatywna, nazwana metodą **zapisu opóźnionego** (ang. *write back*), minimalizuje ilość operacji zapisu do pamięci. W przypadku tej metody aktualizuje się tylko pamięć podręczną. Gdy następuje taka aktualizacja, określana jest wartość bitu aktualizacji (UPDATE) skojarzonego z wierszem pamięci podręcznej. Jeśli następnie blok jest zastępowany, to podlega on wpisaniu do pamięci głównej tylko wtedy, gdy jest ustanowiony bit UPDATE. Problemem tej metody jest to, że część zawartości pamięci głównej jest nieaktualna, przez co dostęp modułów wejścia-wyjścia jest możliwy tylko za pośrednictwem pamięci podręcznej. Komplikuje to układy i stwarza możliwość wąskich gardeł. Doświadczenie wykazało, że udział zapisów w operacjach dostępu do pamięci jest rzędu 15% [SMIT82].

W przypadku organizacji magistralowej, w której więcej niż jedno urządzenie (zwykle procesor) ma pamięć podręczną, a pamięć główna jest wspólna, powstaje nowy problem. Jeśli dane w jednej pamięci podręcznej są zmieniane, to operacja taka powoduje unieważnienie odpowiedniego słowa nie tylko w pamięci głównej, ale także w innych pamięciach podręcznych (jeśli to słowo w nich akurat występuje). Jeśli nawet jest stosowana metoda zapisu jednoczesnego, to inne pamięci podręczne mogą zawierać nieaktualne dane. System, który zapobiega temu problemowi, jest nazywany systemem zapewniającym spójność pamięci podręcznej. Do możliwych rozwiązań problemu spójności pamięci podręcznej należą:

- **Obserwowanie magistrali z zapisem jednoczesnym.** Każdy sterownik pamięci podręcznej monitoruje linie adresowe w celu wykrycia operacji zapisu do pamięci dokonywanych przez inne jednostki nadrzędne magistrali. Jeśli inna jednostka nadrzędna wpisuje dane do takiej lokacji w pamięć wspólną, która występuje również w pamięci podręcznej, to sterownik pamięci podręcznej unieważnia dostęp do pamięci podręcznej. Strategia ta jes

uzależniona od tego, czy wszystkie sterowniki pamięci podręcznych posługują się metodą zapisu jednoczesnego.

- a *Sprzętowe zapewnianie spójności* (ang. *hardware transparency*). Wykorzystywane są dodatkowe rozwiązania sprzętowe zapewniające, że wszystkie aktualizacje pamięci głównej dokonywane za pośrednictwem pamięci podręcznej znajdują odzwierciedlenie we wszystkich pamięciach podręcznych. Jeśli jeden procesor modyfikuje słowo w swojej pamięci podręcznej, aktualizacja ta jest wprowadzana również do pamięci głównej. Ponadto, wszystkie odpowiednie słowa w pozostałych pamięciach podręcznych są podobnie aktualizowane.
- a *Pamięć wyłączona ze współpracy pamięcią podręczną* (ang. *non-cachable memory*). Tylko część pamięci głównej jest wspólna dla więcej niż jednego procesora i ta właśnie część jest oznaczana jako wyłączona ze współpracy z pamięcią podręczną. W takim systemie wszystkie operacje dostępu do pamięci wspólnej nie dotyczą pamięci podręcznych, ponieważ pamięć wspólna nigdy nie jest kopiowana do pamięci podręcznych. Pamięć wyłączona ze współpracy z pamięcią podręczną może być identyfikowana za pomocą układów logicznych wyboru mikroukładu lub najbardziej znaczących bitów adresu.

Spójność pamięci podręcznej stanowi czynne pole badań. Jest prawdopodobne, że w ciągu najbliższych kilku lat pojawiają się prostsze i bardziej efektywne sposoby zapewniania spójności.

Rozmiar bloku

Kolejnym elementem projektowania jest rozmiar bloku lub wiersza. Gdy blok danych jest pobierany i umieszczany w pamięci podręcznej, pobierane jest nie tylko żądane słowo, lecz również pewna liczba sąsiednich słów. Gdy zwiększamy rozmiar bloku od bardzo małego do dużego, współczynnik trafień najpierw wzrasta ze względu na zasadę lokalności: wysokie prawdopodobieństwo, że dane sąsiadujące z wywołanym słowem będą potrzebne w niedalekiej przyszłości. W miarę wzrostu rozmiaru bloku do pamięci podręcznej doprowadzana jest coraz większa ilość użytecznych danych. Przy dalszym wzroście współczynnik trafień zacznie jednak maleć, a prawdopodobieństwo wykorzystania nowo pobieranych informacji stanie się mniejsze niż prawdopodobieństwo ponownego użycia informacji, która jest zastępowana. Grają tu rolę dwa szczególne zjawiska:

1. Użycie większych bloków powoduje zmniejszenie liczby bloków, które mieszczą się w pamięci podręcznej. Ponieważ każde wpisanie bloku powoduje usunięcie starej zawartości pamięci podręcznej, mała liczba bloków sprawia, że dane są usuwane niedługo po ich pobraniu.
2. Gdy blok staje się większy, każde dodatkowe słowo jest dalsze od potrzebnego słowa, staje się więc mniej prawdopodobne jego użycie w niedalekiej przyszłości.

Związek między rozmiarem bloku a współczynnikiem trafień jest złożony i uzależniony od stopnia lokalności określonego programu. Nie znaleziono definitywnej wartości optymalnej. Rozmiar od 4 do 8 jednostek adresowalnych (słów lub bajtów) wydaje się leżeć rozsądnie blisko optimum [SMIT87a], [PRZY88], [PRZY90].

Liczba pamięci podręcznych

Kiedy wprowadzono pamięci podręczne, typowy system miał pojedynczą pamięć podręczną. Ostatnio stało się normą wykorzystywanie wielu pamięci podręcznych. Dwoma aspektami tego zagadnienia projektowego są: liczba poziomów pamięci oraz podział lub scalanie pamięci podręcznych.

Pamięci podręczne jedno- i dwupoziomowe

W miarę wzrostu gęstości upakowania układów logicznych stało się możliwe dysponowanie pamięcią podręczną wewnątrz tego samego mikroukładu co procesor: określamy to mianem pamięci podręcznej w procesorze (ang. *on-chip cache*). W porównaniu z pamięcią osiągalną za pomocą zewnętrznej magistrali wewnątrzprocesorowa pamięć podręczna umożliwia zmniejszenie aktywności procesora w magistrali zewnętrznej i przez to skraca się czasy wykonywania operacji i zwiększa się ogólna wydajność systemu. Jeśli wymagane rozkazy lub dane znajdują się w wewnątrzprocesorowej pamięci podręcznej, konieczność dostępu do magistrali jest eliminowana. Ze względu na krótsze ścieżki danych wewnątrz procesora w porównaniu z długościami magistrali, dostęp do wbudowanej pamięci podręcznej jest wyraźnie szybszy w porównaniu z cyklem magistrali, nawet w stanie nie wymagającym oczekiwania. Ponadto, w tym okresie magistrala pozostaje wolna i mogą zachodzić inne transfery.

Wbudowanie pamięci podręcznej do procesora pozostawia otwarte pytanie, czy nadal jest pożądana zewnętrzna (poza mikroukładem procesora) pamięć podręczna. Odpowiedź brzmi zwykle: tak. Większość współczesnych projektów zawiera zarówno pamięć podręczną wewnątrzprocesorową, jak i zewnętrzną. Wynikająca stąd organizacja jest znana jako dwupoziomowa pamięć podręczna, przy czym pamięć wewnętrzna jest oznaczana jako poziom 1 (L1), a zewnętrzna jako poziom 2 (L2). Przyczyna wprowadzania pamięci L2 jest następująca. Jeśli nie byłoby pamięci podręcznej L2, a procesor zgłosiłby zapotrzebowanie na dostęp do lokacji pamięci nie występującej w pamięci podręcznej L1, to procesor musiałby sięgać do pamięci DRAM lub ROM poprzez magistralę. Ze względu na typowo niewielką szybkość magistrali oraz stosunkowo długi czas dostępu do pamięci rezultatem będzie zmniejszona wydajność. Z drugiej strony, jeśli zastosowano pamięć podręczną L2 SRAM, to często możliwe jest szybkie pobranie brakującej informacji. Jeśli pamięć SRAM jest dostatecznie szybka, aby dostosować się do szybkości magistrali, to możliwy jest dostęp do danych za pomocą

transakcji o zerowym czasie oczekiwania, co stanowi najszybszy rodzaj transferów magistralowych.

Potencjalne oszczędności wynikające z zastosowania pamięci podręcznej L2 zależą od współczynników trafień zarówno pamięci L1, jak i L2. W kilku badaniach wykazano, że na ogół zastosowanie pamięci podręcznej drugiego poziomu poprawia wydajność (patrz np. [AZIM92], [NOVI93]).

Jednolita a podzielona pamięć podręczna

Gdy po raz pierwszy wprowadzono wewnątrzprocesorową pamięć podręczną, w wielu projektach zawierających pojedynczą pamięć podręczną wykorzystywano ją zarówno do przechowywania danych, jak i rozkazów. Obecnie stało się powszechne dzielenie pamięci podręcznej na dwie: jedną przeznaczoną na rozkazy, drugą zaś na dane.

Jest kilka potencjalnych zalet pamięci jednolitej:

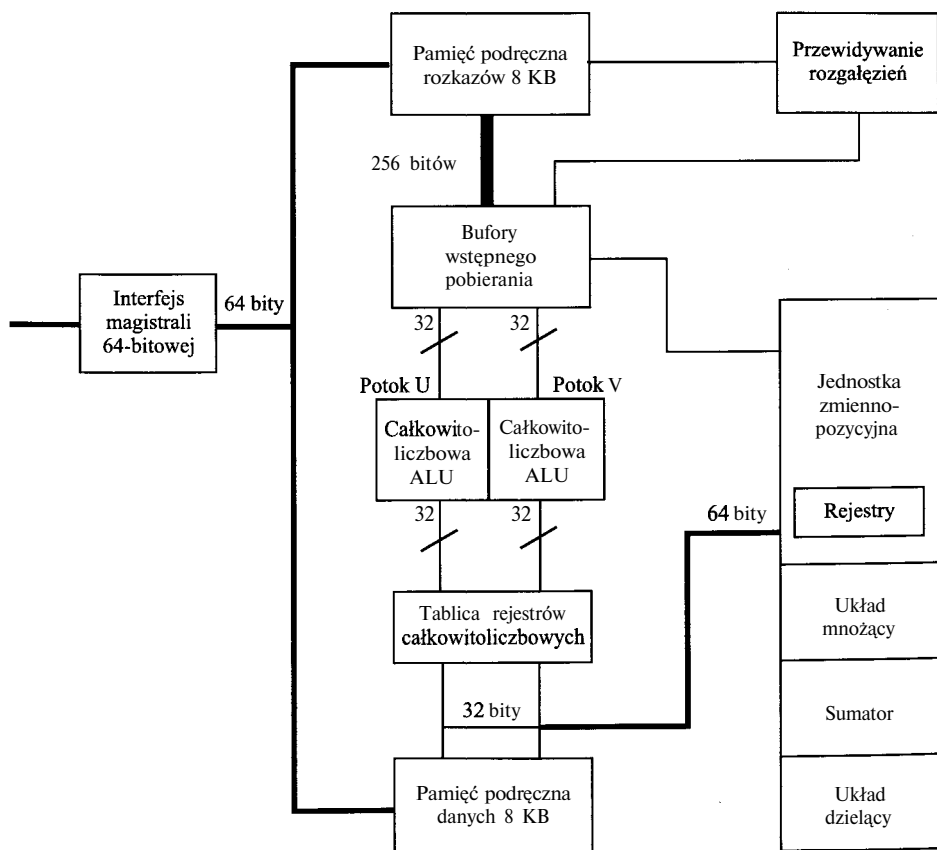
1. Dla danego rozmiaru pamięci podręcznej pamięć jednolita wyróżnia się większym współczynnikiem trafień niż pamięć podzielona, ponieważ automatycznie równoważy ona pobieranie rozkazów z pobieraniem danych. To znaczy, jeśli struktura rozkazów przewiduje o wiele więcej pobrań rozkazów niż danych, pamięć podręczna będzie wykazywała tendencję do wypełniania się rozkazami. Jeżeli natomiast wymagane jest częstsze pobieranie danych, pamięć ta będzie się wypełniała danymi.
2. Tylko jedna pamięć podręczna musi być zaprojektowana i wdrożona.

Mimo tych zalet występuje tendencja do stosowania pamięci podzielonych, szczególnie w przypadku maszyn superskalarnych, jak Pentium i PowerPC, co akcentuje równoległe wykonywanie rozkazów i wstępne pobieranie rozkazów przewidywanych do wykonywania. Kluczową zaletą podzielonej pamięci podręcznej jest to, że eliminowana jest rywalizacja o pamięć podręczną między procesorem rozkazów a jednostką wykonującą. Jest to ważne w każdym rozwiązaniu wykorzystującym potokowe przetwarzanie rozkazów. Typowo procesor pobiera rozkazy z wyprzedzeniem i napełnia bufor - rejestr potokowy - rozkazami, które mają być wykonane. Załóżmy teraz, że mamy do dyspozycji jednolitą pamięć podręczną dla rozkazów i danych. Gdy jednostka wykonująca sięga do pamięci w celu załadowania i przechowania danych, zapotrzebowanie jest doprowadzane do jednolitej pamięci podręcznej. Jeśli w tym samym czasie układ wyprzedzającego pobierania rozkazów wysyła zapotrzebowanie na odczytanie rozkazu z pamięci podręcznej, to zapotrzebowanie to zostanie czasowo zablokowane, żeby pamięć podręczna mogła najpierw obsłużyć jednostkę wykonującą, pozwalając jej na zakończenie właśnie wykonywanego rozkazu. Ta rywalizacja o pamięć podręczną może spowodować zmniejszenie wydajności, interferując z wydajnym wykorzystaniem potoku rozkazów. Struktura podzielonej pamięci podręcznej umożliwia pokonanie tej trudności.

Organizacja pamięci podręcznej Pentium

Ewolucja organizacji pamięci podręcznej może być wyraźnie dostrzeżona w ewolucji mikroprocesorów Intel. Procesor 80386 nie zawiera wewnątrzprocesorowej pamięci podręcznej, natomiast 80486 zawiera wewnątrzprocesorową pamięć podręczną 8 KB, wykorzystującą wiersze o rozmiarze 16 bajtów i cztero-rodzonną organizację sekcyjno-skojarzeniową.

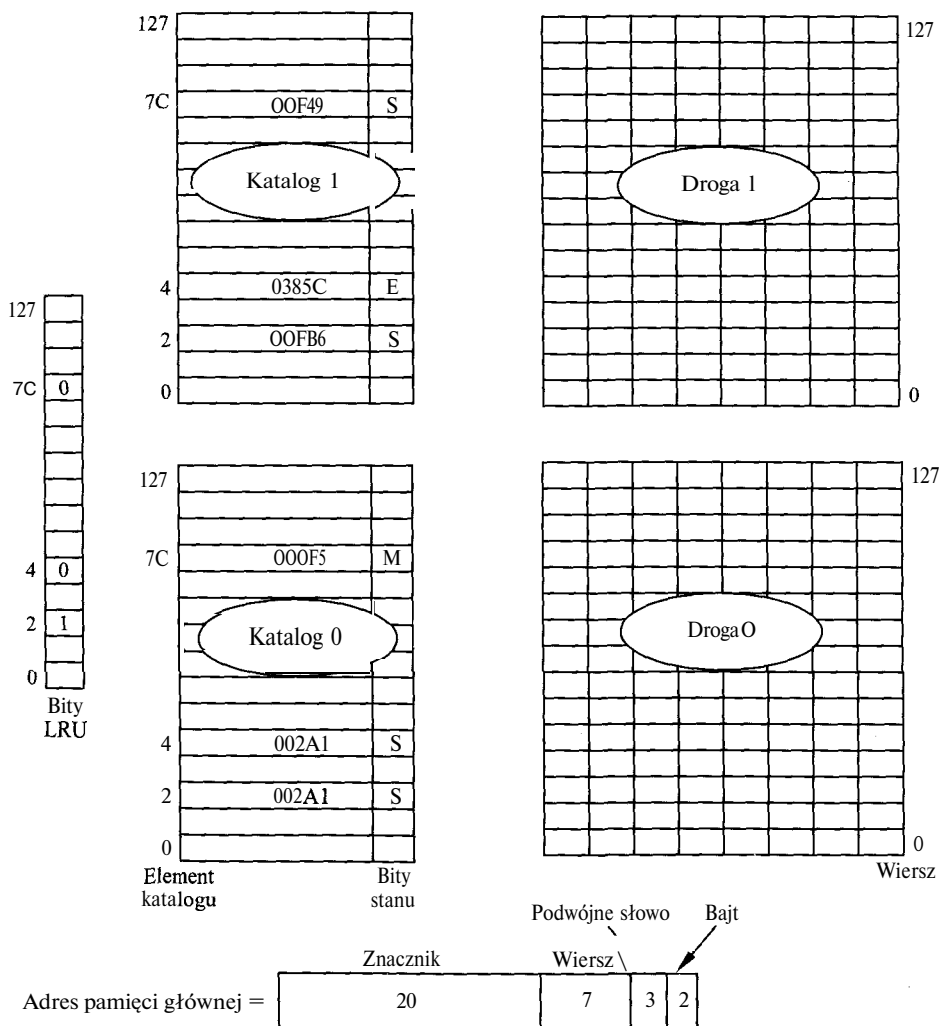
Na rysunku 4.24 jest przedstawiony uproszczony schemat organizacji mikroprocesora Pentium, ze zwróceniem uwagi na rozmieszczenie dwóch pamięci podręcznych. Rdzeniowymi jednostkami wykonawczymi są dwie równoległe działające całkowitoliczbowe jednostki arytmetyczno-logiczne oraz jednostka zmiennopozycyjna mająca własne rejestry i własne układy mnożenia, dodawania i dzielenia. Pamięć podręczna danych jest wykorzystywana zarówno przy operacjach na liczbach całkowitych, jak i przy operacjach zmiennopozycyjnych. Ma ona dwa porty 32-bitowe służące do oddzielnego sprzęgania z dwiema całkowitoliczbowymi jednostkami ALU. Mogą one być łączone w celu stworzenia



RYSUNEK 4.24. Schemat blokowy procesora Pentium

64-bitowego interfejsu z jednostką zmiennopozycyjną. Podręczna pamięć kodów, która jest pamięcią stałą, obsługuje bufor wstępnego pobierania rozkazów; funkcjonowanie potoku rozkazów jest opisane w rozdz. 13.

Na rysunku 4.25 są pokazane kluczowe elementy wewnętrznej pamięci podręcznej. Dane w pamięci tworzą 128 sekcji po 2 linie każda. Organizacja logiczna ma postać dwóch „dróg” po 4 KB. Z każdym wierszem jest skojarzony znacznik oraz 2 bity stanu; są one zorganizowane logicznie w postaci 2 katalogów, tak więc istnieje jedno hasło katalogu dla każdego wiersza pamięci podręcznej. Znacznik obejmuje 20 najbardziej znaczących bitów adresu pamięciowego danych przechowywanych w odpowiednim wierszu. Sterownik pamięci



RYSUNEK 4.25. Struktura wewnętrznej pamięci podręcznej danych procesora Pentium [ANDE93]

podręcznej wykorzystuje algorytm zastępowania typu LRU, tak więc każdej sekcji jest przypisany pojedynczy bit LRU.

W pamięci podręcznej zastosowano algorytm opóźnionego zapisu: dane są wpisywane do pamięci głównej tylko wtedy, gdy są usuwane z pamięci podręcznej i gdy były aktualizowane. Procesor Pentium może być dynamicznie tak skonfigurowany, aby wykorzystywał metodę jednoczesnego zapisu.

Procesor Pentium używa zewnętrznej pamięci podręcznej poziomu 2. Pamięć ta może mieć pojemność 256 lub 512 KB i wykorzystywać wiersze 32-, 64- i 128-bajtowe. W zewnętrznej pamięci podręcznej stosuje się dwudrożne odwzorowanie sekcyjno-skojarzeniowe.

Zgodność pamięci podręcznej danych

W celu zapewnienia zgodności pamięć podręczna danych obsługuje protokół znany jako MESI (ang. *modified/exclusive/shared/invalid*). Protokół MESI został zaprojektowany, aby spełnione były wymagania odnoszące się do zgodności pamięci podręcznych w systemie wieloprocesorowym. Jest jednak również użyteczny w przypadku jednoprocessorowej organizacji Pentium.

Pamięć podręczna danych zawiera dwa bity stanu na każdy znacznik, tak więc każdy wiersz może być w jednym z czterech stanów. Są to stany:

- Q Zmodyfikowany (*modified* - M). Wiersz pamięci podręcznej został zmodyfikowany (różni się w porównaniu z pamięcią główną) i jest dostępny tylko w tej pamięci podręcznej.
- a Wyłączny (*exclusive* - E). Wiersz pamięci podręcznej jest taki sam, jak jego odpowiednik w pamięci głównej i nie jest obecny w jakiegokolwiek innej pamięci podręcznej.
- ❑ Wspólny (*shared* - S). Wiersz pamięci podręcznej jest taki sam, jak jego odpowiednik w pamięci głównej i może występować w innej pamięci podręcznej.
- ❑ Nieważny (*invalid* - I). Wiersz pamięci podręcznej nie zawiera ważnych (aktualnych) danych.

W tabeli 4.5 znajduje się podsumowanie znaczenia tych czterech stanów. Rozważmy najpierw przypadek jednoprocessorowy³. W tym przypadku istotna jest współpraca między pamięciami podręcznymi poziomu 1 i poziomu 2. Po ustawieniu w stanie wyjściowym wiersz pamięci podręcznej pozostaje nieważny (I). Gdy wczytywane są do niego nowe dane, są one najpierw pobierane z pamięci głównej, lokowane w pamięci podręcznej L2, a następnie wprowadzane do pamięci podręcznej danych L1. Stan wiersza w pamięci podręcznej L1 staje się wspólny (S). Następujące po tym odczyty nie zmieniają stanu wiersza.

³ W dyskusji założono, że obie pamięci podręczne L1 i L2 zostały skonfigurowane w sposób umożliwiający zapis opóźniony, co jest rozwiązaniem typowym.

TABELA 4.5. Stany wierszy pamięci podręcznej według MESI

	M	E	S	I
	zmodyfikowany	wyłączny	wspólny	nieważny
Ważność wiersza	tak	tak	tak	nie
Ważność kopii	nieaktualna	ważna	ważna	–
Istnienie kopii w innych pamięciach podręcznych	brak	brak	być może	być może
Zapis w tym wierszu...	nie jest przekazywany do magistrali	nie jest przekazywany do magistrali	jest przekazywany do magistrali i aktualizuje pamięć podręczną	jest przekazywany bezpośrednio do magistrali

Założmy teraz, że jedna z jednostek wykonawczych dokonuje zapisu w tym wierszu. Wiersz w pamięci podręcznej L1 zostanie więc zaktualizowany. Wówczas wiersz w pamięci podręcznej L2 staje się przestarzały. Aby zapobiec niezgodności pamięci podręcznej, za pierwszym razem, gdy aktualizowany jest wiersz w L1, dokonywany jest jednoczesny zapis w pamięci podręcznej L2, a stan wiersza w pamięci L1 ulega zmianie na wyłączny (E). Kolejna aktualizacja wiersza w pamięci L1 powoduje zmianę jego stanu na zmodyfikowany (M). Po dalszych aktualizacjach wiersz pozostaje w stanie M i podczas tych wszystkich dalszych aktualizacji nie zachodzi transfer do pamięci L2. Z tego powodu algorytm ten jest znany jako „zapisz raz” (*write-once*). W końcu, gdy staje się konieczne zastąpienie wiersza w pamięci podręcznej L1, a pozostaje on w stanie S lub E, nie ma potrzeby przepisywania zawartości. Jeśli natomiast jest on w stanie M, jego zawartość jest przepisywana do pamięci podręcznej L2, a następnie wyrzucana z pamięci L1. Gdy do wiersza pamięci L1 są wczytywane nowe dane, zmienia ona stan na S, jak poprzednio.

Operacja ta widziana ze strony pamięci podręcznej L2 jest bardziej złożona. Kiedy wiersz jest po raz pierwszy wczytywany do pamięci podręcznych L1 i L2, wówczas wiersz w pamięci L2 przechodzi do stanu E, co wskazuje, że dane znajdują się wyłącznie w pamięci podręcznej L2, w skojarzonym z nią procesorze i w pamięci L1. Gdy realizowany jest algorytm „zapisz raz”, pamięć L2 aktualizuje wiersz, przez co przechodzi on do stanu M. Pamięć podręczna L2 nie będzie informowana o jakichkolwiek następnych aktualizacjach tego wiersza w pamięci L1. Wobec tego, jeśli inna jednostka nadrzędna magistrali zamierza odczytać dane zawarte w wierszu pamięci L2 znajdującym się w stanie M, to pamięć podręczna L2 powoduje wycofanie się jednostki nadrzędnej magistrali i kieruje zgłoszony adres do procesora Pentium. Procesor przeprowadza cykl opóźnionego zapisu w celu zaktualizowania pamięci głównej. Następnie pamięć podręczna L2 umożliwia jednostce nadrzędnej magistrali wykonanie jej operacji odczytu.

Jeśli inna jednostka nadrzędna magistrali zamierza zapisać dane znajdujące się w wierszu pamięci L2 pozostającym w stanie M, to znów układy logiczne pamięci podręcznej L2 blokują chwilowo tę operację. Pamięć L2 musi zapewnić wykonywanie operacji według właściwej sekwencji - nie może po prostu usunąć zawartość swojego wiersza i pozwolić na realizowanie operacji, ponieważ inna jednostka nadrzędna magistrali może aktualizować inną część wiersza, niż była już zmodyfikowana w wierszu L2. Wobec tego wiersz pamięci L2 musi być przepisany do pamięci głównej, zanim będzie mógł mieć miejsce inny zapis. Zanim to jednak nastąpi pamięć L2 musi stwierdzić, czy odpowiedni wiersz pamięci podręcznej L1 został zaktualizowany od czasu operacji „zapisz raz”. Tak więc właściwa sekwencja jest następująca:

1. Pamięć podręczna L2 wykrywa i blokuje operację zapisu.
2. Pamięć L2 sygnalizuje pamięci podręcznej L1 adres operacji zapisu. Jeśli zawartość pamięci L1 była aktualizowana od czasu operacji „zapisz raz”, to wykonuje ona zapis jednoczesny w pamięci głównej. W każdym przypadku L1 sygnalizuje, że zmieniony wiersz jest nieważny (stan D).
3. Jeśli pamięć L1 nie przeprowadziła zapisu jednoczesnego, pamięć podręczna L2 aktualizuje pamięć główną. W każdym przypadku L2 sygnalizuje, że zmieniony wiersz jest nieważny.
4. Pamięć L2 uwalnia jednostkę nadrzędna magistrali, pozwalając jej na dokończenie operacji zapisu.

Do protokołu MESI wrócimy w rozdz. 16 i opiszemy jego działanie w konfiguracji wieloprocessorowej.

Sterowanie pamięcią podręczną

Wewnętrzna pamięć podręczna jest sterowana za pomocą dwóch bitów w jednym z rejestrów sterowania. Bity te są oznaczone CD (ang. *cache disable* - pamięć podręczna zablokowana) i NW (ang. *not writethrough* - brak zapisu jednoczesnego), co widać w tabeli 4.6. Istnieją także dwa rozkazy procesora Pentium, które mogą być wykorzystane do sterowania pamięcią podręczną. Rozkaz **INVD** powoduje opróżnienie pamięci podręcznej i sygnalizowanie zewnętrznej pamięci podręcznej (jeśli taka istnieje) konieczności opróżnienia. Rozkaz **WBINVD** realizuje tę samą funkcję oraz sygnalizuje zewnętrznej pamięci podręcznej konieczność opóźnionego zapisu zmodyfikowanych bloków przed opróżnieniem.

TABELA 4.6. Tryby działania pamięci podręcznej procesora Pentium

Bity sterowania		Tryb działania		
CD	NW	Zapisy w pamięci podręcznej	Zapisy jednoczesne	Stany nieważności
0	0	dozwolone	dozwolone	dozwolone
1	0	zabronione	dozwolone	dozwolone
1	1	zabronione	zabronione	zabronione

Z badań prowadzonych w firmie Ramtron wynika, że pamięć EDRAM zachowuje się równie dobrze lub lepiej niż zwykła pamięć DRAM uzupełniona zewnętrzną (w stosunku do DRAM), ale większą pamięcią podręczną SRAM.

Pamięć podręczna DRAM

Pamięć podręczna DRAM (CDRAM), opracowana przez Mitsubishi [HIDA90], jest podobna do EDRAM. Zawiera ona większą pamięć podręczną SRAM niż EDRAM (16 wobec 2 Kbit).

Pamięć SRAM zawarta w CDRAM może być używana na dwa sposoby. Po pierwsze, może być wykorzystana jako rzeczywista pamięć podręczna złożona z pewnej liczby wierszy 64-bitowych - w przeciwieństwie do pamięci EDRAM, w której pamięć podręczna SRAM zawiera tylko jeden blok, a mianowicie ostatnio wywoływany wiersz. Praca CDRAM jako pamięci podręcznej polega na zwykłym dostępie swobodnym.

Pamięć SRAM zawarta w CDRAM może być także wykorzystywana jako bufor podtrzymujący szeregowy dostęp do bloku danych. Na przykład, w celu odświeżenia rastrowego obrazu ekranu pamięć CDRAM może z wyprzedzeniem pobierać dane z DRAM do bufora SRAM. Następne operacje dostępu będą już dotyczyły wyłącznie SRAM.

Synchroniczna pamięć DRAM

Zupełnie inne podejście do poprawiania wydajności pamięci DRAM przyjęto w synchronicznej pamięci DRAM (SDRAM) opracowanej wspólnie przez pewną liczbę firm [VOGL94].

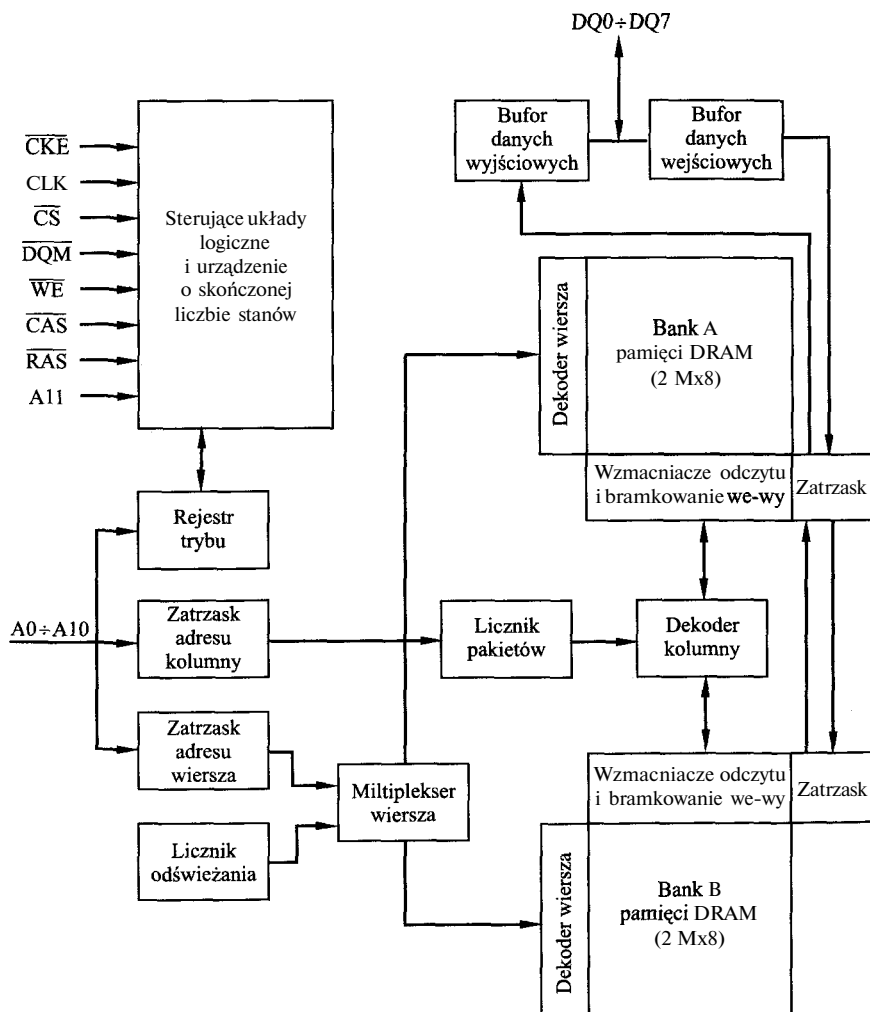
W przeciwieństwie do typowej pamięci DRAM, która jest asynchroniczna, wymiana danych między pamięcią SDRAM a procesorem jest synchronizowana przez sygnał zegara zewnętrznego i zachodzi z pełną szybkością magistrali procesor-pamięć bez narzucania stanów oczekiwania.

W typowej pamięci DRAM procesor przekazuje adresy i sygnały sterujące do pamięci, wskazując że zbiór danych o określonej lokacji w pamięci powinien być albo odczytany z pamięci DRAM, albo do niej zapisany. Po pewnym opóźnieniu (czas dostępu) pamięć DRAM albo odczytuje, albo zapisuje dane. Podczas opóźnienia związanego z czasem dostępu, pamięć DRAM realizuje różne funkcje wewnętrzne, takie jak aktywowanie dużych pojemności linii wierszy i kolumn, odczytywanie danych i wyprowadzanie ich przez bufony wyjściowe. Procesor musi po prostu przeczekać to opóźnienie, zmniejszając wydajność systemu.

W przypadku dostępu synchronicznego pamięć DRAM przenosi dane do wewnątrz i na zewnątrz pod kontrolą zegara systemowego. Procesor lub inna jednostka nadrzędna wydaje rozkaz i podaje informację adresową, które są zatraskiwane w pamięci DRAM. Pamięć DRAM udziela odpowiedzi po upływie

pewnej liczby cykli zegara. W tym czasie jednostka nadrzędna może bezpiecznie realizować inne cele, a pamięć SDRAM przetwarza zgłoszone zapotrzebowanie.

Na rysunku 4.29 są pokazane wewnętrzne układy logiczne pamięci SDRAM. Wykorzystuje ona tryb pakietowy (ang. *burst mode*) w celu wyeliminowania czasu ustalania adresu oraz czasu wstępnego ładowania linii wiersza i kolumny po pierwszej operacji dostępu. W przypadku trybu pakietowego ciągi bitów danych mogą być szybko wyprowadzone za pomocą zegara tuż po uzyskaniu dostępu do pierwszego bitu. Taki tryb pracy jest użyteczny, gdy wszystkie pożądane bity danych są ułożone szeregowo w tym samym wierszu matrycy, którego dotyczyła początkowa operacja dostępu.



RYСУNEK 4.29. Synchroniczna, dynamiczna pamięć RAM (SDRAM) [PRZY94]

Zwróćmy ponadto uwagę, że pamięć SDRAM ma dwusegmentową architekturę wewnętrzną, która umożliwia poprawę warunków równoległych działań w ramach mikroukładu.

Rejestr trybu i związane z nim sterujące układy logiczne stanowią jeszcze jedną istotną własność odróżniającą pamięć SDRAM od konwencjonalnych pamięci DRAM. Pozwalają one na dostosowanie pamięci SDRAM do specyficznych wymagań systemowych. Rejestr trybu ustala długość porcji danych, która jest liczbą oddzielnych jednostek danych synchronicznie doprowadzanych do magistrali. Rejestr ten umożliwia także programiście regulowanie zwłoki między otrzymaniem zapotrzebowania na odczyt a rozpoczęciem transferu danych.

Pamięć SDRAM sprawuje się najlepiej, gdy dokonuje szeregowego transferu dużych bloków danych, co ma miejsce w takich zastosowaniach, jak przetwarzanie tekstów, arkusze kalkulacyjne i multimedia.

Pamięć RDRAM

Pamięć RDRAM, opracowana w firmie Rambus [GARR94], stanowi bardziej rewolucyjne rozwiązanie problemu szerokości pasma pamięci. Mikroukłady RDRAM mają obudowy pionowe, ze wszystkimi końcówkami po jednej stronie. Mikroukład wymienia dane z procesorem poprzez 28 połączeń nie dłuższych niż 12 cm. Magistrala umożliwia adresowanie do 320 mikroukładów RDRAM i może przenosić 500 Mbit/s. Można to porównać z wielkością około 33 Mbit/s dla synchronicznych pamięci DRAM.

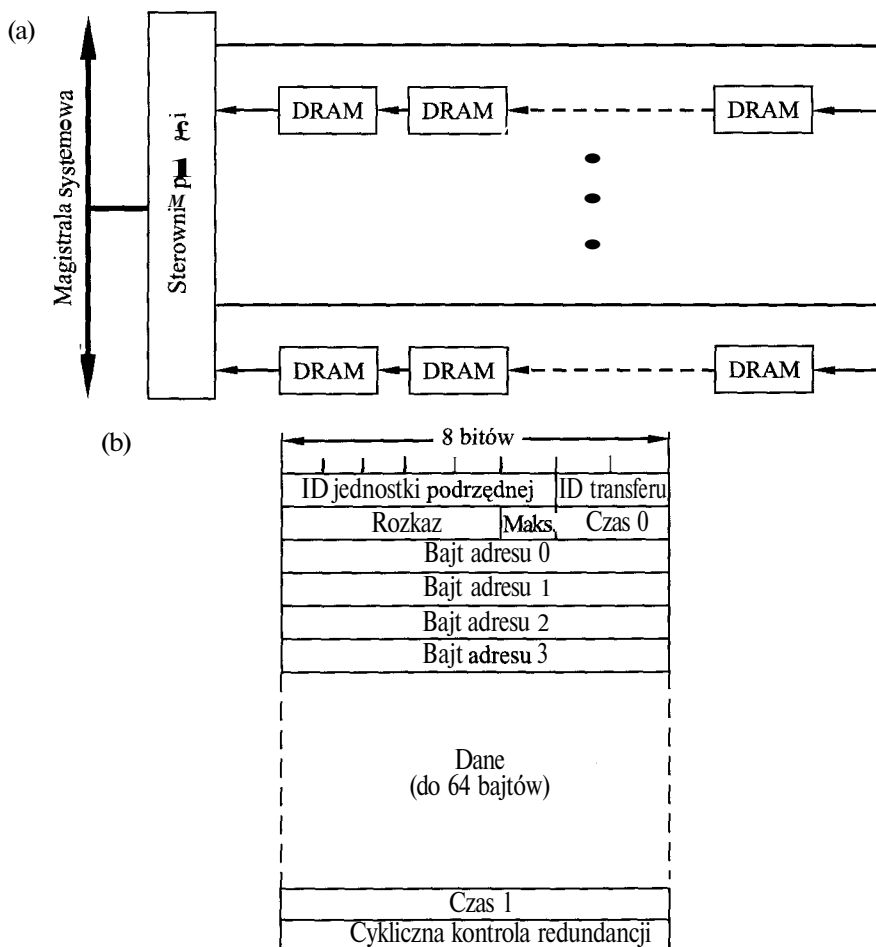
Specjalna magistrala RDRAM dostarcza informacje adresowe i sterujące, wykorzystując asynchroniczny protokół przesyłania danych blokami. Po początkowym czasie dostępu 480 ns osiągana jest szybkość przekazywania danych równa 500 Mbit/s. Tym, co umożliwia taką szybkość, jest sama magistrala, dla której bardzo dokładnie określono impedancje, taktowanie i sygnały. Zamiast sterowania za pomocą jasno określonych sygnałów RAS, CAS, R/W i CE stosowanych w konwencjonalnych pamięciach DRAM, w przypadku pamięci RDRAM dostęp do danych zachodzi poprzez magistralę o dużej szybkości. Żądanie dostępu do danych zawiera adres, rodzaj operacji oraz liczbę bajtów w operacji.

RamLink

Najbardziej radykalne odstępstwo od tradycyjnych pamięci DRAM stanowi pamięć RamLink [GJES92], opracowana przez grupę roboczą IEEE jako część przedsięwzięcia nazwanego **spójnym interfejsem skalowalnym** (*Scalable Coherent Interface*- SCI). Projektanci pamięci RamLink skoncentrowali się raczej na interfejsie procesor-pamięć niż na wewnętrznej architekturze mikroukładów DRAM.

RamLink jest interfejsem pamięciowym z połączeniami zorganizowanymi w postaci pierścienia (rys. 4.30a). Ruchem w pierścieniu zarządza sterownik pamięci, który wysyła wiadomości do mikroukładów DRAM działających jako węzły w pierścieniowej sieci. Dane są wymieniane w formie pakietów (rys. 4.30b).

Transakcje pamięci są inicjowane przez pakiety zapotrzebowania. Są one wysyłane przez sterownik i zawierają nagłówek rozkazu (ang. *command header*), adres, sumę kontrolną oraz - w przypadku rozkazów zapisu - dane, które mają być zapisane. Nagłówek rozkazu zawiera informacje dotyczące typu, rozmiaru i sterowania, a także albo określony czas odpowiedzi, albo maksymalny czas, w ciągu którego wykonawca musi udzielić odpowiedzi. Informacje dotyczące sterowania zawierają bit wskazujący, czy następne zapotrzebowania będą dotyczyły adresów sekwencyjnych. Jednocześnie mogą być aktywne nawet



Rysunek 4.30. Pamięć RamLink: (a) architektura; (b) format pakietu

cztery transakcje na jeden mikroukład; dlatego wszystkie pakiety mają 2-bitowy identyfikator (ID) w celu jednoznacznego powiązania pakietów zapotrzebowania i odpowiedzi.

W przypadku udanego odczytu pamięć DRAM - wykonawca wysyła pakiet odpowiedzi zawierający odczytane dane. W przypadku nieudanego odczytu wykonawca wysyła pakiet ponowienia próby (ang. *retry packet*), który wskazuje, ile dodatkowego czasu potrzebuje na zakończenie transakcji.

Jedną z silnych stron podejścia RamLink jest to, że zapewnia ono skalowalną architekturę umożliwiającą wykorzystanie zarówno małej, jak i dużej liczby pamięci DRAM, a także nie dyktuje wewnętrznej struktury DRAM. Pierścieniowa organizacja RamLink została zaprojektowana w celu skoordynowania działania wielu pamięci DRAM i zapewnienia wydajnego interfejsu ze sterownikiem pamięci.

4.5

POLECANA LITERATURA

Praca [PRIN91]* stanowi obszerne omówienie technologii pamięci półprzewodnikowych, w tym pamięci SRAM, DRAM i pamięci błyskawicznych.

Kody korekty błędów zostały dobrze wyjaśnione w [MCEL85] oraz w [HEGD92]. Bardziej szczegółową analizę umożliwiają pozycje książkowe [ADAM91] i [BLAH83].

Pełne ujęcie projektowania pamięci podręcznych można znaleźć w [HAND93]. Wiele tekstów dotyczących architektury komputerów zawiera również użyteczne wiadomości na temat projektowania pamięci podręcznych; jedno z najbardziej szczegółowych i klarownych ujęć znajduje się w [ALEX93].

Szczegółowy opis organizacji pamięci podręcznej Pentium można znaleźć w [ANDE93] i [INTE94a], natomiast organizację pamięci podręcznej PowerPC przedstawiono w [SHAN94b] i [WEIS94]. Klasycznym, wciąż wartym przeczytania artykułem jest [SMIT82]; dokonano w nim przeglądu różnych elementów projektowania pamięci podręcznych i przedstawiono wyniki wielu badań. Szczegółową analizę różnorodnych problemów projektowania pamięci podręcznych związanych z przetwarzaniem wieloprogramowym i wieloprosesorowym przedstawiono w [AGAR89]. Po proste wzory, które mogą być użyte do szacowania wydajności pamięci podręcznych jako funkcji ich różnych parametrów, można sięgnąć do [HIGB90]

ADAM91 Adamek J.: *Foundations of Coding*. New York, Wiley, 1991.

AGAR89 Agarwal A.: *Analysis of Cache Performance for Operating Systems and Multiprogramming*. Boston, Kluwer Academic Publishers, 1989.

* Nakładem WNT ukazało się w 1999 r. tłumaczenie innej książki tej autorki pt. *Nowoczesne pamięci półprzewodnikowe. Architektura i organizacja układów pamięci DRAM i SRAM (przyp. red.)*.

- ALEX93 Alexandridis N.: *Design of Microprocesor-Based Systems*. Englewood Cliffs, Prentice Hall, 1993.
- ANDE93 Anderson D., Shanley T.: *Pentium Processor System Architecture*. Richardson, Mindshare Press, 1993.
- BLAH83 Blahut R.: *Theory and Practice of Error Control Codes*. Reading, Addison-Wesley, 1983.
- HAND93 Handy J.: *The Cache Memory Book*. San Diego, Academic Press, 1993.
- HEDG92 Hedge A.: „Detect/Correct Errors to Improve Data Reliability”. *Electronic Design*, June 11, 1992.
- HIGB90 Higbie L.: „Quick and Easy Cache Performance Analysis”. *Computer Architecture News*, June 1990.
- INTE94a Intel Corp.: *Pentium Family User's Manual. Vol. 1: Data book*. Santa Clara, 1994.
- MCEL85 McEliece R.: „The Reliability of Computer Memories”. *Scientific American*, January 1985.
- PRIN91 Prince B.: *Semiconductor Memories*. New York, Wiley, 1991.
- SHAN94b Shanley T.: *PowerPC 601 System Architecture*. Richardson, Mindshare Press, 1994.
- SMIT82 Smith A.: „Cache Memories”. *ACM Computing Surveys*, September 1982.
- WEIS94 Weiss S., Smith J.: *POWER and PowerPC*. San Francisco, Morgan Kaufmann, 1994.

4.6

PROBLEMY DO ROZWIĄZANIA

- 4.1. Naszkicuj konfigurację zawierającą procesor, cztery pamięci ROM 1 K x 8 bit oraz magistralę składającą się z 12 linii adresu i 8 linii danych. Dodaj blok logiczny wyboru mikroukładu, który wybiera jeden z czterech modułów ROM dla każdego z 4 K adresów.
- 4.2. Podaj przyczyny, dla których pamięci RAM tradycyjnie miały organizację 1-bitową, podczas gdy pamięci ROM - wielobitową.
- 4.3. Linie adresowe pokazane na rys. 4.7 są multipleksowane, dzięki czemu jest potrzebna tylko połowa linii w porównaniu z układem linii wyspecjalizowanych. Czy spodziewasz się, że takie rozwiązanie spowoduje straty czasowe? Jeśli nie, to uzasadnij odpowiedź.
- 4.4. Rozważ dynamiczną pamięć RAM, która wymaga cyklu odświeżania 64 razy w ciągu 1 ms. Każda operacja odświeżania wymaga 150 ns; cykl pamięci zajmuje 250 ns. Jaka część całego czasu działania pamięci musi być poświęcona na odświeżanie?
- 4.5. Zaprojektuj pamięć 16-bitową o pojemności całkowitej 8192 bit, wykorzystując mikroukłady SRAM 64 x 1 bit. Podaj konfigurację matrycy mikroukładów na płycie drukowanej, wskazując wszystkie wymagane sygnały wejściowe i wyjściowe, które umożliwiłyby przypisanie tej pamięci do najmniejszej przestrzeni adresowej. Projekt powinien pozwalać zarówno na dostęp do bajtów, jak i do słów 16-bitowych.
Źródło: [ALEX93].
- 4.6. Opracuj kod SEC dla 16-bitowego słowa danych. Utwórz ten kod dla słowa danych 010100000111001. Wykaż, że kod ten prawidłowo zidentyfikuje błąd 4. bitu danych.

- 4.7. Pamięć podręczna sekcyjno-skojarzeniowa składa się z 64 wierszy podzielonych na 4-wierszowe sekcje. Pamięć główna zawiera 4 K bloków po 128 słów każdy. Jaki format mają adresy pamięci głównej?
- 4.8. Rozważ mikroprocesor 32-bitowy mający wewnętrzną, czterodroźną pamięć podręczną sekcyjno-skojarzeniową 16 KB. Załóż, że wiersz pamięci podręcznej ma długość czterech słów 32-bitowych. Narysuj schemat blokowy tej pamięci podręcznej ukazujący jej organizację oraz ujawniający sposób wykorzystania różnych pól adresowych do ustalania trafień i chybień. Gdzie w pamięci podręcznej jest odwzorowane słowo z lokacji pamięci ABCDE8F8?
Źródło: [ALEX93].
- 4.9. Dane są następujące własności zewnętrznej pamięci podręcznej: czterodroźna sekcyjno-skojarzeniowa, rozmiar linii - dwa słowa 16-bitowe, może przyjąć 4 K słów 32-bitowych z pamięci głównej, wykorzystywana z procesorem 16-bitowym, który generuje adresy 24-bitowe. Zaprojektuj strukturę pamięci podręcznej ze wszystkimi niezbędnymi informacjami i pokaż, jak interpretuje ona adresy z procesora.
Źródło: [ALEX93].
- 4.10. Intel 80486 ma wewnętrzną, zintegrowaną w mikroukładzie pamięć podręczną. Zawiera ona 16 KB danych, ma czterodroźną organizację sekcyjno-skojarzeniową, a długość bloku jest równa czterem słowom 32-bitowym. Pamięć ta jest zorganizowana w postaci 128 sekcji. Istnieje pojedynczy „bit ważności wiersza” oraz trzy bity „LRU” (BO, B1 i B2) przypisane każdemu wierszowi. W przypadku chybienia w pamięci podręcznej procesor 80486 odczytuje 16-bajtowy wiersz z pamięci głównej, kierując go do pakietu odczytu magistrali pamięci. Narysuj uproszczony schemat pamięci podręcznej i pokaż, jak są interpretowane różne pola adresu.
- 4.11. Algorytm wymiany procesora Intel 80486 jest określany jako „pseudo-LRU”. Z każdą ze 128 sekcji zawierających po cztery wiersze (oznaczone jako L1, L2, L3 i L4) są skojarzone trzy bity: BO, B1 i B2. Algorytm wymiany działa następująco. Gdy wiersz musi być wymieniony, pamięć podręczna określa najpierw, czy ostatnie wykorzystanie dotyczyło L1, L2, L3 czy też L4. Następnie określa, która para bloków była najdawniej wykorzystywana i znakuje ją jako przeznaczoną do wymiany.
- Podaj sposób ustawiania bitów BO, B1 i B2 oraz sposób ich wykorzystania w algorytmie wymiany.
 - Pokaż, że algorytm wykorzystywany przez procesor 80486 stanowi przybliżenie prawdziwego algorytmu LRU.
 - Wykaż, że prawdziwy algorytm LRU wymaga 6 bitów na sekcję.
- 4.12. Blok podręcznej pamięci sekcyjno-skojarzeniowej ma rozmiar 4 słów 16-bitowych, a sekcja ma rozmiar 2 bloków. Pamięć ta może przyjąć 4096

słów. Pamięć główna współpracująca z tą pamięcią ma pojemność $64\text{ K} \times 32\text{ bit}$. Zaprojektuj strukturę pamięci podręcznej i pokaż, jak są interpretowane adresy z procesora.

Źródło: [ALEX93].

- 4.13. Uogólnij równania 4.1 i 4.2 z dodatku 4A tak, aby odnosiły się do N -poziomowych hierarchii pamięciowych.
- 4.14. System komputerowy zawiera pamięć główną o pojemności 32 K słów 16 -bitowych. Ma także pamięć podręczną 4 K słów podzieloną na 4 -wierszowe sekcje z 64 słowami w każdym wierszu. Załóż, że pamięć podręczna jest na początku pusta. Procesor pobiera słowa z lokacji $0, 1, 2, \dots, 4351$ w tym właśnie porządku. Następnie powtarza tę sekwencję pobierania jeszcze 9 razy. Pamięć podręczna jest 10 razy szybsza niż pamięć główna. Oszacuj korzyść wynikającą z zastosowania pamięci podręcznej. Przyjmij, że wymiana bloków jest zgodna z algorytmem LRU.
- 4.15. Opisz prostą metodę implementacji algorytmu wymiany LRU w cztero-drożnej pamięci podręcznej sekcyjno-skojarzeniowej.
- 4.16. Rozważ system pamięciowy o następujących parametrach*:

$$T_c = 100\text{ ns} \quad C_c = 0,01\text{ centa/bit}$$

$$T_m = 1200\text{ ns} \quad C_m = 0,001\text{ centa/bit}$$

- (a) Jaki jest koszt pamięci głównej 1 MB ?
- (b) Jaki jest koszt pamięci głównej 1 MB wykorzystującej pamięć podręczną?
- (c) Jeśli efektywny czas dostępu jest o 10% większy niż czas dostępu pamięci podręcznej, to ile wynosi współczynnik trafień H ?

DODATEK 4A

Charakterystyka wydajności pamięci dwupoziomowych

W rozdziale tym omówiliśmy pamięć podręczną działającą jako bufor między pamięcią główną a procesorem. Mamy tu do czynienia z dwupoziomową pamięcią wewnętrzną. Architektura taka umożliwia poprawienie wydajności w stosunku do porównywalnej pamięci jednopoziomowej przez wykorzystanie właściwości zwanej lokalnością (tę właściwość przeanalizujemy poniżej).

Mechanizm pamięci podręcznej stanowi część architektury komputera zrealizowanej sprzętowo i jest zwykle niewidoczny dla systemu operacyjnego. Występują ponadto dwa inne przypadki pamięci dwupoziomowych, które również wykorzystują lokalność i które - przynajmniej częściowo - są wdrażane poprzez system operacyjny: pamięć wirtualna i dyskowa pamięć podręczna (tab. 4.8). Pamięć wirtualna jest opisana w rozdz. 7; dyskowa pamięć podręczna wykracza

* Indeks c odnosi się do pamięci podręcznej, indeks m - do głównej (przyp.tłum.).

poza zakres tej książki, jest jednak przedstawiona w [STAL95]. W tym dodatku rozpatrzmy wybrane charakterystyki wydajnościowe pamięci dwupoziomowych, które są wspólne dla wszystkich trzech przypadków.

TABELA 4.8. Własności pamięci dwupoziomowych

	Pamięć podręczna pamięci głównej	Pamięć wirtualna (stronicowana)	Dyskowa pamięć podręczna
Typowy stosunek czasów dostępu	5/1	1000/1	1000/1
System zarządzania pamięcią	wdrażany układowo	kombinacja układowego i programowego	wdrażany w postaci oprogramowania systemowego
Typowy rozmiar bloku	4÷128 B	64÷4096 B	64÷4096 B
Dostęp procesora do drugiego poziomu	bezpośredni	pośredni	pośredni

4A.1. Lokalność

Podstawą zwiększonej wydajności pamięci dwupoziomowej jest zasada znana jako **lokalność odniesienia** [DENN68]. Zgodnie z tą zasadą odniesienia do pamięci wykazują tendencję do grupowania się. W ciągu długiego czasu wykorzystywane ugrupowania zmieniają się, jednak w krótkim czasie procesor przede wszystkim pracuje z ustalonymi ugrupowaniami odniesień do pamięci.

Z intuicyjnego punktu widzenia zasada lokalności ma sens. Rozważmy następujący sposób uzasadnienia:

1. Z wyjątkiem rozkazów skoku i wywołania, które stanowią małą część wszystkich rozkazów programu, realizacja programu ma charakter sekwencyjny. Tak więc, w większości przypadków następny rozkaz przewidziany do pobrania następuje bezpośrednio po ostatnio pobranym rozkazie.
2. Rzadkością jest występowanie długich, nieprzerwanych sekwencji wywołań procedury, po których następowałyby odpowiednie sekwencje powrotów. Program utrzymuje się raczej w wąskim zakresie głębokości wywołań procedury. Tak więc, w ciągu krótkiego czasu odwołania do rozkazów wykazują tendencję do pozostawiania w obrębie niewielu procedur.
3. Większość tworów iteracyjnych składa się ze względnie małej liczby wielokrotnie powtarzanych rozkazów. Podczas iteracji obliczenia pozostają ograniczone do niewielkiej, zwartej części programu.
4. W wielu programach znaczna część obliczeń obejmuje przetwarzanie struktur danych, takich jak tablice lub szeregi rekordów. W wielu przypadkach kolejne odniesienia do tych struktur danych będą dotyczyły zbiorów położonych w sąsiedztwie.

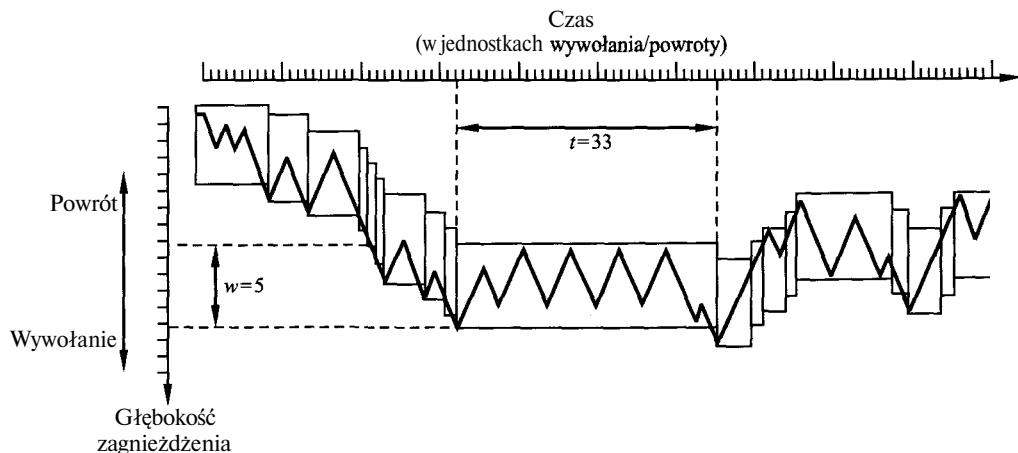
Ten sposób rozumowania *znalazł* potwierdzenie w wielu badaniach. Rozpatrzmy na przykład punkt 1. Przeprowadzono wiele badań, których celem było

przeanalizowanie działania programów wyrażonych w języku wysokiego poziomu. W tabeli 4.9 są przedstawione ich główne wyniki, z uwzględnieniem częstości występowania różnych typów instrukcji podczas wykonywania programów. Pochodzą one z niżej przedstawionych badań.

TABELA 4.9. Względna dynamiczna częstość występowania operacji języka wysokiego poziomu

Badanie	[HUCK83]	[KNUT71]	[P82a]		[TANE78]
Język	Pascal	FORTTRAN	Pascal	C	SAL
Zadania	naukowe	studenckie	systemowe	systemowe	systemowe
Przypisania	74	67	45	38	42
Pętle	4	3	5	3	4
Wywołania	1	3	15	12	12
IF	20	11	29	43	36
GO TO	2	9	-	3	-
Inne	-	7	6	1	6

W najwcześniejszym badaniu funkcjonowania języków programowania, przeprowadzonym przez Knutha [KNUT71], przeanalizowano zbiór programów w FORTRAN-ie wykorzystywanych jako ćwiczenia dla studentów. Tanenbaum [TANE78] opublikował pomiary ponad 300 procedur wykorzystywanych w programach systemów operacyjnych, napisanych w języku umożliwiającym programowanie strukturalne (SAL). Patterson i Sequin [PATT82a] przeanalizowali zestaw pomiarów dokonanych na kompilatorach, programach do składania tekstu, CAD, sortowania i porównywania plików. Zbadano języki programowania C i Pascal. Huck [HUCK83] przeanalizował cztery programy reprezentujące zbiór obliczeń naukowych o ogólnym przeznaczeniu, w tym szybkie transformacje Fouriera i całkowanie układów równań różniczkowych. Wyniki tych badań



RYSUNEK 4.31. Przebieg wywołań i powrotów w programach

prowadzą do zgodnego wniosku, że rozkazy skoku i wywołania stanowią tylko część instrukcji wykonywanych w czasie życia programu. Badania te stanowią więc potwierdzenie pierwszego z wyżej wymienionych twierdzeń.

Drugie twierdzenie znajduje uzasadnienie w badaniach przedstawionych w [PATT85a]. Jest to zilustrowane na rys. 4.31, na którym są pokazane procesy wywołanie-powrót. Każde wywołanie jest reprezentowane przez linię poprowadzoną w dół i na prawo, a każdy powrót przez linię poprowadzoną do góry i na prawo. Na rysunku zdefiniowano okno o głębokości $w = 5$. Jedynie sekwencja wywołań i powrotów powodująca efektywne przesunięcie w jakimkolwiek kierunku przekraczające 6 sprawia, że okno ulega przesunięciu. Jak można zauważyć, wykonywanie programu może pozostawać w ramach stacjonarnego okna przez całkiem długi czas. Badania tej samej grupy programów w C i Pascalu wykazały, że okno o głębokości 8 wymaga przesunięcia zaledwie w przypadku mniej niż 1% wywołań lub powrotów [TAMI83].

4A.2. Działanie pamięci dwupoziomowej

Własność lokalności może być wykorzystana przy budowie pamięci dwupoziomowych. Pamięć wyższego poziomu (M1) jest mniejsza, szybsza i droższa (na bit) niż pamięć niższego poziomu (M2). Pamięć M1 jest wykorzystywana do tymczasowego przechowywania części zawartości większej pamięci M2. Gdy następuje odniesienie do pamięci, przeprowadzana jest próba znalezienia poszukiwanych danych w pamięci M1. Jeśli się ona powiedzie, wykonywany jest szybki dostęp. Jeśli nie, to odpowiedni blok pamięci M2 jest kopiowany do pamięci M1, a dostęp następuje za pośrednictwem pamięci M1. Ze względu na lokalność, jeśli blok został już doprowadzony do pamięci M1, powinno nastąpić wiele odniesień do tego bloku, co powoduje przyspieszenie obsługi.

Aby określić średni czas dostępu do obiektu, musimy rozważyć nie tylko szybkości obu poziomów pamięci, ale także prawdopodobieństwo, że poszukiwane dane mogą być znalezione w pamięci M1. Prawdopodobieństwo to jest znane jako współczynnik trafień. Mamy:

$$\begin{aligned} T_s &= H \times T_1 + (1 - H) \times (T_1 + T_2) = \\ &= T_1 + (1 - H) \times T_2 \end{aligned} \quad (4.1)$$

gdzie:

T_s - średni (systemowy) czas dostępu,

T_1 - czas dostępu do pamięci M1 (np. pamięci podręcznej, w tym dyskowej),

T_2 - czas dostępu do pamięci M2 (np. pamięci głównej, pamięci dyskowej),

H - współczynnik trafień (ułamek czasu, w którym odniesienia są odnajdywane w pamięci M1).

Na rysunku 4.2 jest pokazany średni czas dostępu jako funkcja współczynnika trafień. Jak można zauważyć, przy wysokim procencie trafień średni całkowity czas dostępu jest o wiele bliższy czasowi dostępu do pamięci M1 niż do M2.

4A.3. Wydajność

Popatrzmy na pewne parametry istotne przy ocenie funkcjonowania pamięci dwupoziomowej. W pierwszej kolejności rozważmy koszt. Mamy:

$$C_s = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2}$$

gdzie:

C_s - średni koszt na bit pamięci dwupoziomowej,

C_1 - średni koszt na bit pamięci wyższego poziomu - M1,

C_2 - średni koszt na bit pamięci niższego poziomu - M2,

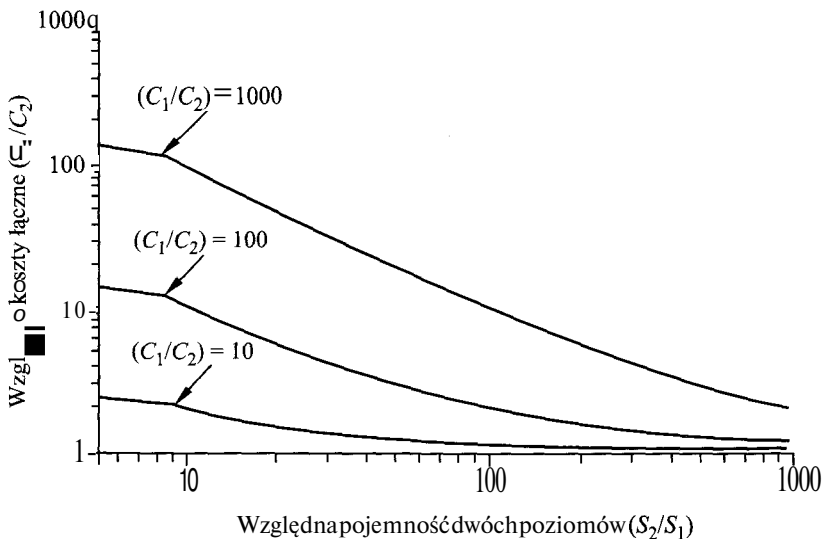
S_1 - rozmiar pamięci M1,

S_2 - rozmiar pamięci M2.

Chcielibyśmy, aby $C_s = C_2$. Ponieważ $C_1 \gg C_2$, wymagane jest $S_1 \ll S_2$. Zależność tę widać na rys. 4.32.

Rozważmy następnie czas dostępu. Żeby osiągnąć znaczącą poprawę wydajności przez wprowadzenie pamięci dwupoziomowej, trzeba osiągnąć T_s równe w przybliżeniu T_1 ($T_s \sim T_1$). Ponieważ T_1 jest o wiele mniejszy niż T_2 ($T_1 \ll T_2$), wymagany jest współczynnik trafień bliski 1.

Tak więc chcielibyśmy, żeby pamięć M1 była mała w celu ograniczenia kosztów, a jednocześnie żeby była duża w celu poprawienia współczynnika trafień i przez to wydajności. Czy istnieje rozmiar pamięci M1 satysfakcjonujący oba wymagania w rozsądnym stopniu? Możemy odpowiedzieć na to pytanie, posługując się szeregiem pytań pomocniczych:



RYSUNEK 4.32. Zależność między przeciętnym kosztem pamięci a względną pojemnością w przypadku pamięci dwupoziomowej

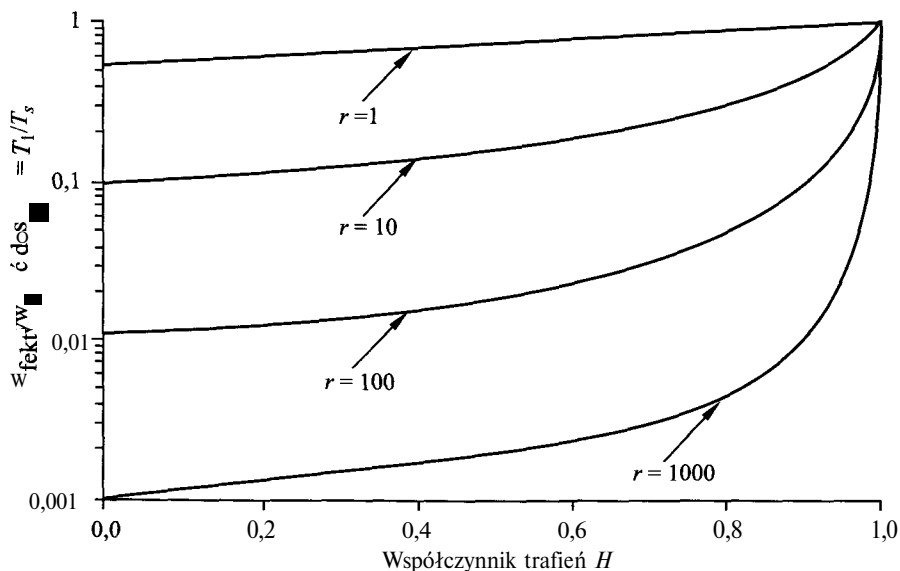
- a Jaka wartość współczynnika trafień jest wymagana do osiągnięcia pożądanej wydajności?
- a Jaki rozmiar pamięci M1 zapewni wymagany współczynnik trafień?
- a Czy przy tym rozmiarze będzie utrzymany dopuszczalny poziom kosztów?

Żeby na nie odpowiedzieć, rozważmy wielkość T_1/T_s , określaną jako **efektywność dostępu**. Jest ona miarą tego, jak bliski jest średni czas dostępu (T_s) czasowi dostępu do pamięci M1 (T_1). Na podstawie równania (4.1) możemy napisać:

$$\frac{T_1}{T_s} = \frac{1}{1 + (1-H) \times \left\{ \frac{T_2}{T_1} \right\}}$$

Na rysunku 4.33 widać wykres T_1/T_s w funkcji współczynnika trafień H , przy czym parametrem jest wielkość T_2/T_1 . Zwykle czas dostępu do pamięci podręcznej jest około 5 razy krótszy od czasu dostępu do pamięci głównej (tzn. T_2/T_1 wynosi 5 do 10), a czas dostępu do pamięci głównej jest około 1000 razy krótszy niż czas dostępu do pamięci dyskowej ($T_2/T_1 = 1000$). Tak więc, wymagany wzrost wydajności będzie osiągnięty, jeśli współczynnik trafień będzie równy 0,8 do 0,9.

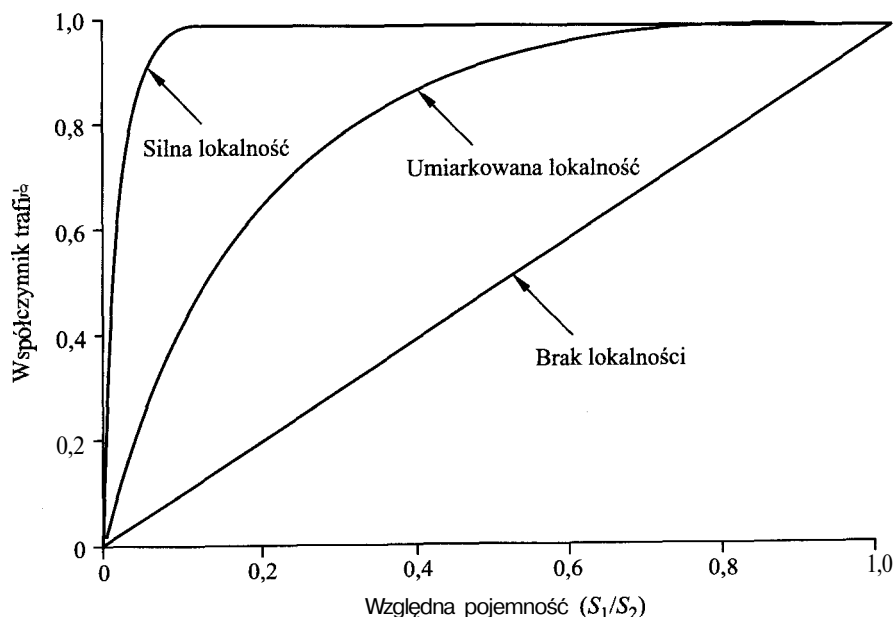
Zajmiemy się teraz dokładniej kwestią względnych rozmiarów pamięci. Czy współczynnik trafień wynoszący 0,8 lub więcej jest wielkością rozsądną przy $S_1 \ll S_2$? Będzie to zależało od wielu czynników, włącznie z naturą wykorzystywanego oprogramowania oraz ze szczegółami projektowymi pamięci dwupo-



RYСУNEK 4.33. Efektywność dostępu jako funkcja H ($r = T_2/T_1$)

ziomowej. Główną determinantą jest oczywiście stopień lokalności. Rysunek 4.34 ilustruje wpływ lokalności na współczynnik trafień. Oczywiście, jeżeli pamięć M1 ma taki sam rozmiar jak M2, to współczynnik trafień wyniesie 1,0: cała zawartość pamięci M2 jest zawsze przechowywana również w pamięci M1. Załóżmy teraz, że lokalność nie występuje; oznacza to, że odniesienia są całkowicie przypadkowe. Wówczas współczynnik trafień powinien być ściśle liniową funkcją względnego rozmiaru pamięci. Jeśli na przykład pamięć M1 jest o połowę mniejsza niż M2, to w dowolnej chwili połowa zawartości pamięci M2 znajduje się również w M1, a współczynnik trafień wyniesie 0,5. W praktyce jednak odniesienia wykazują pewien stopień lokalności. Wpływ umiarkowanej i silnej lokalności jest pokazany na rysunku.

Jeśli więc występuje silna lokalność, to możliwe jest osiągnięcie dużych wartości współczynnika trafień, nawet przy względnie małej pamięci wyższego poziomu. W licznych badaniach wykazano na przykład, że raczej małe rozmiary pamięci podręcznej umożliwiają uzyskanie współczynnika trafień ponad 0,75 *niezależnie od rozmiaru pamięci głównej* (patrz np. [AGAR89], [PRZY88], [STRE83] i [SMIT82]). Pamięć podręczna o pojemności od 1 K do 512 K słów jest na ogół wystarczająca, natomiast pojemność pamięci głównej wynosi zwykle wiele MB. Gdy będziemy rozpatrywać pamięć wirtualną i dyskową pamięć podręczną, zacytujemy inne badania potwierdzające to samo zjawisko: względnie mała pamięć M1 z powodu lokalności umożliwia uzyskanie dużego współczynnika trafień.



RYSUNEK 4.34. Współczynnik trafień jako funkcja względnej pojemności pamięci

Dochodzimy w ten sposób do ostatniego z wcześniej postawionych pytań: czy względny rozmiar obu pamięci pozwala na utrzymanie dopuszczalnych kosztów? Odpowiedź jest oczywiście twierdząca. Jeśli potrzebujemy tylko względnie małej pamięci wyższego poziomu, aby osiągnąć dobrą wydajność, to średni koszt na bit obu poziomów pamięci zbliży się do tańszej pamięci niższego poziomu.

Pamięć zewnętrzna

W tym rozdziale zajmiemy się urządzeniami i systemami pamięci zewnętrznej. Rozpocznijemy od najważniejszego urządzenia, jakim jest dysk magnetyczny. Dyski magnetyczne stanowią podstawę pamięci zewnętrznej, praktycznie wszystkich systemów komputerowych. W następnym podrozdziale przeanalizujemy wykorzystanie tablic dysków w celu osiągnięcia większej wydajności, zwracając szczególną uwagę na rodzinę systemów zwanych RAID (*Redundant Array of Independent Disks*). Składnikiem wielu systemów komputerowych, którego znaczenie wzrasta, jest zewnętrzna pamięć optyczna - zajmiemy się nią w podrozdz. 5.3. Na zakończenie opiszemy magnetyczną pamięć taśmową.

5.1

DYSK MAGNETYCZNY

Dysk to okrągła płyta wykonana z metalu lub plastiku, pokryta materiałem magnetycznym. Dane są zapisywane a następnie odczytywane z dysku za pomocą przewodzącej cewki zwanej **głowicą**. Podczas operacji odczytu lub zapisu głowica pozostaje nieruchoma, natomiast obraca się dysk.

Zapis polega na wykorzystaniu pola magnetycznego wytwarzanego przez prąd elektryczny płynący przez cewkę. Do głowicy są wysyłane impulsy, co powoduje zapisywanie wzorów magnetycznych na powierzchni znajdującej się pod głowicą, przy czym wzory te są różne dla prądów dodatnich i ujemnych. Przy odczycie wykorzystuje się prąd elektryczny powstający w cewce pod wpływem pola magnetycznego dysku poruszającego się względem cewki. Gdy powierzchnia dysku przesuwana się pod głowicą, generowany jest prąd o tej samej biegunowości, jak użyty do zapisu.

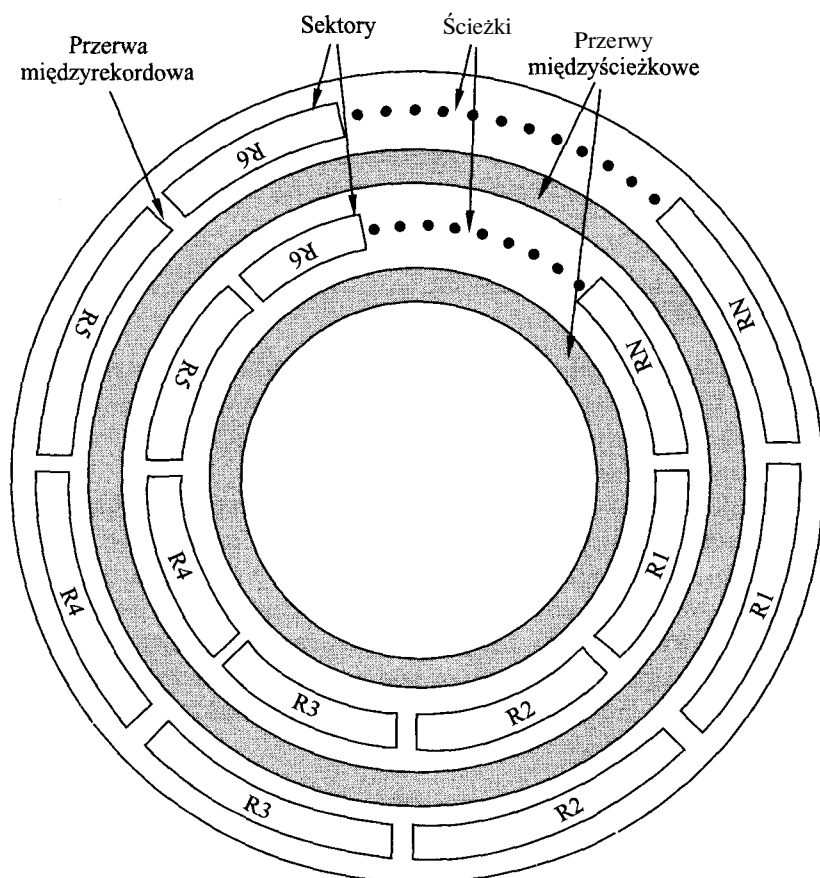
Organizacja i formatowanie danych

Głowica jest względnie małym przyrządem umożliwiającym odczytywanie lub zapisywanie z części płyty obracającej się pod nią. To właśnie sprawiło, że organizacja danych na płycie ma postać koncentrycznego zespołu pierś-

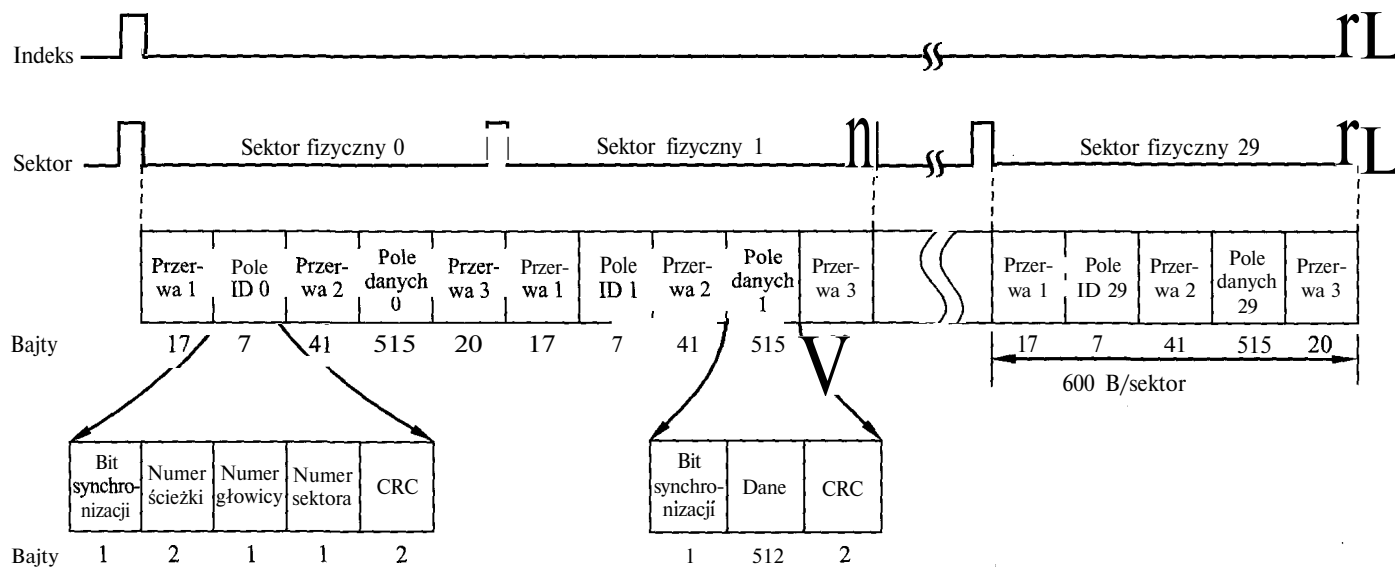
cieni, nazywanych **ścieżkami**. Każda ścieżka ma taką samą szerokość, jak głowica.

Ten rozkład danych jest przedstawiony na rys. 5.1. Sąsiednie ścieżki są oddzielone **przerwami**. Zapobiega to, a przynajmniej minimalizuje błędy spowodowane przez niewłaściwe ustawienie głowicy lub po prostu interferencję pola magnetycznego. W celu uproszczenia układów elektronicznych na każdej ścieżce jest przechowywana taka sama liczba bitów. Tak więc **gęstość**, wyrażana w bitach na cal, wzrasta w miarę przesuwania się od ścieżki zewnętrznej do wewnętrznej (to samo zjawisko występuje na płytach fonograficznych).

Jak zauważyliśmy wcześniej, dane są przenoszone na dysk i z dysku blokami. Zwykle blok jest mniejszy niż pojemność ścieżki. Wobec tego dane są przechowywane w obszarach o rozmiarach odpowiadających blokom, zwanych **sektorami** (rys. 5.1). Na ścieżkę przypada zwykle od 10 do 100 sektorów, mogą one przy tym mieć długość ustaloną lub zmienną. W celu zapobieżenia nadmier-



RYSUNEK 5.1. Rozkład danych na dysku



RYSUNEK 5.2. Format ścieżki dysku typu Winchester (Seagate ST506)

nie wysokim wymaganiom dotyczącym precyzji systemu, sąsiednie sektory są oddzielone przerwami wewnątrzścieżkowymi (międzyrekordowymi).

Jak są identyfikowane pozycje sektora wewnątrz ścieżki? Musi oczywiście istnieć pewien punkt startowy na ścieżce oraz sposób identyfikowania początku i końca każdego sektora. Wymagania te są spełniane za pomocą danych kontrolnych zapisanych na dysku. Dysk jest więc formatowany za pomocą dodatkowych danych wykorzystywanych tylko przez napęd dysku i niedostępnych dla użytkownika.

Przykład formatowania dysku jest pokazany na rys. 5.2. W tym przypadku każda ścieżka zawiera 30 sektorów o ustalonej długości, po 600 bajtów każdy. Każdy sektor mieści 512 bajtów danych oraz informacje kontrolne wykorzystywane przez sterownik dysku. Pole ID jest unikatowym (jednoznacznym) identyfikatorem lub adresem wykorzystywanym do lokalizowania określonego sektora. Bajt SYNCH jest specjalnym wzorem bitowym wyznaczającym początek pola. Numer ścieżki identyfikuje ścieżkę na powierzchni. Numer głowicy identyfikuje głowicę, ponieważ dysk ma zwykle wiele powierzchni (będzie to wyjaśnione w dalszym ciągu). Zarówno pola ID, jak i pola danych zawierają kod służący do wykrywania błędów.

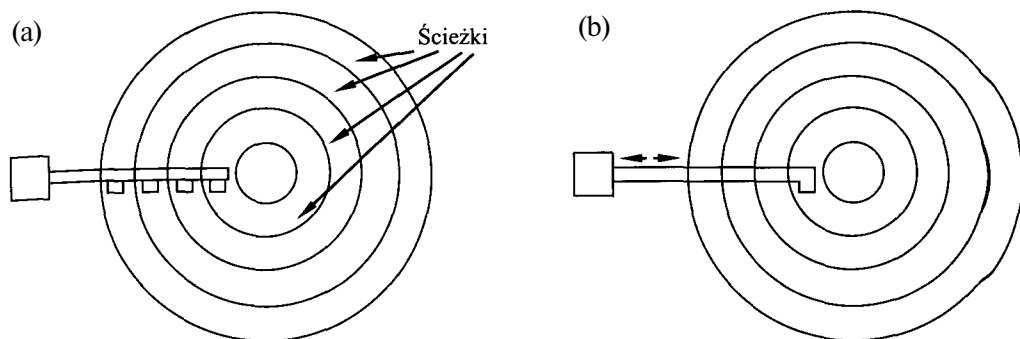
Własności

W tabeli 5.1 są wymienione główne własności, które umożliwiają różnicowanie poszczególnych rodzajów dysków. Po pierwsze, głowica może być nieruchoma lub poruszać się wzdłuż promienia płyty. W przypadku dysków z *nieruchomą głowicą*, występuje jedna głowica zapisu/odczytu na jedną ścieżkę. Wszystkie głowice są zmontowane na sztywnym ramieniu, które rozciąga się przez wszystkie ścieżki (rys. 5.3a). W przypadku dysków z *ruchomą głowicą*, istnieje tylko jedna głowica zapisu/odczytu (rys. 5.3b). Jak poprzednio, głowica jest umocowana na ramieniu. Ponieważ jednak musi istnieć możliwość pozycjonowania głowicy nad dowolną ścieżką, ramię może być w tym celu wydłużane lub skracane.

Sam dysk jest umocowany w napędzie dysku, który składa się z ramienia, wałka obracającego dysk i z układów elektronicznych potrzebnych do wprowadzania i wyprowadzania danych binarnych. Dysk **niewymienny** jest na stałe

TABELA 5.1. Własności systemów dyskowych

Ruch głowicy	Liczba dysków
głowica nieruchoma (jedna na ścieżkę)	jednodyskowy
głowica ruchoma (jedna na powierzchnię)	wielodyskowy
Wymiennosc dysku	Mechanizm głowicy
niewymienny	kontaktowy (floppy)
wymienny	ustalona przerwa
Wykorzystanie stron	przerwa aerodynamiczna (Winchester)
jednostronny	
dwustronny	



RYSUNEK 5.3. Stałe (a) i ruchome (b) głowice dysku

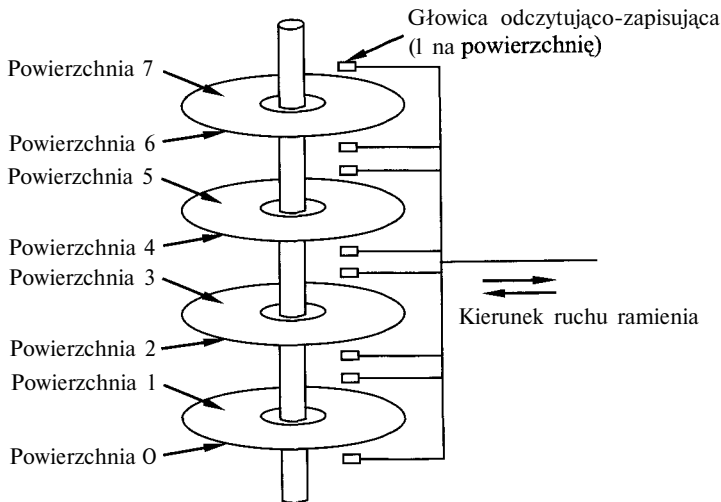
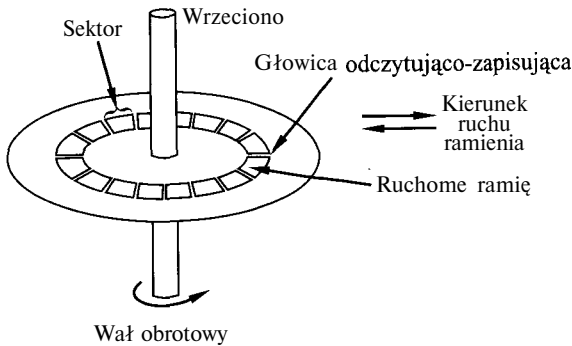
mocowany w napędzie dysku. Dysk **wymienny** (ang. *removable*) może być usunięty i zastąpiony innym dyskiem. Zaletą tego ostatniego jest nieograniczona ilość danych osiągalna przy ograniczonej liczbie systemów dyskowych. Ponadto dysk taki może być przenoszony z jednego systemu komputerowego do innego.

W przypadku większości dysków warstwa magnetyczna jest nanoszona po obu stronach płyty, co jest określane jako **dysk dwustronny**. Niektóre tańsze systemy dyskowe wykorzystują **dyski jednostronne**.

Niektóre napędy dysków umożliwiają stosowanie wielu płyt ustawionych pionowo w odległościach około 2,5 cm (rys. 5.4). Wykorzystuje się wówczas wiele ramion. Takie urządzenie jest nazywane **pakiem dysków**.

Wreszcie, mechanizm głowicy stanowi podstawę do klarownej klasyfikacji dysków na trzy rodzaje. Tradycyjnie głowica zapisująco-odczytująca była umieszczana w ustalonej odległości nad płytą, przy czym była pozostawiona przerwa powietrzna. Krańcowo różniące się rozwiązanie mechanizmu głowicy polega na pozostawieniu jej w fizycznym kontakcie z płytą podczas operacji odczytu lub zapisu. Mechanizm ten jest stosowany w przypadku napędu **dyskielek**, które są niewielkimi, elastycznymi płytkami i stanowią najtańszy rodzaj dysków magnetycznych.

Aby zrozumieć powód powstania trzeciego rodzaju dysków, musimy skomentować zależność między gęstością danych a rozmiarem przerwy powietrznej. Głowica musi generować lub wykrywać pole elektromagnetyczne o wielkości wystarczającej do zapisu i odczytu. Im węższa jest głowica, tym bardziej musi być zbliżona do powierzchni płyty, aby urządzenie działało. Ponieważ węższa głowica oznacza węższe ścieżki i dzięki temu większą gęstość danych, jest to pożądane. Jednak im bliżej znajduje się głowica w stosunku do dysku, tym większe jest ryzyko błędu spowodowanego przez zanieczyszczenia lub niedokładności. W wyniku ulepszenia technologii wprowadzono dysk typu Winchester. Głowice dysku Winchester pracują w zamkniętych zespołach napędowych, które są niemal wolne od zanieczyszczeń. Zostały zaprojektowane do działania w mniejszej odległości od powierzchni dysków w porównaniu z konwencjonalnymi głowicami dysków, co umożliwia większą gęstość upakowania



RYSUNEK 5.4. Pakiet dysków

danych. Głowica ta to aerodynamiczny pasek folii, spoczywający lekko na powierzchni płyty w czasie, gdy dysk jest nieruchomy. Ciśnienie powietrza generowane przez wirujący dysk wystarcza, aby spowodować uniesienie się folii nad powierzchnią. Dzięki temu można wykorzystać węższe głowice, które pracują bliżej powierzchni płyty niż konwencjonalne sztywne głowice dysków.

Historycznie rzecz biorąc, termin *Winchester* został po raz pierwszy użyty przez IBM jako nazwa kodowa modelu dysków 3340 przed jego zaanonsovaniem. Model 3340 był wymiennym pakietem dysków z głowicami zamkniętymi wewnątrz pakietu. Termin ten jest obecnie stosowany w odniesieniu do dowolnego zamkniętego napędu dysków wykorzystującego głowice aerodynamiczne.

Czas dostępu do dysków

Podczas pracy dysk obraca się ze stałą prędkością. W celu zapisu lub odczytu głowica musi być ustawiona nad pożądaną ścieżką i na początku pożądanego sektora na tej ścieżce. Wybór ścieżki polega na przesunięciu głowicy w przypadku systemu z ruchomą głowicą lub na elektronicznym wyborze jednej głowicy w systemie z nieruchomymi głowicami. W systemie z ruchomą głowicą czas pozycjonowania głowicy nad ścieżką nosi nazwę **czasu przeszukiwania** (ang. *seek time*). W każdym przypadku po dokonaniu wyboru ścieżki system czeka, aż odpowiedni sektor znajdzie się pod głowicą. Czas osiągnięcia głowicy przez sektor jest nazywany **opóźnieniem obrotowym** (ang. *rotational latency*). Suma czasu przeszukiwania, jeśli taki występuje, oraz opóźnienia obrotowego nazywa się **czasem dostępu** - czasem, który jest wymagany do osiągnięcia stanu umożliwiającego odczyt lub zapis. Gdy głowica znajduje się już na właściwej pozycji, operacja odczytu lub zapisu jest dokonywana w miarę przemieszczania się sektora pod głowicą.

52

PAMIĘĆ RAID

Jak mówiliśmy wcześniej, wydajność pamięci pomocniczych rosła wyraźnie wolniej niż wydajność procesorów i pamięci głównych. To niedopasowanie spowodowało, że system pamięci dyskowych stał się być może głównym ogniskiem zainteresowania w pracach nad poprawą ogólnej wydajności systemów komputerowych.

Podobnie jak w przypadku pozostałych obszarów wydajności komputera, projektanci pamięci dyskowych stwierdzili, że przy ograniczonej możliwości ulepszenia pojedynczych składników dodatkowe zwiększenie wydajności można osiągnąć, wykorzystując równolegle wiele składników. W przypadku pamięci dyskowej doprowadziło to do opracowania tablic dysków, które pracują niezależnie i równolegle. Gdy mamy do czynienia z wieloma dyskami, oddzielne zapotrzebowania na operacje wejścia-wyjścia mogą być przetwarzane równolegle, jeśli tylko poszukiwany blok danych jest rozproszony na wielu dyskach.

W przypadku używania wielu dysków istnieje wiele sposobów organizowania danych i poprawy niezawodności przez wykorzystanie nadmiarowości (redundancji). Mogłoby to utrudnić opracowanie schematów baz danych, które mogłyby być używane na wielu platformach sprzętowych i z różnymi systemami operacyjnymi. Na szczęście w przemyśle uzgodniono znormalizowany schemat projektowania baz danych dla pamięci wielodyskowych, znany jako redundancyjna tablica niezależnych dysków - RAID (ang. *Redundant Array of Independent Disks*). Schemat RAID składa się z sześciu poziomów¹, od zerowego do

¹ Niektórzy badacze i niektóre firmy wprowadziły dodatkowe poziomy, jednak powszechne uzgodnienie obejmuje tylko sześć poziomów opisanych w tym podrozdziale.

piątego. Poziomy te nie implikują zależności hierarchicznej, lecz wyznaczają różne architektury, które mają trzy cechy wspólne:

1. RAID jest zespołem fizycznie istniejących napędów dyskowych widzianych przez system operacyjny jako pojedynczy napęd logiczny.
2. Dane są rozproszone w tych napędach, tworzących tablicę.
3. Redundancyjna pojemność dysków jest wykorzystywana do przechowywania informacji o parzystości, co gwarantuje odzyskiwanie danych w przypadku uszkodzenia dysku.

Szczegółowe rozwiązania dotyczące drugiej i trzeciej cechy są różne dla różnych poziomów RAID. Poziom RAID 0 nie wspiera trzeciej cechy.

Termin RAID został po raz pierwszy zaproponowany w artykule napisanym przez grupę badaczy z University of California w Berkeley². W artykule przedstawiono różne konfiguracje i zastosowania RAID, a także wprowadzono definicje poziomów RAID stosowaną do dzisiaj. RAID zaproponowano w celu zamknięcia poszerzającej się luki między szybkimi procesorami a stosunkowo wolnymi elektromechanicznymi napędami dysków. Strategia ta polega na zastąpieniu napędu dysku o wielkiej pojemności wieloma napędami o mniejszej pojemności oraz na takim rozproszeniu danych, które umożliwia jednoczesny dostęp do danych w wielu napędach, dzięki czemu uzyskuje się zwiększenie wydajności wejścia-wyjścia i łatwiejsze zwiększanie pojemności.

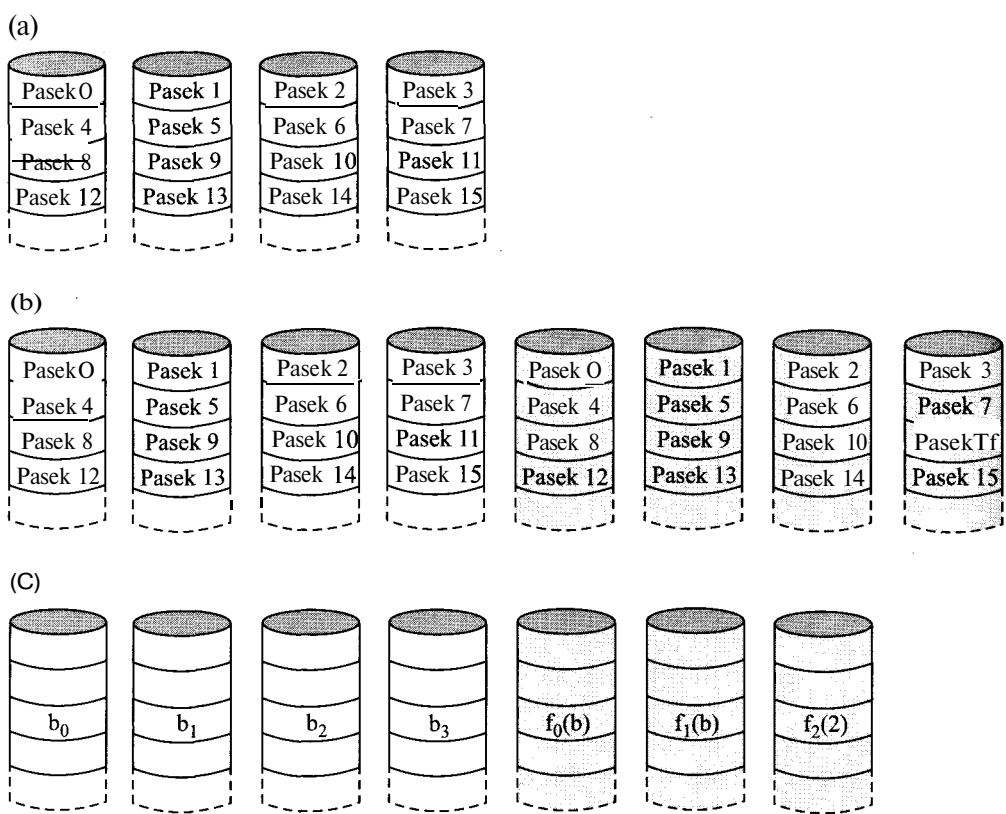
Unikalnym wkładem wniesionym w propozycji RAID jest efektywne zwiększenie niezawodności przez redundancję. Chociaż jednoczesne działanie wielu głowic i urządzeń uruchamiających umożliwiło przyspieszenie działania wejścia-wyjścia, jednak stosowanie wielu podzespołów zwiększa prawdopodobieństwo uszkodzenia. Aby skompensować tę zmniejszoną niezawodność, RAID wykorzystuje informację o parzystości, która pozwala na odtwarzanie danych utraconych z powodu uszkodzenia dysku.

Przeanalizujemy teraz każdy z poziomów RAID. W tabeli 5.2 są podsumowane własności wszystkich sześciu poziomów. Spośród nich poziomy 2 i 4 nie są oferowane komercyjnie, a ich akceptacja w przemyśle nie jest prawdopodobna, chociaż opis tych poziomów pomaga wyjaśnić decyzje projektowe odnoszące się do pozostałych poziomów.

² W artykule tym akronim RAID dotyczył określenia Redundant Array of Inexpensive Disks (redundancyjna matryca niedrogich dysków). Termin „niedrogich” zastosowano dla odróżnienia małych, stosunkowo tanich dysków w matrycy RAID od stanowiących alternatywę, pojedynczych, dużych i drogich dysków (SLED). Dzisiaj SLED należy do przeszłości, a podobna technologia dysków jest stosowana zarówno w konfiguracjach RAID, jak i w pozostałych. Wobec tego w przemyśle przyjęto termin „niezależnych” w celu podkreślenia, że matryca RAID umożliwia znaczącą poprawę wydajności i niezawodności.

TABELA 5.2. Poziomy RAID

Kategoria	Poziom	Opis	Częstość zgłaszania żądania wejścia-wyjścia (odczyt/zapis)	Szybkość transferu danych (odczyt/zapis)	Typowe zastosowanie
Paskowanie	0	bez redundancji	duże paski: doskonała	małe paski: doskonała	zastosowania wymagające wysokiej wydajności w odniesieniu do niekrytycznych danych
Kopiowanie lustrzane	1	lustrzane kopiowanie danych	dobra/dostateczna	dostateczna	napędy systemowe; krytyczne pliki
	2	redundancja poprzez kod Hamminga	mała	doskonała	
	3	parzystość na poziomie bitów	mała	doskonała	zastosowania o dużych rozmiarach zapotrzebowań we-wy, np. obrazowanie, CAD
Dostęp równoległy	4	parzystość na poziomie bloków	doskonała/dostateczna	dostateczna/mała	
	5	rozproszona parzystość na poziomie bloków	doskonała/dostateczna	dostateczna/mała	duża częstość żądań, duża intensywność odczytów, wyszukiwanie danych



RYSUNEK 5.5. Poziomy RAID: (a) RAID 0 (bez redundancji); (b) RAID 1 (zwierciadlany); (c) RAID 2 (redundancja z wykorzystaniem kodu Hamminga)

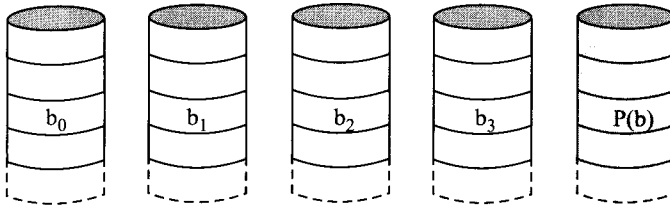
Na rysunku 5.5 widać sześć schematów RAID, które dotyczą pojemności wymagających czterech dysków bez redundancji. Na rysunku są pokazane: rozkład danych użytkownika i danych redundancyjnych, a także względne wymagania dotyczące przechowywania na różnych poziomach. Odniesiemy się do tego rysunku w dalszym ciągu.

RAID poziom 0

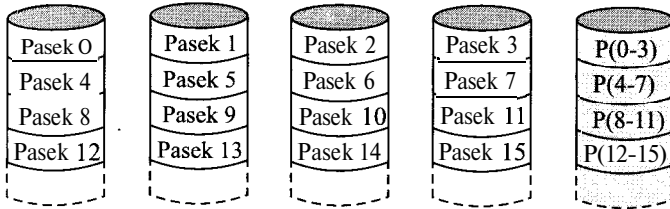
RAID 0 nie powinien być zaliczany do rodziny RAID, ponieważ w celu poprawienia wydajności nie przewiduje on redundancji. Istnieją jednak nieliczne zastosowania (należą do nich niektóre superkomputery), w których wydajność i pojemność są najważniejsze, a niski koszt jest ważniejszy od zwiększonej niezawodności.

W przypadku RAID 0 dane użytkownika i dane systemowe są rozproszone na wszystkich dyskach tablicy. Stanowi to wyraźną korzyść w stosunku do du-

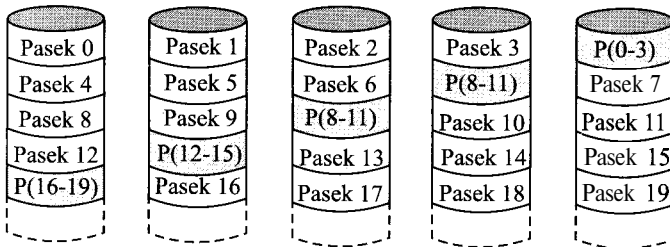
(d)



(e)



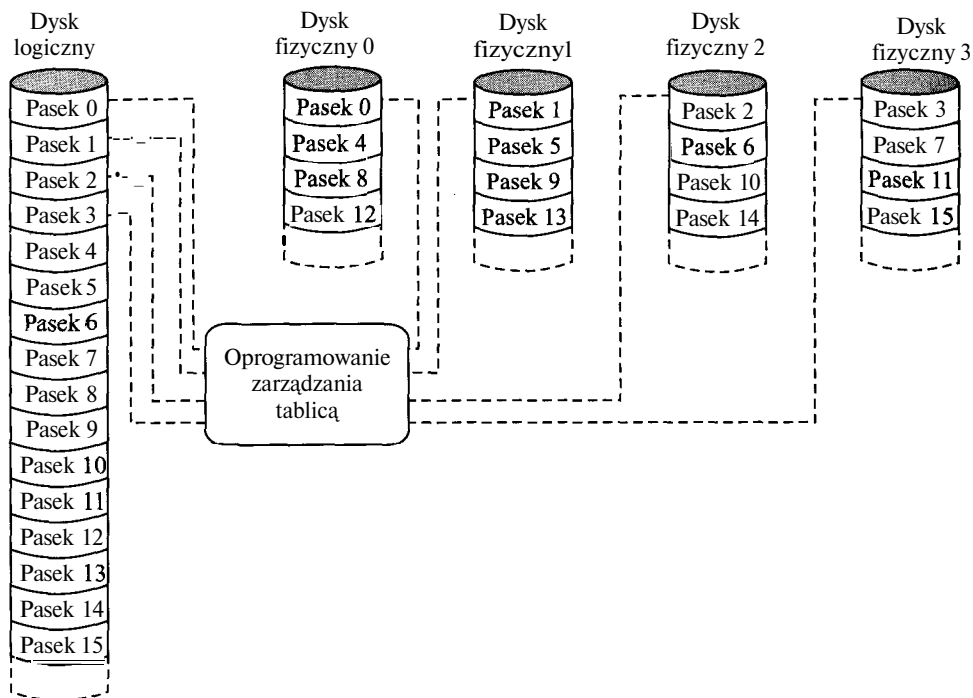
(f)



RYSUNEK 5.5. (ciąg dalszy) (d) RAID 3 (parzystość z przeplataniem bitów); (e) RAID 4 (parzystość na poziomie bloków); (f) RAID 5 (rozproszona parzystość na poziomie bloków)

z tego pojedynczego dysku: jeśli dwa różne żądania wejścia-wyjścia dotyczą dwóch różnych bloków danych, istnieje duże prawdopodobieństwo, że potrzebne bloki znajdują się na różnych dyskach. Dzięki temu oba zapotrzebowania mogą być przetwarzane równolegle, co skraca oczekiwanie w kolejkach.

Jednak RAID 0, podobnie jak wszystkie poziomy RAID, nie ogranicza się do prostego rozproszenia danych w tablicy dysków: dane są układane w postaci pasków (ang. *striped*) na dostępnych dyskach. Można to lepiej wyjaśnić na podstawie rys. 5.6. Wszystkie dane użytkownika i systemowe mogą być postrzegane jako przechowywane na jednym dysku logicznym. Dysk jest podzielony na paski; paski te mogą być fizycznymi blokami, sektorami lub innymi jednostkami. Paski są odwzorowywane cyklicznie na kolejnych dyskach tablicy. W tablicy n -dyskowej pierwszych n logicznych pasków przechowuje się fizycznie jako pierwszy pasek na każdym z n dysków, następnych n pasków mieści się fizycznie w postaci drugich pasków na każdym dysku itd. Zaletą takiego rozkładu jest to, że jeśli pojedyncze zapotrzebowanie wejścia-wyjścia dotyczy wielu logicz-



RYŚUNEK 5.6. Odwzorowanie danych dla poziomu RAID 0 [MASS94]

nie sąsiadujących pasków, to nawet n pasków dotyczących tego zapotrzebowania może być obsługiwanych równolegle, co znacznie redukuje czas transferu wejścia-wyjścia.

Na rysunku 5.6 jest pokazane wykorzystanie oprogramowania zarządzania tablicą w celu odwzorowania między logiczną a fizyczną przestrzenią dysków. Oprogramowanie to może być uruchamiane albo w podsystemie dyskowym, albo w komputerze macierzystym.

Wykorzystywanie RAID 0 do zwiększania wydajności transferu danych

Wydajność dowolnego poziomu RAID krytycznie zależy od rozkładu zapotrzebowania systemu macierzystego oraz od rozmieszczenia danych. Jest to najwyraźniej widoczne w przypadku RAID 0, w którym redundancja nie zaburza analizy. Rozpatrzmy najpierw wykorzystywanie RAID 0 do osiągania dużych szybkości przesyłania danych. W takich zastosowaniach muszą być spełnione dwa wymagania. Po pierwsze, wysoka wydajność transferu danych musi mieć miejsce na całej drodze między pamięcią macierzystą a indywidualnymi napędami dysków. Należy więc uwzględnić wewnętrzne magistrale sterownika, magistrale

wejścia-wyjścia systemu macierzystego, urządzenia dopasowujące wejście-wyjście oraz magistrale pamięci macierzystej.

Drugim wymaganiem jest to, żeby żądania wejścia-wyjścia pochodzące z aplikacji efektywnie kierowały tablicą dysków. Wymaganie to jest spełnione, jeśli typowe żądanie dotyczy takiej ilości logicznie sąsiadujących danych, która jest duża w porównaniu z rozmiarem paska. W takim przypadku, pojedyncze żądanie wejścia-wyjścia wywołuje równoległe przesyłanie danych z wielu dysków, zwiększając efektywną szybkość transferu w porównaniu z pojedynczym dyskiem.

Wykorzystanie RAID 0 do przyspieszenia obsługi żądania wejścia-wyjścia

W środowisku transakcyjnym użytkownik jest zwykle bardziej zainteresowany czasem odpowiedzi niż szybkością transferu. W przypadku indywidualnego żądania wejścia-wyjścia dotyczącego małej ilości danych, czas dostępu wejścia-wyjścia jest zdominowany przez ruch głowic dyskowych (czas przeszukiwania) i ruch dysków (opóźnienie obrotowe).

W środowisku transakcyjnym mogą występować setki żądań wejścia-wyjścia na sekundę. Tablica dysków może zapewnić szybkie odpowiedzi na żądanie dostępu przez rozkładanie obciążenia wejścia-wyjścia na wiele dysków. Efektywne równoważenie obciążenia jest możliwe tylko wtedy, gdy dominują typowe, wielokrotne żądania wejścia-wyjścia. To z kolei sprawia, że może istnieć wiele niezależnych aplikacji lub że pojedyncza aplikacja o charakterze transakcyjnym może powodować wielokrotne, asynchroniczne żądanie wejścia-wyjścia. Na wydajność ma również wpływ rozmiar paska. Jeśli jest on stosunkowo duży, w wyniku czego pojedyncze żądanie wejścia-wyjścia wywołuje tylko pojedynczy dostęp do dysku, to wiele oczekujących żądań wejścia-wyjścia można realizować równoległe, co redukuje czas oczekiwania w kolejce każdego zapotrzebowania.

RAID poziom 1

RAID 1 różni się od poziomów RAID od 2 do 5 sposobem osiągania redundancji. W pozostałych schematach RAID jest wykorzystywana pewna forma obliczeń parzystości w celu wprowadzenia redundancji. Natomiast w przypadku RAID 1 redundancja jest osiągana po prostu przez duplikowanie wszystkich danych. Jak widać na rys. 5.5b, wykorzystywane jest paskowanie danych, podobnie jak w RAID 0. Jednak w tym przypadku każdy pasek logiczny jest odwzorowywany na dwóch oddzielnych dyskach fizycznych, dzięki czemu każdy dysk w tablicy ma swój dysk zwierciadlany zawierający te same dane.

Organizacja RAID 1 ma wiele pozytywnych aspektów:

1. Żądanie odczytu może być obsługiwane przez ten spośród dwóch dysków zawierających potrzebne dane, który wymaga mniejszego czasu przeszukiwania plus opóźnienie obrotowe.
2. Żądanie zapisu wymaga aktualizacji obu odpowiednich pasków, jednak może to być wykonywane równolegle. Wobec tego wydajność zapisu jest dyktowana przez wolniejszy z dwóch zapisów (tzn. ten, który zajmuje dłuższy czas przeszukiwania plus opóźnienie obrotowe). W przypadku RAID 1 nie występuje jednak „kara” związana z zapisem. W RAID 2 do 5 są wykorzystywane bity parzystości. Jeśli więc jest aktualizowany pojedynczy pasek, oprogramowanie zarządzania tablicą musi obliczyć i zaktualizować bity parzystości oprócz aktualizowania samego paska danych.
3. W razie awarii rozwiązanie jest proste. Gdy napęd ulega uszkodzeniu, dane mogą być uzyskane z drugiego napędu.

Główną wadą RAID 1 jest koszt; wymaga on 2-krotnie większej przestrzeni dyskowej w porównaniu z dyskiem logicznym, który obsługuje. Z tego powodu wykorzystanie konfiguracji RAID 1 pozostaje raczej ograniczone do napędów przechowujących oprogramowanie systemowe oraz inne bardzo ważne pliki. W tych przypadkach RAID 1 zapewnia rezerwowe dane dostępne w czasie realnym, dzięki czemu w razie uszkodzenia dysku wszystkie ważne dane są natychmiast dostępne.

W środowisku transakcyjnym RAID 1 umożliwia szybkie uzyskiwanie odpowiedzi na żądania wejścia-wyjścia, jeśli tylko główna część żądań dotyczy odczytu. W takiej sytuacji wydajność RAID 1 może być niemal 2-krotnie większa niż wydajność RAID 0. Jeśli jednak istotna część żądań wejścia-wyjścia dotyczy zapisu, to może nie występować *znaczna* poprawa wydajności w stosunku do RAID 0. RAID 1 może także mieć większą wydajność niż RAID 0 w przypadku zastosowań wymagających intensywnego transferu danych przy znaczącym udziale odczytów. Wzrost wydajności następuje, gdy możliwe jest rozdzielenie każdego żądania odczytu w ten sposób, aby w operacji uczestniczyły oba dyski.

RAID poziom 2

RAID 2 i 3 wykorzystują metodę dostępu równoległego. W przypadku tablicy o dostępie równoległym wszystkie dyski uczestniczą w realizacji każdego żądania wejścia-wyjścia. Zwykle poszczególne napędy są synchronizowane tak, że w dowolnym momencie każda głowica znajduje się w tej samej pozycji nad każdym dyskiem.

Podobnie jak w przypadku pozostałych schematów RAID, wykorzystywane jest paskowanie danych. W RAID 2 i 3 paski są bardzo małe, często równe jednemu bajtowi lub słowu. W RAID 2 kod korekcji błędów jest obliczany na pod-

stawie odpowiednich bitów na każdym dysku danych, a bity kodu są przechowywane w odpowiednich pozycjach bitowych zlokalizowanych na wielu dyskach parzystości. Zwykle jest wykorzystywany kod Hamminga (patrz rozdz. 4), który umożliwia korygowanie błędów jednobitowych i wykrywanie dwubitowych.

Chociaż RAID 2 wymaga mniejszej liczby dysków niż RAID 1, nadal jest raczej kosztowny. Liczba dysków redundancyjnych jest proporcjonalna do logarytmu liczby dysków danych. W przypadku pojedynczego odczytu następuje jednoczesny dostęp do wszystkich dysków. Potrzebne dane oraz towarzyszący im kod korekcji błędów są dostarczane do sterownika tablicy. Jeśli wystąpił błąd jednobitowy, sterownik może go rozpoznać i natychmiast skorygować, dzięki czemu czas dostępu odczytu nie ulega wydłużeniu. W przypadku pojedynczego zapisu musi nastąpić dostęp do wszystkich dysków danych i dysków parzystości.

Poziom RAID 2 byłby właściwym wyborem tylko w środowisku, w którym występuje wiele błędów dyskowych. Jeśli indywidualne dyski i napędy dyskowe wyróżniają się dużą niezawodnością, RAID 2 jest rozwiązaniem rozrzutnym i po prostu nie jest wykorzystywany.

RAID poziom 3

RAID 3 jest zorganizowany podobnie do RAID 2. Różnica polega na tym, że wymaga on tylko jednego dysku redundancyjnego, niezależnie od wielkości matrycy dysków. W RAID 3 wykorzystuje się dostęp równoległy, przy czym dane są rozmieszczone w postaci małych pasków. Zamiast kodu korekcyjnego jest obliczany bit parzystości dla zespołu indywidualnych bitów znajdujących się w tej samej pozycji na wszystkich dyskach danych.

Redundancja

W przypadku uszkodzenia napędu sięga się do napędu parzystości, po czym następuje rekonstrukcja danych na podstawie danych zawartych na pozostałych urządzeniach. Gdy uszkodzony dysk jest wymieniony, zaginione dane mogą być do niego wprowadzone, a przerwana operacja może być podjęta na nowo.

Rekonstrukcja danych jest całkiem prosta. Rozważmy tablicę złożoną z pięciu napędów, w której dyski X0÷X3 zawierają dane, natomiast X4 jest dyskiem parzystości. Parzystość i -tego bitu jest obliczana następująco:

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

Założmy, że został uszkodzony napęd X1. Jeśli dodamy $X4(i) \oplus X1(i)$ do obu stron powyższego równania, to otrzymamy

$$X1(i) = X4(i) \oplus X3(i) \oplus X2(i) \oplus X0(i)$$

Wobec tego zawartość dowolnego paska danych na dowolnym dysku danych w tablicy może być zregenerowana na podstawie zawartości odpowiednich pasków na pozostałych dyskach tablicy. Zasada ta jest prawdziwa dla poziomów RAID 3, 4 i 5.

W przypadku uszkodzenia dysku wszystkie dane są nadal dostępne w tzw. trybie zredukowanym. W tym trybie w przypadku odczytu zaginione dane są regenerowane na bieżąco za pomocą obliczeń (jak wyżej). Gdy dane są zapisywane w zredukowanej tablicy RAID 3, musi być zachowywana zgodność parzystości w celu umożliwienia późniejszej regeneracji. Powrót do pełnego działania wymaga wymiany uszkodzonego dysku i zregenerowania całej jego zawartości na nowym dysku.

Wydajność

Ponieważ dane są rozmieszczone w bardzo małych paskach, RAID 3 może osiągać bardzo duże szybkości transferu danych. Jakiegokolwiek żądanie wejścia-wyjścia spowoduje równoległe przesyłanie danych ze wszystkich dysków danych. Poprawa wydajności jest szczególnie widoczna w przypadku dużych transferów.

Należy jednak zauważyć, że w określonym momencie może być realizowane tylko jedno żądanie wejścia-wyjścia. Wobec tego w środowisku transakcyjnym wydajność ulega pogorszeniu.

RAID poziom 4

W RAID 4 i 5 wykorzystuje się metodę dostępu niezależnego. W tablicy o dostępie niezależnym każdy dysk działa niezależnie, dzięki czemu oddzielne żądania wejścia-wyjścia mogą być obsługiwane równoległe. Z tego względu tablice o dostępie niezależnym są bardziej odpowiednie w przypadku zastosowań wymagających szybkiej odpowiedzi na żądania wejścia-wyjścia, są natomiast stosunkowo mało przydatne w zastosowaniach wymagających dużych szybkości transferu danych.

Podobnie jak w przypadku pozostałych poziomów RAID, w RAID 4 wykorzystuje się paskowanie danych. Paski są tu stosunkowo duże. Pasek parzystości tworzony bit po bicie jest obliczany na podstawie odpowiednich pasków na każdym dysku danych, a bity parzystości są przechowywane w odpowiednim pasku na dysku parzystości.

W schemacie RAID 4 występuje pogorszenie wydajności zapisu, jeśli realizowane jest żądanie zapisu małej ilości danych. Za każdym razem, gdy następuje zapis, oprogramowanie zarządzania tablicą musi zaktualizować nie tylko dane użytkownika, ale również odpowiednie bity parzystości. Rozważmy tablicę złożoną z pięciu napędów, w której dyski X0÷X3 zawierają dane, natomiast X4 jest dyskiem parzystości. Załóżmy, że dokonywany jest zapis obejmujący tylko

pasek na dysku $X1$. Początkowo dla każdego i -tego bitu zachodzi następująca zależność:

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(0) \oplus X0(i)$$

Jeśli potencjalnie zmienione bity oznaczmy primem, to po aktualizacji zachodzi

$$\begin{aligned} X4'(i) &= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) = \\ &= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) \oplus X1(i) \oplus X1(i) = \\ &= X4(i) \oplus X1(0) \oplus X1'(i) \end{aligned}$$

W celu obliczenia nowego bitu parzystości oprogramowanie zarządzania tablicą musi odczytać stary pasek użytkownika i stary pasek parzystości. Następnie musi ono zaktualizować te oba paski za pomocą nowych danych oraz na nowo obliczonej parzystości. Tak więc każdy zapis paska powoduje dwa odczyty i dwa zapisy.

W przypadku żądania zapisu dużej ilości danych, angażującym paski na wszystkich dyskach, parzystość jest z łatwością obliczana wyłącznie na podstawie nowych bitów danych. Dzięki temu dysk parzystości może być aktualizowany równolegle z dyskami danych i nie występują dodatkowe odczyty ani zapisy.

Przy każdej operacji zapisu muszą być uaktualizowane dane na dysku parzystości, który przez to może się stać wąskim gardłem.

RAID poziom 5

RAID 5 jest zorganizowany podobnie jak RAID 4. Różnica polega na tym, że w przypadku RAID 5 paski parzystości są rozproszone na wszystkich dyskach. Typowo wykorzystuje się schemat cykliczny (ang. *round-robin*), co widać na rys. 5.5f. W przypadku tablicy n -dyskowej pasek parzystości jest umieszczany na różnych dyskach w odniesieniu do pierwszych n pasków danych, po czym schemat ten jest powtarzany.

Rozproszenie pasków parzystości na wszystkich napędach zapobiega ewentualnemu występowaniu wąskich gardeł, o których była mowa w odniesieniu do RAID 4.

5.3

PAMIĘĆ OPTYCZNA

W roku 1983 wprowadzono jeden z najbardziej udanych wyrobów konsumpcyjnych wszystkich czasów: cyfrowy system akustyczny wykorzystujący płyty kompaktowe (CD) [GUTE88]. CD jest niewymazywalnym dyskiem, na którym można zapisać ponad 60 minut informacji audio na jednej stronie. Potężny suk-

ces komercyjny płyty CD umożliwił opracowanie taniej technologii optycznego dysku pamięciowego, która stanowi zapowiedź rewolucji obejmującej przechowywanie danych w komputerach. W ciągu minionych kilku lat wprowadzono wiele systemów dysków optycznych (tabela 5.3). Trzy spośród tych systemów są coraz częściej wykorzystywane w systemach komputerowych: CD-ROM, WORM i wymazywalny dysk optyczny^{*}. Krótko przeanalizujemy każdy z tych systemów.

TABELA 5.3. Dyski optyczne

CD

Dysk kompaktowy. Niewymazywalny dysk do przechowywania informacji audio w postaci cyfrowej. W systemie znormalizowanym są używane dyski o średnicy 12 cm, przy czym czas ciągłego odtwarzania przekracza 60 min.

CD-ROM

Pamięć stała na dysku kompaktowym. Niewymazywalny dysk służący do przechowywania danych komputerowych. W systemie znormalizowanym są używane dyski o średnicy 12 cm i pojemności ponad 550 MB.

CD-I

Norma określająca sposoby umieszczania informacji audio, wideo, graficznych i tekstowych oraz programów komputerowych na CD-ROM.

DVI

Cyfrowy dysk wideo. Technologia tworzenia cyfrowej, zagęszczonej reprezentacji informacji wideo. Reprezentacja ta może być przechowywana na CD lub na innym dysku. W aktualnych systemach są używane CD, na których można zapisać około 20 minut informacji wideo.

WORM

Jednokrotny zapis, wielokrotny odczyt. Dysk, na którym zapis jest łatwiejszy niż na CD-ROM, co umożliwia handlową dostępność unikalnych dysków w jednym egzemplarzu. Podobnie jak w przypadku CD-ROM, po wykonaniu operacji zapisu dysk może być tylko odczytywany. Najpopularniejszy rozmiar to 5,25 cala, pojemność od 200 do 800 MB danych.

Wymazywalny dysk optyczny

Dysk optyczny, który może być z łatwością wymazywany i zapisywany. Używane są zarówno dyski o średnicy 3,25 cala, jak i 5,25 cala. Typowa pojemność to 650 MB.

^{*} Rodzajem pamięci optycznej, który zyskuje znaczną popularność, jest płyta DVD (*Digital Video Disk*). Historia się powtórzyła; płyta zaprojektowana głównie do rejestracji filmów stała się bardziej uniwersalnym nośnikiem danych cyfrowych o pojemności 4,7÷17 GB (*przyp.tlum.*).

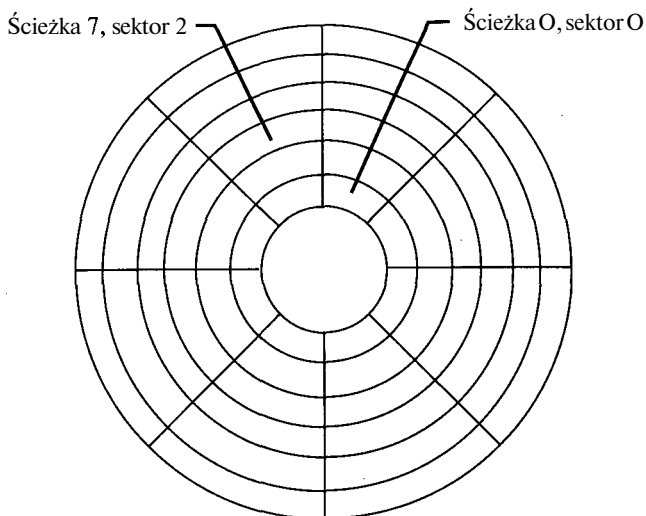
CD-ROM

Zarówno akustyczne płyty CD, jak i płyty CD-ROM (ang. *compact disk read-only memory* - pamięć stała na dyskach kompaktowych) wykorzystują podobną technologię. Główną różnicę stanowi to, że odtwarzacze CD-ROM mają podzespoły korekcyjne w celu poprawnego transferu danych z dysku do komputera. Oba typy dysków są produkowane w ten sam sposób. Dysk jest wykonywany z żywicy, np. poliwęglanowej, i pokrywany warstwą dobrze odbijającą światło, zwykle aluminium. Informacja zarejestrowana cyfrowo (albo muzyka, albo dane komputerowe) jest nanoszona w postaci mikroskopijnych zagłębień na powierzchni odbijającą. Po raz pierwszy wykonuje się to za pomocą dobrze zogniskowanego światła lasera o dużej mocy, dzięki czemu powstaje dysk wzorcowy. Dysk ten służy do wykonywania formy, która służy do tłoczenia kopii. Powierzchnia z naniesioną informacją jest zabezpieczana przed kurzem i zarysowaniem przez pokrycie przezroczystym lakierem.

Informacja jest odczytywana z płyty CD lub z CD-ROM za pomocą lasera małej mocy wbudowanego w odtwarzacz płyt lub napęd CD-ROM. Laser świeci przez przezroczyste pokrycie, a dysk jest wprawiany w ruch obrotowy przez silnik. Natężenie odbitego światła zmienia się, gdy napotyka ono wgłębienie. Zmiana ta jest wykrywana przez fotoczujniki zamieniana na sygnał cyfrowy.

Wgłębienie położone bliżej środka obracającego się dysku porusza się względem ustalonego punktu (np. wiązki laserowej) wolniej niż wgłębienie na obrzeżu płyty, trzeba więc znaleźć sposób kompensowania zmian szybkości, żeby laser mógł odczytywać dane z tą samą szybkością. Można to osiągnąć - podobnie jak w przypadku dysków magnetycznych - przez zwiększanie odległości między bitami informacji zapisanymi w segmentach dysku. Informacje mogą więc być odczytywane z tą samą szybkością przy dysku obracającym się ze **stałą prędkością kątową** (ang. *constant angular velocity* - CAV). Na rysunku 5.7 jest pokazany układ dysku wykorzystującego CAV. Dysk jest podzielony na pewną liczbę sektorów oraz na szereg koncentrycznych ścieżek. Zaletą stosowania stałej prędkości kątowej jest to, że indywidualne bloki danych mogą być bezpośrednio adresowane za pomocą ścieżki i sektora. Aby przesunąć głowicę z jej obecnego położenia pod określony adres, wymagany jest tylko mały ruch głowicy do określonej ścieżki oraz krótkie oczekiwanie, aż odpowiedni sektor znajdzie się pod głowicą. Wadą napędów CAV jest to, że ilość danych, które mogą być przechowywane na długiej, zewnętrznej ścieżce jest taka sama, jak na krótkiej ścieżce **wewnętrznej**.

Ponieważ lokowanie mniejszej ilości informacji na brzegu dysku powoduje marnowanie powierzchni, metoda CAV nie znalazła zastosowania w napędach CD i CD-ROM. Zamiast tego informacja jest umieszczana na dysku równomiernie w segmentach o tych samych rozmiarach, które są skanowane z tą samą szybkością w wyniku obracania dysku ze zmienną prędkością kątową. Wgłębienia są więc odczytywane przez laser ze **stałą prędkością liniową** (ang. *constant linear velocity* - CLV). Dysk obraca się wolniej w przypadku dostępu do danych



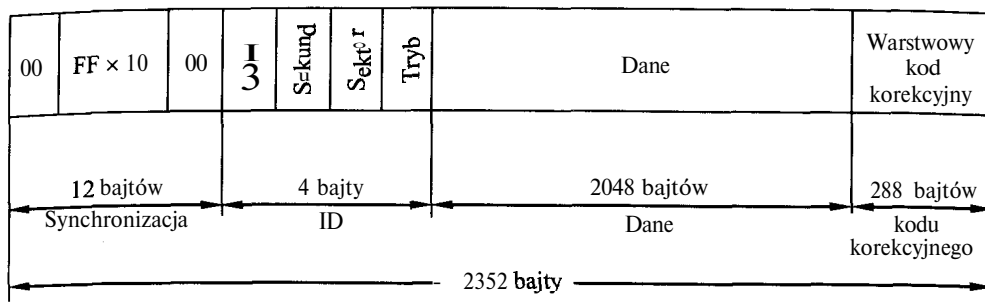
RYСУNEK 5.7. Mapa powierzchni dysku obracającego się ze stałą prędkością kątową

na obrzeżu niż w przypadku sięgania w okolice środka dysku. Tak więc pojemność ścieżki i opóźnienie obrotowe wzrastają, gdy ścieżka znajduje się blisko brzegu dysku.

Produkowano płyty CD-ROM o różnych gęstościach. Zwykle odstęp między ścieżkami wynosi 1,6 urn ($1,6 \times 10^{-6}$ m). Zapisywalna szerokość płyty CD-ROM wzdłuż promienia wynosi 32,55 mm, a więc całkowita liczba możliwych ścieżek jest równa $32\,550 \mu\text{m}$ podzielone przez odstęp ścieżek, co daje 20 344 ścieżek. W rzeczywistości mamy do czynienia z pojedynczą ścieżką spiralną i możemy obliczyć jej długość, mnożąc średni obwód przez liczbę zwojów spirali; daje to około 5,27 km. Stała prędkość liniowa napędu CD-ROM wynosi 1,2 m/s, co daje czas zapisu całej ścieżki 4 391 s lub 73,2 min, co jest zbliżone do standardowego maksymalnego czasu odtwarzania akustycznej płyty kompaktowej. Ponieważ dane są odczytywane z dysku z szybkością 176,4 KB/s, pojemność płyty CD-ROM wynosi 774,57 MB. Jest to równoważne pojemności ponad 550 dyskietek 3,25-calowych.

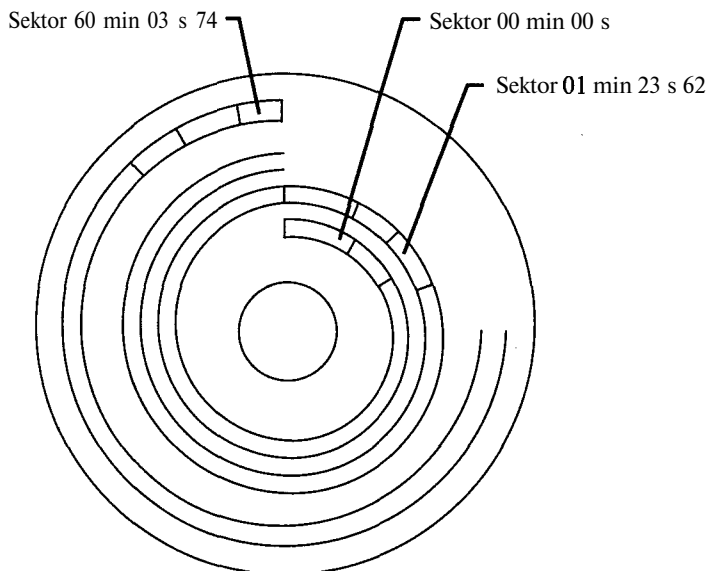
Dane na płycie CD-ROM są zorganizowane w postaci ciągu bloków. Typowy format bloku widać na rys. 5.8. Blok składa się z następujących pól:

- a Sync (synchronizacja): Pole identyfikujące początek bloku; składa się z bajta samych 0, 10 bajtów samych 1 oraz z bajta samych 0.
- Header (nagłówek): Zawiera adres bloku i bajt trybu. Tryb 0 oznacza czyste pole danych, tryb 1 - wykorzystanie kodu korekcyjnego i 2048 bajtów danych, a tryb 2 - 2336 bajtów danych użytkownika bez kodu korekcyjnego.
- a Data (dane): Dane użytkownika.
- Auxiliary (pomocnicze): Dodatkowe dane użytkownika w trybie 2, w trybie 1 jest to 288-bajtowy kod korekcyjny.



RYSUNEK 5.8. Format bloku CD-ROM

Na rysunku 5.9 jest pokazany rozkład danych na płytach CD i CD-ROM. Jak już wspomnieliśmy, dane są ułożone w ciągu wzdłuż spiralnej ścieżki. W przypadku wykorzystywania napędu CLV dostęp swobodny jest trudniejszy. Lokalizowanie określonego adresu wymaga wstępnego przesunięcia głowicy, dostosowania prędkości obrotowej, odczytania adresu, a następnie dokonania nieznacznych korekcyi w celu znalezienia i uzyskania dostępu do potrzebnego sektora.



RYSUNEK 5.9. Mapa powierzchni dysku obracającego się ze stałą prędkością liniową

Dysk CD-ROM jest odpowiedni do rozpowszechniania dużych ilości danych wśród dużej liczby użytkowników. Ze względu na koszt procesu pierwotnego zapisu, nie jest odpowiedni do zastosowań jednostkowych. W porównaniu z tradycyjnymi dyskami magnetycznymi CD-ROM ma trzy główne zalety:

- ❑ Pojemność dysku CD-ROM jest o wiele większa.

- a Dysk optyczny wraz z przechowywaną na nim informacją może być tanio masowo powielany, w przeciwieństwie do dysku magnetycznego. Baza danych na dysku magnetycznym może być reprodukowana przez skopiowanie każdorazowo jednego dysku za pomocą dwóch napędów.
- a Dysk optyczny jest dyskiem wymiennym, co pozwala na wykorzystywanie samego dysku do archiwizacji. Większość dysków magnetycznych to dyski niewymienne. Informacja musi być skopiowana na taśmę, zanim dysk z napędem będą gotowe do wprowadzania nowej informacji.

Wady dysku CD-ROM są następujące:

- a Jest to pamięć stała i nie może być aktualizowana.
- a Ma czas dostępu o wiele dłuższy niż magnetyczna pamięć dyskowa, wynoszący nawet 0,5 s.

WORM

W celu dostosowania się do takich zastosowań, w których potrzebna jest tylko jedna lub niewielka liczba kopii danych, opracowano pamięć CD z jednokrotnym zapisem i wielokrotnym odczytem (ang. *write-once read-many* - WORM). W przypadku pamięci WORM dysk jest przygotowywany w taki sposób, że może on być jednorazowo zapisany za pomocą wiązki laserowej o umiarkowanym natężeniu. Dzięki temu, posługując się nieco droższym sterownikiem niż w przypadku CD-ROM, klient może jednorazowo zapisać i wielokrotnie odczytywać dysk. W celu umożliwienia szybszego dostępu w napędach WORM wykorzystuje się stałą prędkość kątową, poświęcając część pojemności.

Typowa metoda przygotowania dysku polega na wykorzystaniu lasera dużej mocy do wytworzenia szeregów pęcherzyków na dysku. Gdy wstępnie uformowany dysk jest umieszczony w napędzie WORM, laser o małej mocy jest w stanie wytworzyć dostatecznie dużo ciepła, aby przepalić wstępnie przygotowane pęcherzyki. Podczas operacji odczytywania dysku laser w napędzie WORM oświetla powierzchnię dysku. Ponieważ przepalone pęcherzyki kontrastują z otoczeniem, mogą być z łatwością rozpoznane za pomocą prostych układów elektronicznych.

Dysk optyczny WORM jest atrakcyjny do archiwizowania dokumentów i plików. Umożliwia trwałe zapisywanie dużej ilości danych użytkownika.

Wymazywalny dysk optyczny

Osiągnięciem w zakresie optycznych dysków komputerowych jest wymazywalny dysk optyczny. Dysk ten umożliwia powtarzalny zapis, podobnie jak dysk magnetyczny. Chociaż wypróbowano wiele rozwiązań, jedyną technologią, która znalazła komercyjne zastosowanie, jest system magneto-optyczny. W systemie tym energia wiązki laserowej jest stosowana w połączeniu z polem magnetycznym do zapisywania i wymazywania informacji, co jest wykonywane przez

odwracanie biegunów magnetycznych w małej powierzchni dysku pokrytego materiałem magnetycznym. Wiązka laserowa nagrzewa określoną plamkę na dysku, a pole magnetyczne jest w stanie zmienić orientację magnetyczną tej plamki w czasie, gdy jej temperatura jest zwiększona. Ponieważ proces polaryzacji nie powoduje trwałej zmiany w dysku, może on być powtarzany wielokrotnie. W przypadku odczytu kierunek namagnesowania może być wykryty przez spolaryzowane światło lasera. Spolaryzowane światło odbite od określonej plamki zmienia swój kąt polaryzacji zależnie od orientacji pola magnetycznego.

Wymazywalny dysk optyczny ma oczywistą zaletę w porównaniu z dyskami CD-ROM i WORM: może być ponownie zapisywany i dzięki temu może grać rolę prawdziwej pamięci pomocniczej. Głównymi zaletami wymazywalnych dysków optycznych w porównaniu z dyskami magnetycznymi są:

- a *Duża pojemność.* Na dysku optycznym o średnicy 5,25 cala można przechowywać około 650 MB danych. Najbardziej złożone dyski typu Winchester mają połowę tej pojemności*.
- Q *Możliwość przenoszenia.* Dysk optyczny może być wyjmowany z napędu.
- a *Niezawodność.* Tolerancje wymagane w odniesieniu do dysków optycznych są znacznie większe niż w odniesieniu do dysków magnetycznych o dużej pojemności. Dzięki temu są one bardziej niezawodne i trwałe.

Tak jak dysk WORM, wymazywalny dysk optyczny jest odczytywany ze stałą prędkością kątową.

5.4

TAŚMA MAGNETYCZNA

W systemach taśmowych wykorzystuje się te same metody odczytu i zapisu, co w systemach dyskowych. Nośnikiem jest elastyczna taśma mylarowa pokryta tlenkiem magnetycznym. Taśma i napęd taśmy są podobne do wykorzystywanych w domowych systemach magnetofonowych.

Nośnikowi taśmowemu nadaje się strukturę złożoną z niewielkiej liczby równoległych ścieżek. Wcześniejsze systemy taśmowe wykorzystywały zwykle

* Współczesne systemy dysków twardych przeznaczone dla komputerów powszechnego użytku mają pojemności 4+25 GB (*przyp.tlum*).

- a Dysk optyczny wraz z przechowywaną na nim informacją może być tanio masowo powielany, w przeciwieństwie do dysku magnetycznego. Baza danych na dysku magnetycznym może być reprodukowana przez skopiowanie każdorazowo jednego dysku za pomocą dwóch napędów.
- a Dysk optyczny jest dyskiem wymiennym, co pozwala na wykorzystywanie samego dysku do archiwizacji. Większość dysków magnetycznych to dyski niewymienne. Informacja musi być skopiowana na taśmę, zanim dysk z napędem będą gotowe do wprowadzania nowej informacji.

Wady dysku CD-ROM są następujące:

- a Jest to pamięć stała i nie może być aktualizowana.
- a Ma czas dostępu o wiele dłuższy niż magnetyczna pamięć dyskowa, wynoszący nawet 0,5 s.

WORM

W celu dostosowania się do takich zastosowań, w których potrzebna jest tylko jedna lub niewielka liczba kopii danych, opracowano pamięć CD z jednokrotnym zapisem i wielokrotnym odczytem (ang. *write-once read-many* - WORM). W przypadku pamięci WORM dysk jest przygotowywany w taki sposób, że może on być jednorazowo zapisany za pomocą wiązki laserowej o umiarkowanym natężeniu. Dzięki temu, posługując się nieco droższym sterownikiem niż w przypadku CD-ROM, klient może jednorazowo zapisać i wielokrotnie odczytywać dysk. W celu umożliwienia szybszego dostępu w napędach WORM wykorzystuje się stałą prędkość kątową, poświęcając część pojemności.

Typowa metoda przygotowania dysku polega na wykorzystaniu lasera dużej mocy do wytworzenia szeregów pęcherzyków na dysku. Gdy wstępnie uformowany dysk jest umieszczony w napędzie WORM, laser o małej mocy jest w stanie wytworzyć dostatecznie dużo ciepła, aby przepalić wstępnie przygotowane pęcherzyki. Podczas operacji odczytywania dysku laser w napędzie WORM oświetla powierzchnię dysku. Ponieważ przepalane pęcherzyki kontrastują z otoczeniem, mogą być z łatwością rozpoznane za pomocą prostych układów elektronicznych.

Dysk optyczny WORM jest atrakcyjny do archiwizowania dokumentów i plików. Umożliwia trwałe zapisywanie dużej ilości danych użytkownika.

Wymazywalny dysk optyczny

Osiągnięciem w zakresie optycznych dysków komputerowych jest wymazywalny dysk optyczny. Dysk ten umożliwia powtarzalny zapis, podobnie jak dysk magnetyczny. Chociaż wypróbowano wiele rozwiązań, jedyną technologią, która znalazła komercyjne zastosowanie, jest system magneto-optyczny. W systemie tym energia wiązki laserowej jest stosowana w połączeniu z polem magnetycznym do zapisywania i wymazywania informacji, co jest wykonywane przez

odwracanie biegunów magnetycznych w małej powierzchni dysku pokrytego materiałem magnetycznym. Wiązka laserowa nagrzewa określoną plamkę na dysku, a pole magnetyczne jest w stanie zmienić orientację magnetyczną tej plamki w czasie, gdy jej temperatura jest zwiększona. Ponieważ proces polaryzacji nie powoduje trwałej zmiany w dysku, może on być powtarzany wielokrotnie. W przypadku odczytu kierunek namagnesowania może być wykryty przez spolaryzowane światło lasera. Spolaryzowane światło odbite od określonej plamki zmienia swój kąt polaryzacji zależnie od orientacji pola magnetycznego.

Wymazywalny dysk optyczny ma oczywistą zaletę w porównaniu z dyskami CD-ROM i WORM: może być ponownie zapisywany i dzięki temu może grać rolę prawdziwej pamięci pomocniczej. Głównymi zaletami wymazywalnych dysków optycznych w porównaniu z dyskami magnetycznymi są:

- a *Duża pojemność.* Na dysku optycznym o średnicy 5,25 cala można przechowywać około 650 MB danych. Najbardziej złożone dyski typu Winchester mają połowę tej pojemności*.
- a *Możliwość przenoszenia.* Dysk optyczny może być wyjmowany z napędu.
- a *Niezawodność.* Tolerancje wymagane w odniesieniu do dysków optycznych są znacznie większe niż w odniesieniu do dysków magnetycznych o dużej pojemności. Dzięki temu są one bardziej niezawodne i trwałe.

Tak jak dysk WORM, wymazywalny dysk optyczny jest odczytywany ze stałą prędkością kątową.

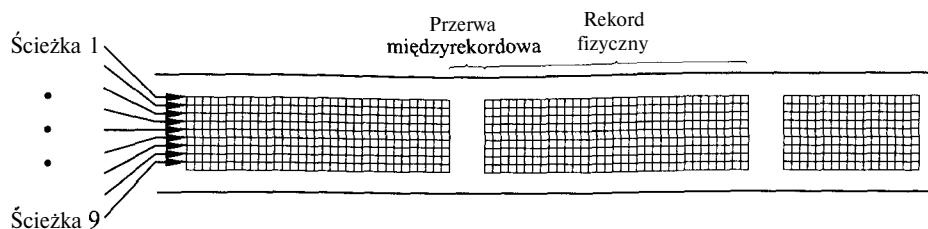
5.4

TAŚMA MAGNETYCZNA

W systemach taśmowych wykorzystuje się te same metody odczytu i zapisu, co w systemach dyskowych. Nośnikiem jest elastyczna taśma mylarowa pokryta tlenkiem magnetycznym. Taśma i napęd taśmy są podobne do wykorzystywanych w domowych systemach magnetofonowych.

Nośnikowi taśmowemu nadaje się strukturę złożoną z niewielkiej liczby równoległych ścieżek. Wcześniejsze systemy taśmowe wykorzystywały zwykle

* Współczesne systemy dysków twardych przeznaczone dla komputerów powszechnego użytku mają pojemności 4÷25 GB (*przyp. tłum.*).



RYSunEK 5.10. Format 9-ścieżkowej taśmy magnetycznej

9 ścieżek. Umożliwiało to jednoczesne zapisywanie jednego bajta danych; na dziewiątej ścieżce zapisywano dodatkowy bit parzystości. Nowsze systemy taśmowe wykorzystują 18 lub 36 ścieżek, co odpowiada słowu lub podwójnemu słowu. Podobnie jak w przypadku dysku, dane są odczytywane i zapisywane na taśmie w postaci kolejnych bloków zwanych **rekordami fizycznymi**. Bloki na taśmie są oddzielane przerwami określanymi jako **przerwy międzyrekordowe**. Na rysunku 5.10 jest pokazana taka struktura taśmy 9-ścieżkowej. Jak w przypadku dysku, taśma jest formatowana w celu ułatwienia lokalizacji rekordów fizycznych.

Napęd taśmowy jest urządzeniem o *dostępie sekwencyjnym*. Jeśli głowica znajduje się przy rekordzie 1, to w celu odczytania rekordu N konieczne jest kolejne odczytanie rekordów fizycznych od 1 do $N - 1$. Jeśli głowica znajduje się za poszukiwanym rekordem, konieczne jest przewinięcie pewnej długości taśmy i ponowne rozpoczęcie odczytu. W przeciwieństwie do dysku taśma znajduje się w ruchu tylko podczas operacji odczytu lub zapisu.

W odróżnieniu od taśmy napęd dyskowy jest określany jako urządzenie o *dostępie bezpośrednim*. Napęd dyskowy nie musi odczytywać sekwencyjnie wszystkich sektorów na dysku w celu dotarcia do poszukiwanego sektora. Musi tylko czekać na właściwy sektor na jednej ścieżce i może sukcesywnie sięgać do dowolnej ścieżki.

Pamięć taśmowa była pierwszym rodzajem pamięci pomocniczej. Wciąż jest szeroko używana jako najtańszy i najpowolniejszy element hierarchii pamięci.

POLECANA LITERATURA

Dobry opis techniki dyskowej zawiera [SIER90].

Doskonałym przeglądem techniki RAID, napisanym przez wynalazców tej koncepcji, jest [CHEN94]. Bardziej szczegółowa dyskusja została opublikowana przez RAID Advisory Board, będące stowarzyszeniem dostawców i użytkowników wyrobów związanych z RAID [MASS94].

Praca [MARC90] zawiera doskonały przegląd pamięci optycznych. Obszerzny opis wszystkich typów zewnętrznych systemów pamięci znajduje się w [ROSC94], przy czym ilość szczegółów technicznych na temat każdego z nich jest umiarkowana.

CHEN94 Chen P., Lee E., Gibson G., Katz R., Patterson D.: „RAID: High-Performance, Reliable Secondary Storage”. *ACM Computing Surveys*, June 1994.

MARC90 Marchant A.: *Optical Recording*. Reading, Addison-Wesley, 1990.

MASS94 Massiglia P. (redaktor): *The RAIDbook: a Sourcebook for Disk Array Technology*. St. Peter, The Raid Advisory Board, 1994.

ROSC94 Rosch W.: *The Winn L. Rosch Hardware Bible*. Indianapolis, Sams, 1994.

SIER90 Sierra H.: *An Introduction to Direct Access Storage Devices*. Boston, Academic Press, 1990.

5.6

PROBLEMY DO ROZWIĄZANIA

5.1. Zdefiniowane są następujące parametry systemu dyskowego:

t_s - czas przeszukiwania; średni czas umieszczania głowicy nad ścieżką,

r - prędkość obrotowa dysku, w obrotach na sekundę,

n - liczba bitów na sektor,

N - pojemność ścieżki, w bitach,

t_A - czas dostępu do sektora.

Wyprowadź wzór na t_A jako funkcję pozostałych parametrów.

5.2. Przyjmij 10-napędową konfigurację RAID. Wypełnij następującą tablicę, służącą do porównania różnych poziomów RAID:

Poziom RAID	Gęstość upakowania	Szerokość pasma	Wydajność transakcyjna
0	1		1
1			
2			
3		1	
4			
5			

Każdy parametr jest znormalizowany w stosunku do tego poziomu RAID, który zapewnia najlepszą wydajność. Gęstość upakowania pamięci określa część pojemności dysków dostępną dla danych użytkownika. Szerokość pasma odzwierciedla szybkość transferu danych z tablicy. Wydajność transakcyjna jest miarą liczby operacji na sekundę, którą może realizować tablica.

5.3. Powinno być oczywiste, że paskowanie dysku może poprawić szybkość przesyłania danych, jeśli rozmiar paska jest mały w porównaniu z rozmiarem żądania wejścia-wyjścia. Powinno być również oczywiste, że RAID 0 umożliwia zwiększenie wydajności w stosunku do pojedynczego, dużego

dysku, ponieważ wiele żądań wejścia-wyjścia może być obsługiwanych równolegle. Jednakże, czy w tym ostatnim przypadku paskowanie dysku jest konieczne? To znaczy, czy paskowanie dysku umożliwia zwiększenie szybkości odpowiedzi na żądania wejścia-wyjścia w porównaniu z analogiczną tablicą dysków bez paskowania?

- 5.4. Jaka jest szybkość przesyłania danych w przypadku 9-ścieżkowej magnetycznej pamięci taśmowej, w której prędkość taśmy wynosi 120 cali na sekundę (304,8 cm/s), a gęstość upakowania wynosi 1600 bitów na cal (ok. 630 bit/cm)?
- 5.5. Przyjmij, że na szpuli jest 2400 stóp (731,5 m) taśmy; przerwa międzyrekordowa, w której taśma zatrzymuje się między odczytami, wynosi 0,6 cala (1,52 cm); zmniejszenie lub zwiększenie prędkości przewijania taśmy w przerwach ma charakter liniowy; pozostałe własności taśmy są takie same, jak w problemie 5.4. Dane na taśmie są zorganizowane w postaci rekordów fizycznych, przy czym każdy rekord fizyczny zawiera ustaloną liczbę jednostek zdefiniowanych przez użytkownika, zwanych rekordami logicznymi.
- (a) Ile czasu zajmie odczytanie całej taśmy zawierającej 120-bajtowe rekordy logiczne zgrupowane po 10 na każdy rekord fizyczny?
 - (b) Ile czasu zajmie odczytanie całej taśmy zawierającej 120-bajtowe rekordy logiczne zgrupowane po 30 na każdy rekord fizyczny?
 - (c) Ile rekordów logicznych zawiera taśma w obu powyższych przypadkach?
 - (d) Jaka jest efektywna ogólna szybkość przesyłania danych w przypadkach (a) i (b)?
 - (e) Jaka jest pojemność taśmy?
- 5.6. Oblicz, ile przestrzeni dyskowej (mierzonej w sektorach, ścieżkach i powierzchni) jest wymagane do przechowania rekordów logicznych przedstawionych w problemie 5.5a, jeśli dysk ma ustalone sektory o pojemności 512 bajtów na sektor, 96 sektorów przypada na ścieżkę, 110 ścieżek na jedną powierzchnię, a używalnych powierzchni jest 8. Zignoruj jakiekolwiek rekordy obejmujące głowice plików oraz indeksy ścieżek. Załóż, że rekordy nie mogą obejmować więcej niż 1 sektor.
- 5.7. Jeśli zaniedbamy czas procesora, jak długo będzie trwało zapisanie rekordów z problemu 5.6 sekwencyjnie na dysku, jeśli średni czas przejścia od ścieżki do ścieżki wynosi 8 ms, a dysk obraca się z prędkością 360 obrotów na minutę? Załóż, że głowica znajduje się początkowo nad pierwszą ścieżką przeznaczoną do zapisu.
- 5.8. Jeśli rozważymy razem problemy 5.5b i 5.7 oraz zaniedbamy czas procesora, czy kopiowanie z taśmy na dysk będzie ograniczane wydajnością taśmy, czy też wydajnością dysku? Załóż, że moduły wejścia-wyjścia są oddzielne.

- 5.9. W przypadku jednostki taśmowej z problemu 5.4, jaka będzie efektywna szybkość przesyłania danych, jeśli dane będą numeryczne, upakowane po 2 cyfry na bajt?
- 5.10. Gdyby w problemie 5.5a taśma nie zwalniała ani nie zatrzymywała się w przerwach (tzw. szybka taśma - ang. *streaming tape*), jaka byłaby poprawa czasu odczytu tych samych danych?

Wejście-wyjście

Obok procesora i zespołu modułów pamięci trzecim kluczowym elementem systemu komputerowego jest zespół modułów wejścia-wyjścia. Każdy moduł jest dołączony do magistrali systemowej lub centralnego przełącznika i steruje jednym lub wieloma urządzeniami peryferyjnymi. Moduł wejścia-wyjścia nie jest po prostu złączem mechanicznym służącym do połączenia urządzenia z magistralą systemową. Zawiera on pewną „inteligencję”, to znaczy układy logiczne umożliwiające komunikację między urządzeniem peryferyjnym a magistralą.

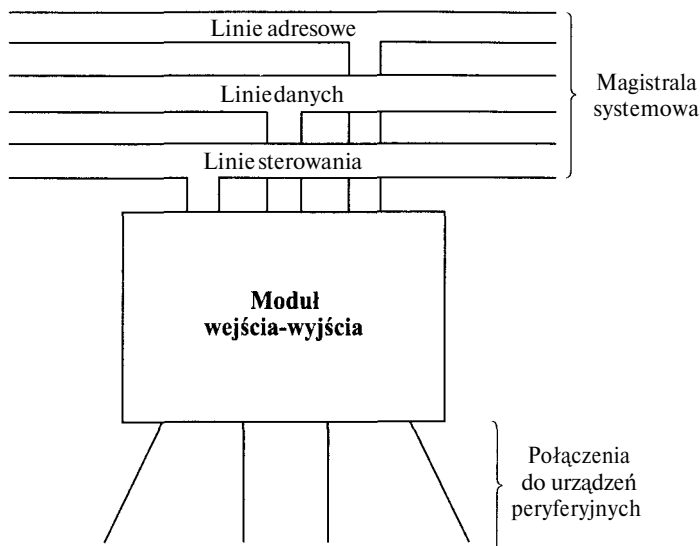
Czytelnik może się dziwić, dlaczego nie łączy się urządzeń peryferyjnych bezpośrednio z magistralą systemową. Przyczyny są następujące:

- a Istnieje znaczna różnorodność urządzeń peryferyjnych, różniących się sposobem pracy. Byłoby niepraktyczne wbudowywanie niezbędnych układów logicznych do procesora w celu umożliwienia sterowania tak szerokim zakresem urządzeń.
- Q Szybkość transferu (przesyłania) danych do/z urządzeń peryferyjnych jest o wiele mniejsza niż w przypadku pamięci czy procesora. Jest więc niepraktyczne wykorzystywanie szybkiej magistrali systemowej do bezpośredniego komunikowania się z urządzeniami peryferyjnymi.
- a Urządzenia peryferyjne wykorzystują często inne formaty danych i długości słowa niż komputery, do których są dołączone.

Potrzebny jest więc moduł wejścia-wyjścia. Moduł ten gra dwie role (rys. 6.1):

- a Interfejsu z procesorem i z pamięcią poprzez magistralę systemową lub centralnego przełącznika.
- Interfejsu z jednym lub wieloma urządzeniami peryferyjnymi przez dostosowanie łącza danych.

Rozpoczniemy ten rozdział od krótkiego przeglądu urządzeń zewnętrznych, po czym omówimy strukturę i funkcję modułu wejścia-wyjścia. Rozpatrzmy następnie różne sposoby realizowania funkcji modułu wejścia-wyjścia we współpracy z procesorem i z pamięcią, tzn. wewnętrzny interfejs wejścia-wyjścia. Na zakończenie przeanalizujemy zewnętrzny interfejs wejścia-wyjścia między modułem wejścia-wyjścia a światem zewnętrznym.



RYSUNEK 6.1. Ogólny model modułu wejścia-wyjścia

6.1

URZĄDZENIA ZEWNĘTRZNE

System komputerowy pozbawiony wejścia i wyjścia jest bezużyteczny. Operacje wejścia-wyjścia są realizowane za pomocą szerokiego asortymentu urządzeń zewnętrznych, które umożliwiają wymianę danych między otoczeniem zewnętrznym a komputerem. Urządzenie zewnętrzne współpracuje z komputerem poprzez łącze z modułem wejścia-wyjścia (rys. 6.1). Łącze jest używane do wymiany sygnałów sterowania i stanu oraz danych między modułem wejścia-wyjścia a urządzeniem zewnętrznym. Urządzenie zewnętrzne połączone z modułem wejścia-wyjścia jest często określane jako **urządzenie peryferyjne**.

Możemy sklasyfikować urządzenia zewnętrzne na trzy kategorie:

- a *Przeznaczone do odczytywania przez człowieka.* Odpowiednie do komunikowania się z użytkownikiem komputera.
- *Przeznaczone do odczytywania przez maszynę.* Odpowiednie do komunikowania się ze sprzętem.
- a *Komunikacyjne.* Odpowiednie do komunikowania się z odległymi urządzeniami.

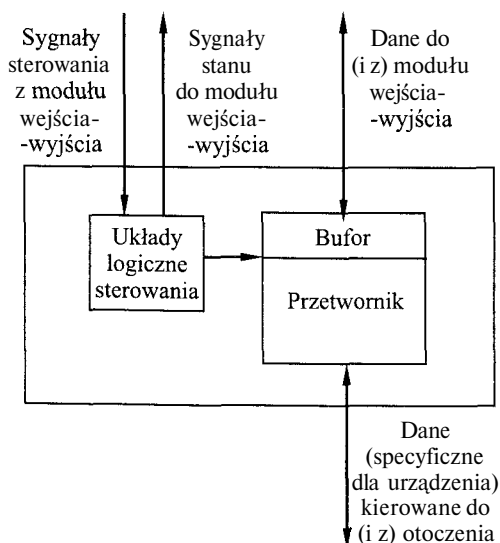
Przykładami urządzeń przeznaczonych dla człowieka są terminale wizyjne (ang. *video display terminal* - VDT) i drukarki. Przykładami urządzeń przeznaczonych dla maszyny są dyski magnetyczne, systemy taśmowe oraz czujniki i urządzenia wykonawcze wykorzystywane w robotach. Zauważmy, że w tym rozdziale traktujemy systemy dyskowe i taśmowe jako urządzenia wejścia-

-wyjścia, podczas gdy w rozdz. 5 uważaliśmy je za urządzenia pamięci. Z funkcjonalnego punktu widzenia urządzenia te stanowią część hierarchii pamięci i dlatego dyskusowanie ich w rozdziale 5 było właściwe. Ze strukturalnego punktu widzenia urządzenia te są sterowane przez moduły wejścia-wyjścia i dlatego zostaną rozważone także w tym rozdziale.

Urządzenia komunikacyjne umożliwiają komputerowi wymianę danych z odległymi urządzeniami, które mogą być urządzeniami przeznaczonymi dla człowieka, jak terminale, urządzeniami przeznaczonymi dla maszyny lub nawet innymi komputerami.

W bardzo ogólnym ujęciu, natura urządzenia zewnętrznego jest pokazana na rys. 6.2. Interfejs z modułem wejścia-wyjścia ma postać sygnałów sterowania, stanu i danych. *Dane* mają postać zespołu bitów przeznaczonych do wysłania lub odbieranych z modułu wejścia-wyjścia. *Sygnały sterujące* określają funkcję, jaką ma pełnić urządzenie, np. wysłać dane do modułu wejścia-wyjścia (WEJŚCIE lub ODCZYT), przyjąć dane z modułu wejścia-wyjścia (WYJŚCIE lub ZAPIS), poinformować o stanie lub wykonać pewne funkcje sterujące właściwe dla danego urządzenia (np. ustawić głowicę dysku). *Sygnały stanu* wskazują stan urządzenia. Przykładami są GOTOWOŚĆ/BRAK GOTOWOŚCI (READY/NOT READY), wskazujące, czy urządzenie jest gotowe do przesyłania danych.

Logiczne układy sterowania związane z urządzeniem sterują jego pracą w odpowiedzi na polecenia płynące z modułu wejścia-wyjścia. *Przetwornik* dokonuje konwersji danych z postaci elektrycznej na inną formę energii w przypadku wyjścia, zaś z innych form na elektryczną podczas wejścia. Zwykle z przetwornikiem jest związany bufor umożliwiający czasowe przechowywanie danych przenoszonych między modułem wejścia-wyjścia a otoczeniem zewnętrznym; bufor ten ma zwykle rozmiar 8÷16 bitów.



RYSUNEK 6.2. Urządzenie zewnętrzne

Interfejs między modułem wejścia-wyjścia a urządzeniem zewnętrznym omówimy w podrozdz. 6.7. Interfejs między urządzeniem zewnętrznym a otoczeniem wykracza poza zakres tej książki, jednak przytoczymy kilka krótkich przykładów.

Klawiatura/monitor

Najbardziej powszechnym środkiem współpracy między komputerem a użytkownikiem jest zespół klawiatura/monitor. Użytkownik wprowadza dane za pomocą klawiatury. Dane wejściowe są następnie transmitowane do komputera i mogą być również zobrazowane na monitorze. Monitor pokazuje również dane dostarczane przez komputer.

Podstawową wymienianą jednostką jest znak. Z każdym znakiem wiąże się kod, zwykle o długości 7 do 8 bitów. Najczęściej używanym kodem jest kod 7-bitowy nazywany ASCII (skrót od *American Standard Code for Information Interchange* - amerykański kod standardowy do wymiany informacji), zwany również CCITT Alphabet No 5. Każdy znak w tym kodzie jest reprezentowany przez unikalny 7-bitowy kod binarny; możliwe jest więc reprezentowanie 128 różnych znaków¹. W tabeli 6.1 są podane wszystkie wartości kodu. W tablicy tej bity każdego znaku są oznaczane od b_7 , który jest najbardziej znaczącym bitem, do b_0 - najmniej znaczącego bitu. Występują dwa typy znaków: drukowalne i kontrolne. Drukowalnymi znakami są znaki alfabetu, cyfry i znaki specjalne, które mogą być drukowane na papierze lub wyświetlane na ekranie (np. znaki interpunkcyjne). Bitową reprezentacją na przykład znaku „K” jest 1001011. Niektóre ze znaków kontrolnych są wykorzystywane przy drukowaniu lub wyświetlaniu znaków; przykładem jest znak powrotu karetki*. Inne znaki kontrolne są związane z procedurami komunikacyjnymi.

W przypadku wejścia z klawiatury naciśnięcie klawisza przez użytkownika powoduje wygenerowanie sygnału elektrycznego, który jest interpretowany przez przetwornik w klawiaturze i tłumaczony na wzór bitowy odpowiedniego kodu ASCII. Wzór bitowy jest następnie transmitowany do modułu wejścia-wyjścia w komputerze. W samym komputerze tekst może być przechowywany w tym samym kodzie ASCII. W przypadku wyjścia kody ASCII znaków są transmitowane z modułu wejścia-wyjścia do urządzenia zewnętrznego. Przetwornik w tym urządzeniu interpretuje kod i wysyła niezbędne sygnały elektryczne do urządzenia wyjściowego w celu albo wyświetlenia wskazanego znaku, albo wykonania wymaganej funkcji sterującej.

¹ Znaki kodowane za pomocą ASCII są prawie zawsze przechowywane lub przesyłane po 8 bitów na znak (blok 8 bitów jest określany jako oktet lub bajt). Ósmy bit jest bitem parzystości wykorzystywanym do wykrywania błędów. Bit parzystości znajduje się w pozycji najbardziej znaczącego bitu i dlatego jest symbolizowany jako b_8 .

* Dotyczy urządzeń drukujących (przyp. tłum.).

Tabela 6.1. Amerykański kod standardowy do wymiany informacji (ASCII)

	0	0	0	0	1	1	0	1	1	1	1	
		0	0	1	1	0	0	1	1	1	1	
			0	1	0	1	0	1	0	1		
b ₇ b ₆ b ₅ b ₄ b ₃ b ₂ b ₁												
0 0 0 0	NUL	DLE	SP	0	@	P	-	p				
0 0 0 1	SOH	DC1	!	1	A	Q	a	q				
0 0 1 0	STX	DC2	"	2	B	R	b	r				
0 0 1 1	ETX	DC3	#	3	C	S	c	s				
0 1 0 0	EOT	DC4	\$	4	D	T	d	t				
0 1 0 1	ENQ	NAK	%	5	E	U	e	u				
0 1 1 0	ACK	SYN	&	6	F	V	f	v				
0 1 1 1	BEL	ETB	.	7	G	W	g	w				
1 0 0 0	BS	CAN	(	8	H	X	h	x				
1 0 0 1	HT	EM	)	9	I	Y	i	y				
1 0 1 0	LF	SUB	*	:	J	Z	j	z				
1 0 1 1	VT	ESC	+	;	K	[	k	{				
1 1 0 0	FF	FS	,	<	L	\	l					
1 1 0 1	CR	GS	-	=	M	]	m	}				
1 1 1 0	SO	RS	.	>	N	^	n	~				
1 1 1 1	SI	US	/	?	O	_	o	DEL				

Jest to wersja USA Międzynarodowego Alfabetu Numer 5 CCITT (T.50). Znaki kontrolne objaśniono w dalszym ciągu.

Kontrola formatu

BS (cofnięcie; ang. *backspace*): oznacza przesunięcie mechanizmu drukującego lub kursora wstecz o jedną pozycję

HT (tabulacja pozioma; ang. *horizontal tab*): oznacza przesunięcie mechanizmu drukującego lub kursora do następnej pozycji tabulacyjnej

LF (przesunięcie o wiersz; ang. *line feed*): oznacza przesunięcie mechanizmu drukującego lub kursora do początku następnego wiersza

VT (tabulacja pionowa; ang. *vertical tab*): oznacza przesunięcie mechanizmu drukującego lub kursora do następnego wiersza

FF (wysunięcie strony; ang. *form feed*): oznacza przesunięcie mechanizmu drukującego lub kursora do początkowej pozycji na następnej stronie lub na ekranie

CR (powrót karetki; ang. *carriage return*): oznacza przesunięcie mechanizmu drukującego lub kursora do początku tego samego wiersza

Kontrola transmisji

SOH (początek nagłówka; ang. *start of heading*): wskazuje początek nagłówka, który może zawierać adres lub informacje o trasie

NAK (potwierdzenie negatywne; ang. *negative acknowledgment*): znak transmitowany przez przyrząd odbierający jako odpowiedź negatywna skierowana do nadawcy; używany jako odpowiedź negatywna na komunikaty odpytywania

STX (początek tekstu; ang. *start of text*): wskazuje początek tekstu, a także koniec nagłówka

SYN (synchronizacja; ang. *synchronous/idle*): używany do osiągnięcia synchronizacji w synchronicznych systemach transmisyjnych; gdy nie są wysyłane dane, system taki może ciągle nadawać znaki SYN

Tabela 6.1 (cd.)

ETX (koniec tekstu; ang. <i>end of text</i>): używany do zakończenia tekstu rozpoczętego za pomocą STX	ETB (koniec bloku transmisji; ang. <i>end of transmission block</i>): wskazuje koniec bloku danych przy połączeniu; używany do formowania bloków danych, gdy struktura blokowa niekoniecznie wynika z formatu przetwarzania
EOT (koniec transmisji; ang. <i>end of transmission</i>): oznacza koniec transmisji jednego lub więcej tekstu opatrzonego nagłówkiem	ACK (potwierdzenie; ang. <i>acknowledge</i>): nadawany przez przyrząd odbierający jako odpowiedź potwierdzająca; używany jako odpowiedź pozytywna na komunikaty odpytowania
ENQ (zapytanie; ang. <i>enquiry</i>): zapytanie o odpowiedź odległej stacji; może być używane jako zapytanie „Kim jesteś?” kierowane w celu identyfikacji stacji	
<i>Separatory informacji</i>	
FS (separator plików; ang. <i>file separator</i>)	Separatory informacji mogą być używane dowolnie, jednak z zachowaniem hierarchii od FS do US
GS (separator grup; ang. <i>group separator</i>)	
RS (separator rekordów; ang. <i>record separator</i>)	
US (separator połączony; ang. <i>united separator</i>)	
<i>Różne</i>	
NUL (brak znaku; ang. <i>null</i>): używany do wypełniania czasu lub wypełniania przestrzeni na taśmie, gdy nie ma danych	DLE (znak sterujący transmisją; ang. <i>data link escape</i>): znak zmieniający znaczenie jednego lub wielu następnych znaków. Umożliwia dodatkowe sterowanie lub pozwala na wysyłanie znaków danych o dowolnej kombinacji bitów
BEL (znak wywołujący sygnał akustyczny; ang. <i>bell</i>): używany, gdy istnieje potrzeba zwrócenia uwagi obsługi; może sterować alarmem lub przyrządami zwracającymi uwagę	DC1, DC2, DC3 i DC4 (znaki sterowania przyrządami; ang. <i>device controls</i>): znaki sterowania przyrządami pomocniczymi lub specjalnymi właściwościami terminali
SO (przełączenie na zestaw niestandardowy; ang. <i>shift out</i>): wskazuje, że następne kombinacje kodowe powinny być interpretowane jako niezgodne z zestawem standardowym, aż do wystąpienia znaku SI	CAN (kasowanie; ang. <i>cancel</i>): wskazuje, że dane poprzedzające ten znak w komunikacie lub w bloku powinny być zaniebane (zwykle z powodu wykrycia błędu)
SI (przełączenie na zestaw standardowy; ang. <i>shift in</i>): wskazuje, że następne kombinacje kodowe powinny być interpretowane zgodnie z zestawem standardowym	EM (fizyczne zakończenie nośnika; ang. <i>end of medium</i>): wskazuje na fizyczne zakończenie karty, taśmy lub innego nośnika, czy też zakończenie wymaganej lub używanej części nośnika
DEL (usuwanie; ang. <i>delete</i>): używany do usuwania niepożądanych znaków (np. przez dziurkowanie każdej pozycji bitowej na taśmie papierowej)	SUB (zastąpienie; ang. <i>substitute</i>): wstawiany zamiast znaku, który okazał się błędny lub nieważny
SP (odstęp; ang. <i>space</i>): znak niedrukowany, służący do oddzielania słów lub do przesuwania mechanizmu drukującego albo kursora o jedną pozycję do przodu	ESC (znak unikowy; ang. <i>escape</i>): znak umożliwiający rozszerzenie kodu w ten sposób, że nadaje określonej liczbie następnych znaków zmienione znaczenie

Napęd dysku

Napęd dysku zawiera układy elektroniczne służące do wymiany danych oraz sygnałów sterowania i stanu z modułem wejścia-wyjścia, a także układy elektroniczne sterowania mechanizmem odczytu i zapisu. W przypadku dysku z głowicą nieruchomą przetwornik może wykonać konwersję między wzorami magnetycznymi występującymi na poruszającej się powierzchni dysku a bitami w buforze urządzenia (rys. 6.2). Napęd dysku z ruchomą głowicą musi również móc sterować radialnym przesuwaniem ramienia nad powierzchnią dysku.

62

MODUŁY WEJŚCIA-WYJŚCIA

Działanie modułu

Moduł wejścia-wyjścia jest składnikiem komputera odpowiedzialnym za sterowanie jednym lub wieloma urządzeniami zewnętrznymi oraz za wymianę danych między tymi urządzeniami a pamięcią główną i/lub rejestrami procesora. Moduł wejścia-wyjścia musi więc dysponować interfejsem wewnętrznym w stosunku do komputera (z procesorem i pamięcią główną) oraz interfejsem zewnętrznym w stosunku do komputera (z urządzeniem zewnętrznym).

Główne funkcje modułu wejścia-wyjścia (lub wymagania w stosunku do niego) można podzielić na następujące kategorie:

- ☐ sterowanie i taktowanie,
 - a komunikacja z procesorem,
- ☐ komunikacja z urządzeniem,
 - a buforowanie danych,
- ☐ wykrywanie błędów.

W dowolnym przedziale czasu procesor może komunikować się z jednym lub wieloma urządzeniami zewnętrznymi według nieprzewidywalnego schematu, zależnie od potrzeb programu w odniesieniu do wejścia lub wyjścia. Zasoby wewnętrzne, takie jak pamięć główna i magistrala systemowa, mogą być wykorzystywane wspólnie przez różne operacje, w tym operacje wejścia-wyjścia danych. Dlatego właśnie do funkcji wejścia-wyjścia należy *sterowanie* i *taktowanie* mające na celu koordynowanie przepływu informacji między zasobami wewnętrznymi a urządzeniami zewnętrznymi. Na przykład, sterowanie transferem danych z urządzenia zewnętrznego do procesora może być określone następującą sekwencją kroków:

1. Procesor żąda od modułu wejścia-wyjścia sprawdzenia stanu dołączonego urządzenia.
2. Moduł wejścia-wyjścia udziela odpowiedzi o stanie urządzenia.

3. Jeśli urządzenie działa i jest gotowe do transmitowania, procesor zgłasza zapotrzebowanie na przesłanie danych, posługując się rozkazem do modułu wejścia-wyjścia.
4. Moduł wejścia-wyjścia otrzymuje jednostkę danych (np. 8 lub 16 bitów) z urządzenia zewnętrznego.
5. Dane są przenoszone z modułu wejścia-wyjścia do procesora.

Jeśli system wykorzystuje magistralę, to każde z oddziaływań między procesorem a modułem wejścia-wyjścia wymaga jednego lub wielu arbitraży magistralowych.

Z tego uproszczonego scenariusza wynika również, że moduł wejścia-wyjścia musi być zdolny do komunikowania się z procesorem i z urządzeniem zewnętrznym. *Komunikacja z procesorem* obejmuje:

- a *Dekodowanie rozkazu.* Moduł wejścia-wyjścia przyjmuje rozkazy od procesora. Rozkazy te są na ogół wysyłane w postaci sygnałów na magistrali sterowania. Na przykład moduł wejścia-wyjścia napędu dysku może akceptować następujące rozkazy: CZYTAJ SEKTOR, ZAPISZ SEKTOR, ZNAJDŹ numer ścieżki i SKANUJ rekord ID. Każdy z dwóch ostatnich rozkazów zawiera parametr, który jest przesyłany magistralą danych.
- Q *Przesyłanie danych.* Dane są wymieniane między modułem wejścia-wyjścia a procesorem poprzez magistralę danych.
- a *Przesyłanie informacji o stanie.* Ponieważ urządzenia peryferyjne są tak powolne, ważna jest znajomość stanu modułu wejścia-wyjścia. Jeśli na przykład moduł wejścia-wyjścia jest proszony o przesłanie danych do procesora (odczyt), może on nie być gotowy do tego, ponieważ pracuje jeszcze nad poprzednim rozkazem wejścia-wyjścia. Fakt ten może być przedstawiony za pomocą sygnału stanu. Zwykłymi sygnałami stanu są sygnały zajętości i gotowości. Mogą też występować sygnały informujące o różnych warunkach błędu.
- Q *Rozpoznawanie adresu.* Podobnie jak słowa w pamięci, również każde urządzenie wejścia-wyjścia ma swój adres. Moduł wejścia-wyjścia musi więc rozpoznawać unikatowy adres każdego spośród urządzeń peryferyjnych, którym steruje.

Moduł wejścia-wyjścia musi również *komunikować się* z urządzeniem. Komunikacja ta obejmuje rozkazy, informacje o stanie i dane (rys. 6.2).

Podstawowym zadaniem modułu wejścia-wyjścia jest *buforowanie danych*. Potrzeba tej funkcji jasno wynika z tabeli 6.2. Podczas gdy szybkość transferu danych z (i do) pamięci głównej oraz procesora jest całkiem duża, w przypadku większości urządzeń peryferyjnych jest ona o rzędy wielkości mniejsza. Dane nadchodzące z pamięci głównej są przesyłane do modułu wejścia-wyjścia w postaci zwartego pakietu. Są one buforowane w module wejścia-wyjścia, a następnie wysyłane do urządzenia peryferyjnego z szybkością dostosowaną do tego urządzenia. Przy przesyłaniu w przeciwnym kierunku dane są buforowane, żeby

nie angażować pamięci w operację powolnego transferu. Moduł wejścia-wyjścia musi więc móc pracować zarówno z szybkością urządzenia, jak i z szybkością pamięci.

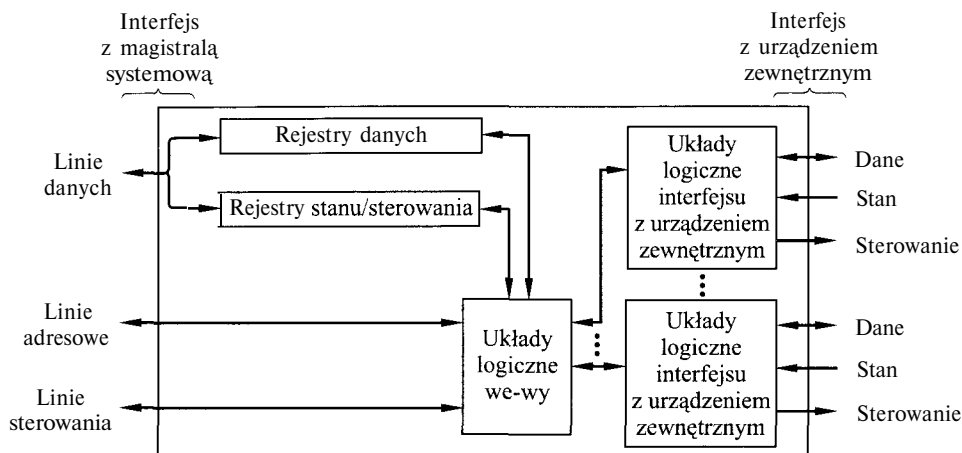
TABELA 6.2. Klasyfikacja przykładowych urządzeń wejścia-wyjścia [HENN90]

Urządzenie	Rodzaj	Partner	Szybkość transferu danych [KB/s]
Klawiatura	wejściowe	człowiek	0,01
Myszka	wejściowe	człowiek	0,02
Wejście głosowe	wejściowe	człowiek	0,02
Skaner	wejściowe	człowiek	200
Wyjście głosowe	wyjściowe	człowiek	0,6
Drukarka wierszowa	wyjściowe	człowiek	1
Drukarka laserowa	wyjściowe	człowiek	100
Monitor graficzny	wyjściowe	człowiek	30 000
Procesor - bufor ramki	wyjściowe	komputer	200
Sieć – terminal	wejściowe lub wyjściowe	komputer	0,05
Sieć – LAN	wejściowe lub wyjściowe	komputer	200
Dysk optyczny	pamięć	komputer	500
Taśma magnetyczna	pamięć	komputer	2 000
Dysk magnetyczny	pamięć	komputer	2 000

Moduł wejścia-wyjścia jest często odpowiedzialny za *wykrywanie błędów* oraz za informowanie o nich procesora. Jedna z klas błędów obejmuje defekty mechaniczne i elektryczne występujące w urządzeniu (np. uszkodzenie papieru, niewłaściwa ścieżka dysku). Do innej klasy należą niezamierzone zmiany wzoru bitowego danych transmitowanych z urządzenia do modułu wejścia-wyjścia. Do wykrywania błędów transmisji często jest używana pewna postać kodu detekcyjnego. Typowym przykładem jest używanie bitu parzystości towarzyszącego każdemu znakowi danych. Na przykład kod znaku ASCII zajmuje 7 bitów w bajcie. Bit ósmy jest ustalany na podstawie tego, czy suma wszystkich jedynek w bajcie jest parzysta, czy nieparzysta. Po odebraniu bajta moduł wejścia-wyjścia sprawdza parzystość w celu stwierdzenia, czy wystąpił błąd.

Struktura modułu wejścia-wyjścia

Moduły wejścia-wyjścia różnią się znacznie pod względem złożoności oraz liczby kontrolowanych urządzeń zewnętrznych. Możemy tu przedstawić jedynie bardzo ogólny opis (szczegółne urządzenie, układ Intel 8255A, jest opisane w podrozdz. 6.4). Na rysunku 6.3 jest przedstawiony ogólny schemat blokowy modułu wejścia-wyjścia. Moduł jest połączony z resztą komputera za pomocą zespołu linii sygnałowych (np. magistrali systemowej). Dane przenoszone do (i z) modułu są buforowane w jednym lub w wielu rejestrach danych. Może też występować jeden lub wiele rejestrów stanu, które dostarczają bieżącej informacji o stanie. Rejestr stanu może również funkcjonować jako rejestr sterowania,



RYSUNEK 6.3. Schemat blokowy modułu wejścia-wyjścia

akceptujący szczegółową informację sterowania pochodzącą z procesora. Układy logiczne wewnątrz modułu współpracują z procesorem poprzez zespół linii sterowania. Są one wykorzystywane przez procesor do wydawania rozkazów modułowi wejścia-wyjścia. Niektóre linie sterowania mogą być używane przez moduł wejścia-wyjścia (np. do przekazywania sygnałów arbitrażowych oraz określających stan). Moduł musi również rozpoznawać i generować adresy urządzeń, którymi steruje. Każdy moduł wejścia-wyjścia ma unikatowy adres lub, jeśli kontroluje więcej niż jedno urządzenie zewnętrzne, unikatowy zespół adresów. Moduł wejścia-wyjścia zawiera też układy logiczne dostosowane do interfejsów z każdym sterowanym urządzeniem.

Moduł wejścia-wyjścia funkcjonuje tak, żeby procesor mógł postrzegać szeroki zakres różnorodnych urządzeń zewnętrznych w prosty sposób. Istnieje wiele możliwości, którymi może dysponować moduł. Moduł wejścia-wyjścia może ukrywać szczegóły dotyczące taktowania, formatowania i działania elektromechanicznego urządzenia zewnętrznego, dzięki czemu procesor może się posługiwać prostymi rozkazami odczytu i zapisu, ewentualnie otwarcia i zamknięcia pliku. W najprostszej formie moduł wejścia-wyjścia może pozostawić wiele pracy dotyczącej sterowania urządzeniem (np. przewijanie taśmy) w gestii procesora.

Moduł wejścia-wyjścia, który przejmuję większość obciążenia szczegółowym przetwarzaniem, mający wysoki poziom priorytetu w stosunku do procesora, jest zwykle określany jako **kanał wejścia-wyjścia lub procesor wejścia-wyjścia**. Moduł wejścia-wyjścia, który jest całkiem prymitywny i wymaga szczegółowego sterowania, jest zwykle określany jako **sterownik wejścia-wyjścia lub sterownik urządzenia**. Sterowniki wejścia-wyjścia powszechnie występują w mikrokomputerach, podczas gdy kanały wejścia-wyjścia są wykorzystywane w dużych komputerach, natomiast w minikomputerach występuje mieszaninamodulów.

W dalszym ciągu będziemy stosowali termin **moduł wejścia-wyjścia** nie wywołujący nieporozumień, wykorzystując bardziej specyficzne określenia, jeśli będzie to konieczne.

63

PROGRAMOWANE WEJŚCIE-WYJŚCIE

Istnieją trzy sposoby realizacji operacji wejścia-wyjścia. W przypadku **programowanego wejścia-wyjścia**, dane są wymieniane między procesorem a modułem wejścia-wyjścia. Procesor wykonuje program, który umożliwia mu bezpośrednie sterowanie operacją wejścia-wyjścia, włącznie z rozpoznawaniem stanu urządzenia, wysyłaniem rozkazu odczytu lub zapisu oraz transferem danych. Gdy procesor wydaje rozkaz modułowi wejścia-wyjścia, musi poczekać na zakończenie operacji wejścia-wyjścia. Jeśli procesor jest szybszy niż moduł wejścia-wyjścia, oznacza to stratę czasu procesora. W przypadku **wejścia-wyjścia sterowanego przerwaniem** procesor wydaje rozkaz wejścia-wyjścia, po czym wykonuje inne rozkazy, co z kolei ulega przerwaniu przez moduł wejścia-wyjścia, gdy zakończył on swoją pracę. Zarówno w przypadku programowanego wejścia-wyjścia, jak i sterowanego przerwaniem procesor jest odpowiedzialny za pobieranie danych z pamięci głównej (wyjście) oraz zapisywanie ich w tej pamięci (wejście). Alternatywne rozwiązanie to **bezpośredni dostęp do pamięci** (ang. *direct memory access* - DMA). W tym trybie moduł wejścia-wyjścia i pamięć główna wymieniają dane bezpośrednio, bez angażowania procesora.

W tabeli 6.3 są podane zależności dotyczące tych trzech metod. W tym podrozdziale przeanalizujemy programowane wejście-wyjście. Wejście-wyjście sterowane przerwaniem oraz DMA zostaną rozpatrzone odpowiednio w następnych dwóch podrozdziałach.

TABELA 6.3. Sposoby działania wejścia-wyjścia

	Brak przerwania	Zastosowanie przerwania
Transfer wejście-wyjście - pamięć poprzez procesor	programowane wejście-wyjście	wejście-wyjście sterowane przerwaniem
Bezpośredni transfer wejście-wyjście - pamięć		bezpośredni dostęp do pamięci (DMA)

Przegląd

Gdy procesor realizuje program i napotyka instrukcję odnoszącą się do wejścia-wyjścia, wykonuje tę instrukcję przez wydanie rozkazu odpowiedniemu modułowi wejścia-wyjścia. W przypadku programowanego wejścia-wyjścia moduł wejścia-wyjścia wykona wymagane działanie, a następnie ustawi odpowiednie bity w rejestrze stanu wejścia-wyjścia (rys. 6.3). Moduł wejścia-wyjścia nie po-

dejmując dalszych działań alarmujących procesor. W szczególności nie przerywa pracy procesora. Tak więc do procesora należy okresowe sprawdzanie stanu modułu wejścia-wyjścia i stwierdzenie, że operacja została zakończona.

Aby wyjaśnić działanie programowanego wejścia-wyjścia, rozpatrzmy je najpierw z punktu widzenia rozkazów wejścia-wyjścia wydawanych przez procesor modułowi wejścia-wyjścia, a następnie z punktu widzenia instrukcji wejścia-wyjścia wykonywanych przez procesor.

Rozkazy wejścia-wyjścia

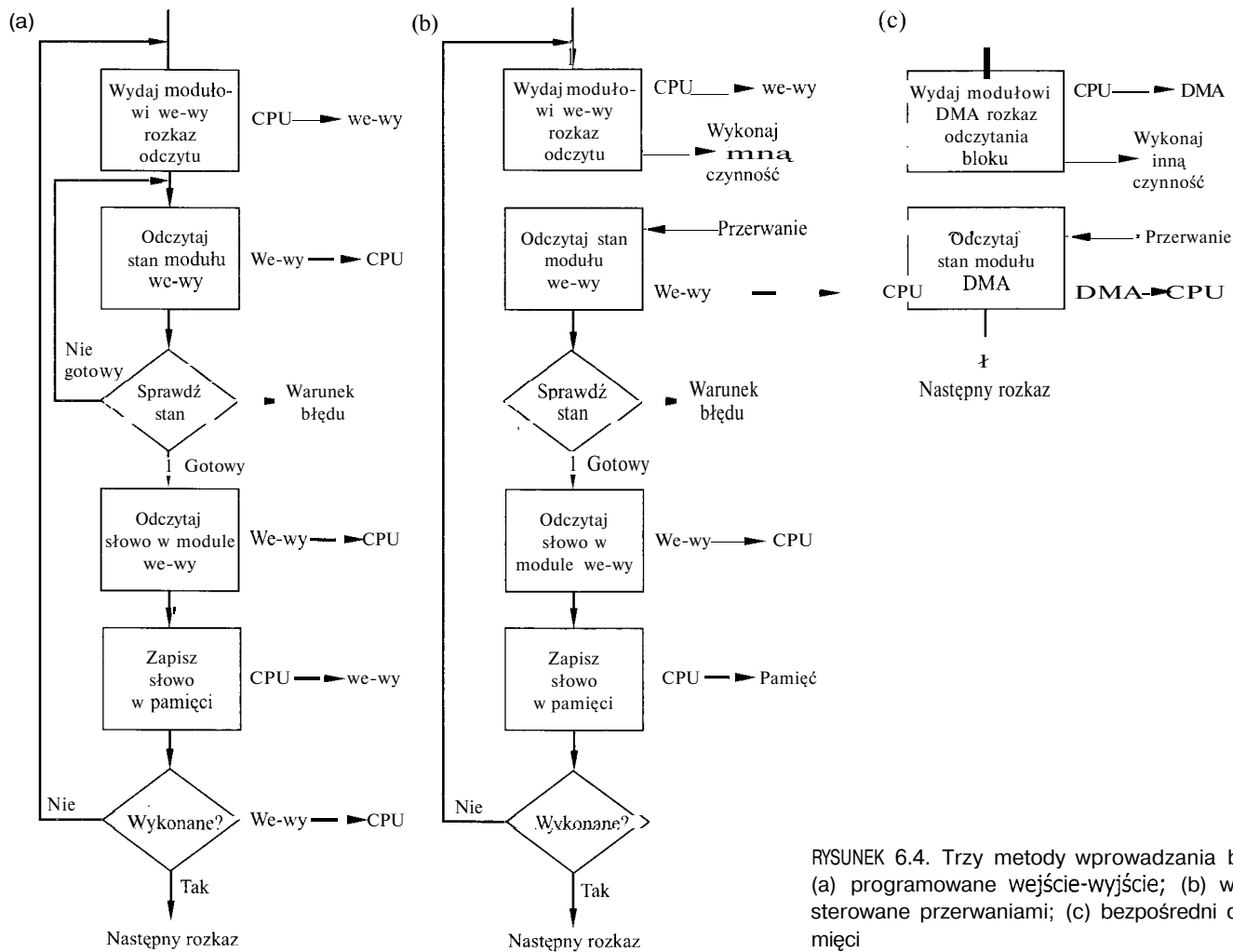
W celu wykonania instrukcji odnoszącej się do wejścia-wyjścia procesor podaje adres określający moduł wejścia-wyjścia i urządzenie zewnętrzne oraz rozkaz wejścia-wyjścia. Istnieją cztery rodzaje rozkazów wejścia-wyjścia, które może otrzymać moduł wejścia-wyjścia adresowany przez procesor. Są one sklasyfikowane jako sterowanie, test, odczyt i zapis.

Rozkaz **sterowania** jest stosowany w celu aktywowania (uruchomienia) urządzenia peryferyjnego i przekazania mu, co ma robić. Na przykład pamięć taśmowa może otrzymać rozkaz przewinięcia lub przesunięcia się do przodu o jeden rekord. Rozkazy te są dopasowane do określonego typu urządzenia peryferyjnego.

Rozkaz **testowania** jest stosowany w celu zbadania różnych warunków stanu związanych z modułem wejścia-wyjścia i jego urządzeniami peryferyjnymi. Procesor może chcieć wiedzieć, czy potrzebne urządzenie peryferyjne jest zasilane i dostępne do wykorzystania. Może także chcieć wiedzieć, czy ostatnia operacja wejścia-wyjścia została zakończona i czy wystąpiły jakieś błędy.

Rozkaz **odczytu** skłania moduł wejścia-wyjścia do pobrania danych z urządzenia peryferyjnego oraz umieszczenia ich w wewnętrznym buforze (określonym na rys. 6.3 jako rejestr danych). Procesor może następnie uzyskać dane, domagając się, żeby moduł wejścia-wyjścia umieścił je na szynie danych. Odwrotnie, rozkaz **zapisu** zmusza moduł wejścia-wyjścia do pobrania danych (bajta lub słowa) z szyny danych i następnie do przekazania ich do urządzenia peryferyjnego.

Na rysunku 6.4a jest pokazany przykład użycia programowanego wejścia-wyjścia do wczytania bloku danych z urządzenia peryferyjnego (np. rekordu z taśmy) do pamięci. Dane są wczytywane słowo (np. 16 bitów) po słowie. Po wczytaniu każdego słowa procesor musi pozostawać w cyklu sprawdzania stanu do momentu, aż stwierdzi, że słowo jest osiągalne w rejestrze danych modułu wejścia-wyjścia. Z tego opisu wynika główna wada tej techniki: jest to proces czasochłonny, angażujący bez potrzeby procesor.



RYSUNEK 6.4. Trzy metody wprowadzania bloku danych: (a) programowane wejście-wyjście; (b) wejście-wyjście sterowane przerwaniem; (c) bezpośredni dostęp do pamięci

Instrukcje wejścia-wyjścia

W przypadku programowanego wejścia-wyjścia występuje ścisła odpowiedniość między instrukcjami dotyczącymi wejścia-wyjścia pobieranymi przez procesor z pamięci a rozkazami wejścia-wyjścia wydawanymi przez procesor modułowi wejścia-wyjścia w celu wykonania instrukcji. To znaczy, że instrukcje są z łańcuchowością odwzorowywane na rozkazy wejścia-wyjścia i często występuje prosta zależność jeden do jednego. Forma instrukcji zależy od sposobu adresowania urządzenia zewnętrznego.

Zwykle mamy do czynienia z wieloma urządzeniami wejścia-wyjścia połączonymi z systemem poprzez moduły wejścia-wyjścia. Każde urządzenie otrzymuje unikatowy identyfikator lub adres. Gdy procesor wydaje rozkaz wejścia-wyjścia, zawiera on adres potrzebnego urządzenia. Wobec tego każdy moduł wejścia-wyjścia musi interpretować linie adresowe w celu stwierdzenia, czy rozkaz go dotyczy.

Jeśli procesor, pamięć główna oraz wejście-wyjście używają wspólnej magistrali, możliwe są dwa tryby adresowania: odwzorowany w pamięci i izolowany. W przypadku wejścia-wyjścia **odwzorowanego w pamięci** ta sama przestrzeń adresowa jest przeznaczona dla komórek pamięci i urządzeń wejścia-wyjścia. Procesor traktuje rejestry stanu i rejestry danych modułów wejścia-wyjścia jako komórki pamięci i wykorzystuje takie same instrukcje maszynowe w celu uzyskania dostępu zarówno do pamięci, jak i do urządzeń wejścia-wyjścia. Na przykład, za pomocą 10 linii adresu możliwe jest obsługiwanie łącznie 1024 komórek pamięci i adresów wejścia-wyjścia, w dowolnej kombinacji.

W przypadku wejścia-wyjścia odwzorowanego w pamięci w magistrali potrzebna jest tylko jedna linia odczytu i jedna linia zapisu. Alternatywnie, magistrala może być wyposażona w linie odczytu i zapisu pamięci oraz linie rozkazów wejścia i wyjścia. Wówczas linia rozkazu określa, czy adres odnosi się do komórki pamięci, czy do urządzenia wejścia-wyjścia. Pełny zakres adresów może być osiągalny dla obu. Za pomocą 10 linii adresu system może teraz obsłużyć zarówno 1024 komórki pamięci, jak i 1024 adresy wejścia-wyjścia. Ponieważ przestrzeń adresowa wejścia-wyjścia jest odizolowana od przestrzeni adresowej pamięci, rozwiązanie takie jest określane jako **izolowane wejście-wyjście**.

Na rysunku 6.5 są przedstawione te dwie techniki programowanego wejścia-wyjścia. Na rysunku 6.5a widać, jak może postrzegać programista interfejs prostego urządzenia wejściowego (np. klawiatury) w przypadku wejścia-wyjścia odwzorowanego w pamięci. Załóżmy, że mamy adres 10-bitowy, z 512-bitową pamięcią (komórki 0+511) i możliwymi 512 adresami wejścia-wyjścia (komórki 512+1023). Dwa adresy są przypisane wejściu z klawiatury określonego terminala. Adres 516 dotyczy rejestru danych a adres 517 rejestru stanu, który działa również jako rejestr sterowania odbierający rozkazy procesora. Pokazany program powoduje wczytanie 1 bajta danych z klawiatury do rejestru akumulatora



(a)

ADRES	ROZKAZ	ARGUMENT	KOMENTARZ
200	Load AC	"1"	
	Store AC	517	Inicjuj odczyt klawiatury
202	Load AC	517	Uzyskaj bajt stanu
	Branch if Sign = 0	202	Wykonuj pętlę, aż do osiągnięcia gotowości
	Load AC	516	Ładuj bajt danych

(b)

ADRES	ROZKAZ	ARGUMENT	KOMENTARZ
200	Start I/O	5	Inicjuj odczyt klawiatury
201	Test I/O	5	Sprawdź zakończenie operacji
	Branch not ready	201	Wykonuj pętlę, aż do zakończenia operacji
	In	5	Ładuj bajt danych

RYSUNEK 6.5. Wejście-wyjście odwzorowane w pamięci (a) oraz izolowane (b)

w procesorze. Zwróćmy uwagę, że procesor wykonuje pętlę do momentu, w którym bajt danych staje się osiągalny.

W przypadku izolowanego wejścia-wyjścia (rys. 6.5b) porty wejścia-wyjścia są dostępne tylko dla specjalnych rozkazów wejścia-wyjścia, które aktywują linie rozkazów wejścia-wyjścia magistrali.

W większości typów procesorów występuje stosunkowo duży zestaw różnych instrukcji dotyczących pamięci. Jeśli wykorzystuje się izolowane wejście-wyjście, to mamy do czynienia tylko z niewieloma instrukcjami wejścia-wyjścia. Wobec tego zaletą wejścia-wyjścia odwzorowanego w pamięci jest możliwość korzystania z obszernego zbioru instrukcji, co pozwala na wydajniejsze programowanie. Wadą jest zużywanie cennej przestrzeni adresowej pamięci. Powszechnie wykorzystuje się zarówno wejście-wyjście odwzorowane w pamięci, jak i izolowane.

6.4

WEJŚCIE-WYJŚCIE STEROWANE PRZERWANIAM

W przypadku programowanego wejścia-wyjścia problemem jest to, że procesor musi długo czekać, aż potrzebny moduł wejścia-wyjścia będzie gotowy do odbioru lub transmisji danych. Podczas oczekiwania procesor musi powtarzać badanie stanu modułu wejścia-wyjścia. W rezultacie wydajność całego systemu ulega poważnej degradacji.

Alternatywą dla procesora jest wydanie modułowi rozkazu wejścia-wyjścia, a następnie przejście do innej użytecznej pracy. Moduł wejścia-wyjścia może przerwać pracę procesora żądaniem obsługi, gdy jest już gotów do wymiany z nim danych. Wówczas procesor dokonuje transferu danych, po czym wraca do poprzedniego przetwarzania.

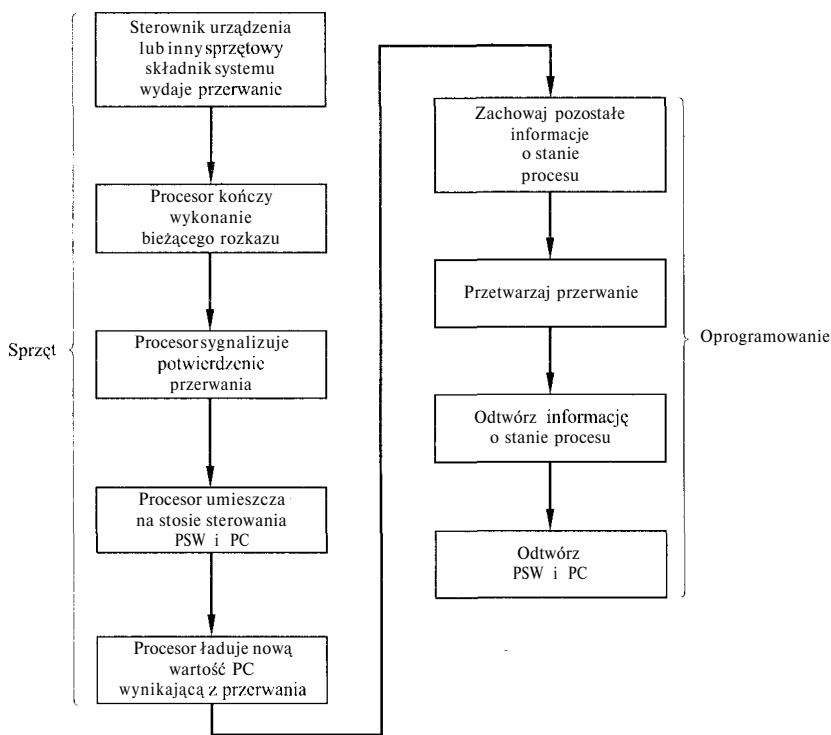
Rozważmy, jak to funkcjonuje - najpierw z punktu widzenia modułu wejścia-wyjścia. W przypadku wejścia moduł wejścia-wyjścia otrzymuje od procesora rozkaz CZYTAJ. Następnie moduł wejścia-wyjścia przystępuje do odczytania danych ze stowarzyszonego urządzenia peryferyjnego. Gdy dane znajdują się już w rejestrze danych modułu, sygnalizuje on procesorowi przerwanie poprzez linię sterowania. Następnie moduł czeka, aż procesor zażąda danych. Gdy żądanie zostanie zgłoszone, moduł umieszcza dane na magistrali danych, po czym jest gotowy do innej operacji wejścia-wyjścia.

Z punktu widzenia procesora działanie w przypadku wejścia jest następujące. Procesor wydaje rozkaz CZYTAJ. Następnie przechodzi do innej czynności (może on jednocześnie realizować kilka różnych programów). Na końcu każdego cyklu rozkazu procesor sprawdza, czy nie nastąpiło przerwanie (rys. 3.9). Gdy następuje przerwanie ze strony modułu wejścia-wyjścia, procesor zachowuje kontekst bieżącego programu (np. zawartości licznika programu i rejestrów procesora) i przetwarza przerwanie. W tym przypadku procesor odczytuje słowo danych z modułu wejścia-wyjścia i kieruje je do pamięci. Następnie odnawia kontekst programu, nad którym pracował (lub jakiś inny program), i wznowia pracę.

Na rysunku 6.4b jest pokazane wykorzystanie wejścia-wyjścia sterowanego przerwaniem do wczytania bloku danych. Porównajmy to z rys. 6.4a. Wejście-wyjście sterowane przerwaniem jest wydajniejsze niż programowane, ponieważ jest tu eliminowane zbędne oczekiwanie. Jednak wejście-wyjście sterowane przerwaniem nadal zużywa dużo czasu procesora, ponieważ każde słowo danych przechodzące z pamięci do modułu wejścia-wyjścia lub z modułu wejścia-wyjścia do pamięci musi przechodzić przez procesor.

Przetwarzanie przerw

Rozważmy bardziej szczegółowo rolę procesora w operacjach wejścia-wyjścia. Wystąpienie przerwania wyzwała szereg zdarzeń zarówno w sprzęcie, jak i w programach. Typową sekwencję zdarzeń widać na rys. 6.6. Gdy urządzenie



RYСУNEK 6.6. Proste przetwarzanie przerw

wejścia-wyjścia kończy operację wejścia-wyjścia, ma miejsce następująca sekwencja zdarzeń sprzętowych:

1. Urządzenie wysyła do procesora sygnał przerwania.
2. Procesor kończy wykonywanie bieżącego rozkazu przed odpowiedzią na przerwanie, co widać na rys. 3.9.
3. Procesor sprawdza, czy nastąpiło przerwanie, stwierdza jego wystąpienie i wysyła sygnał potwierdzenia do urządzenia, które zażądało przerwania. Potwierdzenie to pozwala urządzeniu na usunięcie swojego sygnału przerwania.
4. Procesor musi się teraz przygotować do przeniesienia sterowania do procedury przerwania. Najpierw musi zachować informację niezbędną do wznowienia bieżącego programu w punkcie jego przerwania. Minimalną wymaganą informacją jest (a) stan procesora, zawarty w rejestrze określanym jako słowo stanu programu (ang. *program status word* - PSW), oraz (b) lokalizacja następnego rozkazu przewidzianego do wykonania, która jest zawarta w liczniku programu. Informacja ta może być przekazana na stos sterowania systemu².

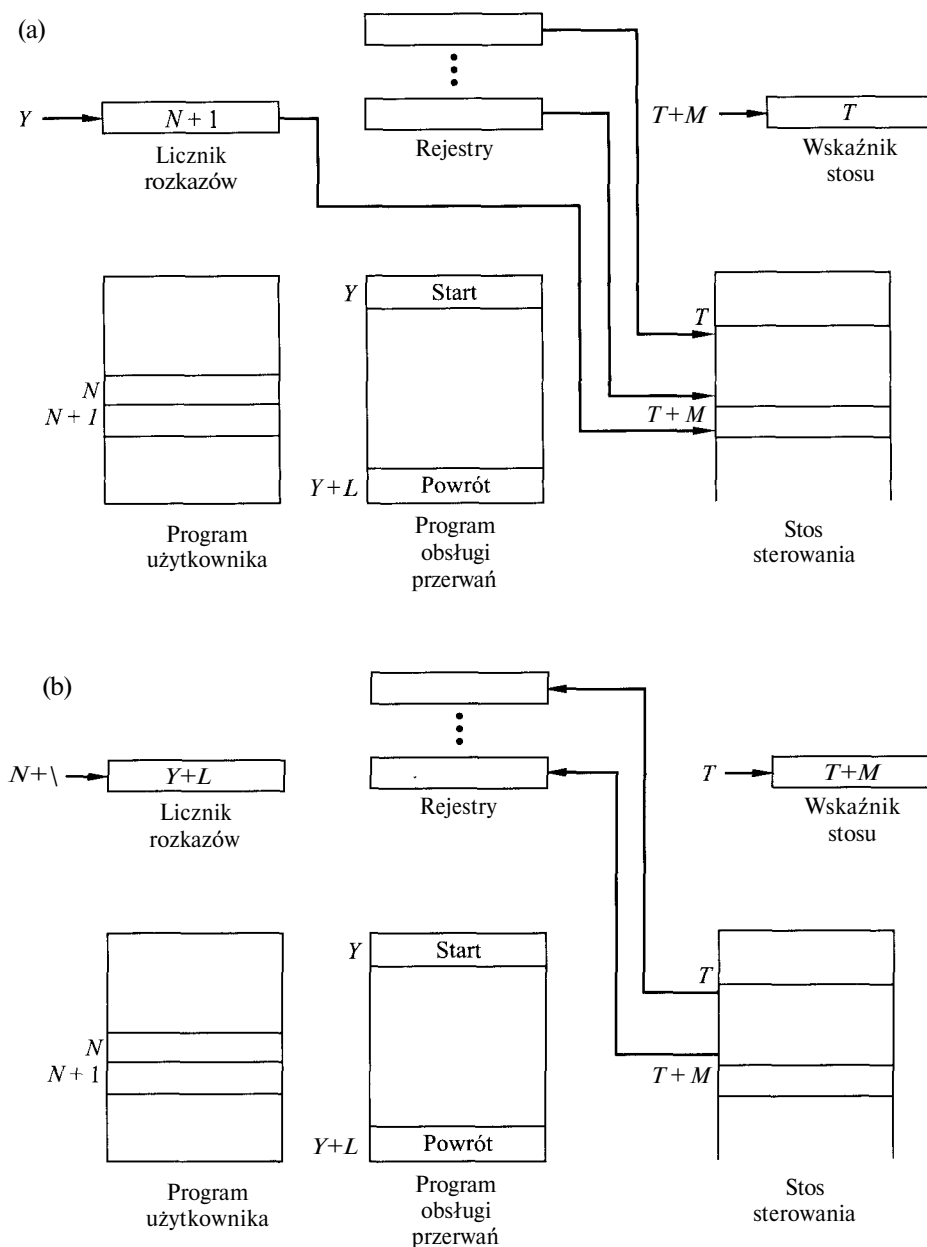
² Działanie stosu jest opisane w dodatku 9A.

5. Procesor ładuje teraz do licznika rozkazów pierwszą pozycję programu obsługi przerw, który odpowiada zgłoszonemu przerwaniu. Zależnie od architektury komputera i rozwiązania systemu operacyjnego może to być pojedynczy program, jeden dla każdego rodzaju przerwania, lub jeden dla każdego urządzenia i każdego typu przerwania. Jeśli występuje więcej niż jedna procedura obsługi przerwania, to procesor musi zdecydować, którą należy uruchomić. Informacja ta może być zawarta w oryginalnym sygnale przerwania lub też procesor może być zmuszony zwrócić się do urządzenia zgłaszającego przerwanie, aby w odpowiedzi uzyskać tę informację.

Gdy licznik rozkazów został załadowany, procesor przechodzi do następnego cyklu rozkazu, który rozpoczyna się od pobrania rozkazu. Ponieważ pobieranie rozkazów jest zdeterminowane przez zawartość licznika rozkazów, w wyniku tego sterowanie zostaje przeniesione do programu obsługi przerw. Wynikiem realizowania tego programu są następujące operacje:

6. W tym punkcie zawartość licznika rozkazów oraz słowo stanu programu związane z przerwanym programem zostały zachowane na stosie systemowym. Są jednak również i inne informacje, które mogą być traktowane jako część „stanu” wykonywanego programu. W szczególności zachowania wymaga zawartość rejestrów procesora, ponieważ rejestry te mogą być wykorzystywane przez program obsługi przerw. Wszystkie te wartości oraz inne informacje o stanie muszą być zachowane. Zwykle działanie programu obsługi przerw rozpoczyna się od zapisania na stosie zawartości wszystkich rejestrów. Prosty przykład jest pokazany na rys. 6.7a. W tym przypadku program użytkownika został przerwany po instrukcji zawartej w pozycji N . Zawartość wszystkich rejestrów oraz adres następnego rozkazu ($N + 1$) są umieszczane na stosie. Wskaźnik wierzchołka stosu jest aktualizowany, aby wskazywał nowy wierzchołek, a licznik rozkazów jest aktualizowany, aby wskazywał początek procedury obsługi przerwania.
7. Program obsługi przerwania może teraz przystąpić do przetwarzania przerwania. Obejmuje to zbadanie informacji stanu odnoszącej się do operacji wejścia-wyjścia lub innego zdarzenia, które spowodowało przerwanie. Może także obejmować wysłanie dodatkowych rozkazów lub potwierdzeń do urządzenia wejścia-wyjścia.
8. Gdy przetwarzanie przerwania jest zakończone, zachowane wartości rejestrów są pobierane ze stosu i wprowadzane do rejestrów (patrz np. rys. 6.7b).
9. Ostatnim działaniem jest odnowienie słowa stanu programu i licznika rozkazów na podstawie wartości ze stosu. W rezultacie następny rozkaz do realizacji będzie pochodził z uprzednio przerwanego programu.

Zauważmy, że ważne jest zachowanie wszystkich informacji stanu dotyczących przerwanego programu w celu jego późniejszego przywrócenia. Dzieje się tak dlatego, że przerwanie nie jest procedurą wywołaną z tego programu. Prze-



RYSUNEK 6.7. Zmiany w pamięci i w rejestrach wynikające z przerwania: (a) przerwanie następuje po rozkazie znajdującym się w lokacji N ; (b) powrót z przerwania

ranie może wystąpić w dowolnym czasie, a więc w dowolnym punkcie realizacji programu użytkownika. Jego wystąpienie jest nieprzewidywalne. Jak zobaczymy w następnym rozdziale, oba te programy mogą nie mieć nic wspólnego i mogą należeć do dwóch różnych użytkowników.

Problemy projektowania

Przy wdrażaniu wejścia-wyjścia sterowanego przerwaniem występują dwa problemy projektowe. Po pierwsze, jeżeli prawie zawsze występuje wiele modułów wejścia-wyjścia, to w jaki sposób procesor określa, które urządzenie wysłało przerwanie? Po drugie, jeśli wystąpiło wiele przerw, to w jaki sposób procesor decyduje, które z nich ma być przetwarzane?

Rozważmy najpierw sposób identyfikacji urządzenia. Wykorzystywane są zwykle cztery kategorie metod:

- a wiele linii przerw, a
- a odpytywanie za pomocą oprogramowania,
- odpytywanie za pomocą sprzętu (łańcuchowe, wektorowe), a
- a arbitraż za pomocą magistrali (wektorowy).

Najprostszym podejściem do problemu jest wprowadzenie *wielu linii przerw* między procesorem a modułami wejścia-wyjścia. Jest jednak niepraktyczne poświęcanie więcej niż kilku linii magistrali lub końcówek procesora na linie przerw. W rezultacie, nawet jeśli występuje wiele linii przerw, to jest prawdopodobne, że do każdej linii będzie dołączonych wiele modułów wejścia-wyjścia. Wobec tego w odniesieniu do każdej linii będzie musiała być zastosowana jedna z trzech pozostałych metod.

Jedną z możliwości jest **odpytywanie za pomocą oprogramowania**. Gdy procesor wykrywa przerwanie, przeskakuje do procedury obsługi przerwania, której zadaniem jest odpytanie każdego modułu wejścia-wyjścia w celu stwierdzenia, który moduł spowodował przerwanie. Odpytywanie może być przeprowadzone za pomocą oddzielnej linii sterowania (np. TEST I/O). W tym przypadku procesor wzbudza TEST I/O i umieszcza adres określonego modułu wejścia-wyjścia na liniach adresu. Moduł wejścia-wyjścia odpowiada pozytywnie, jeśli wysłał przerwanie. Alternatywnie, każdy moduł wejścia-wyjścia może zawierać adresowalny rejestr stanu. Procesor odczytuje wtedy rejestr stanu każdego modułu wejścia-wyjścia w celu zidentyfikowania modułu przerywającego. Gdy właściwy moduł jest zidentyfikowany, procesor przeskakuje do procedury obsługi urządzenia właściwej dla tego urządzenia.

Wadą odpytywania za pomocą oprogramowania jest jego czasochłonność. Bardziej wydajną metodą jest wykorzystanie rozwiązania **łańcuchowego**, które w istocie jest odpytywaniem za pomocą sprzętu. Przykład konfiguracji łańcuchowej widać na rys. 3.30. Wszystkie moduły wejścia-wyjścia są połączone ze wspólną linią żądania przerwania. Linia potwierdzenia przerwania przechodzi

łańcuchowo przez moduły. Gdy procesor wykrywa przerwanie, wysyła potwierdzenie przerwania. Sygnał ten przechodzi przez kolejne moduły wejścia-wyjścia, aż natrafi na moduł żądający przerwania. Moduł ten zwykle odpowiada przez umieszczenie słowa na liniach danych. Słowo to jest określane jako wektor i jest albo adresem, albo innym unikatowym identyfikatorem modułu wejścia-wyjścia. W każdym przypadku procesor wykorzystuje wektor jako znacznik procedury obsługi odpowiedniego urządzenia. Zapobiega to konieczności wykonywania najpierw ogólnej procedury obsługi przerwania. Metoda ta jest nazywana **przerwaniem wektorowym**.

Istnieje jeszcze inna metoda wykorzystująca przerwanie wektorowe. Jest nią **arbitraż za pomocą magistrali**. W tym przypadku zanim moduł wejścia-wyjścia pobudzi linię zapotrzebowania na przerwanie, musi najpierw przejąć sterowanie magistralą. Wobec tego w określonym momencie tylko jeden moduł może pobudzić linię. Gdy procesor wykrywa przerwanie, odpowiada, posługując się linią potwierdzenia przerwania. Moduł żądający przerwania umieszcza następnie swój wektor na liniach danych.

Wymienione wyżej metody służą do identyfikacji modułu żądającego przerwania. Umożliwiają one również przypisywanie priorytetów, jeśli więcej niż jedno urządzenie domaga się obsługi przerwania. W przypadku wielu linii procesor po prostu wybiera linię przerwania o najwyższym priorytecie. Przy wykorzystaniu odpytywania za pomocą oprogramowania priorytet wynika z kolejności odpytywania modułów. Podobnie kolejność modułów w łańcuchu wyznacza ich priorytet. Arbitraż za pomocą magistrali może wreszcie zawierać schemat arbitrażu opisany w podrozdz. 3.4.

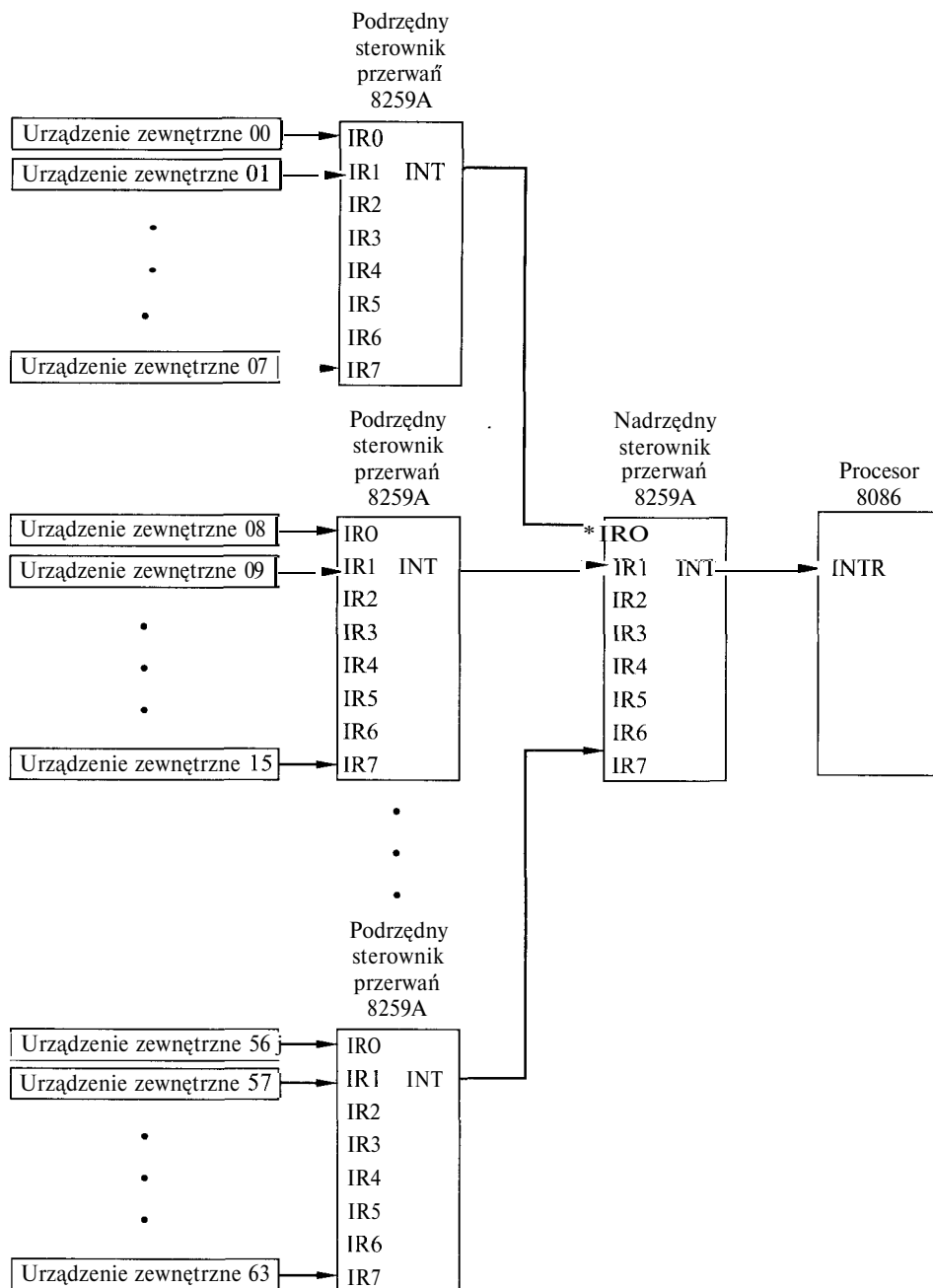
Rozpatrzmy obecnie dwa przykłady struktur wykorzystujących przerwanie.

Sterownik przerwania Intel 8259A

Procesor Intel 8086 współpracuje z jedną linią żądania przerwania (INTR) i z jedną linią potwierdzania przerwania (INTA). Aby umożliwić procesorowi 8086 elastyczne współdziałanie z różnymi urządzeniami i strukturami priorytetów, konfiguruje się go zwykle z zewnętrznym sterownikiem przerwania - 8259A. Urządzenia zewnętrzne są łączone z układem 8259A, który z kolei jest połączony z procesorem 8086.

Rysunek 6.8 ilustruje wykorzystanie układu 8259A do łączenia wielu modułów wejścia-wyjścia z procesorem 8086. Jeden układ 8259A może obsługiwać do 8 modułów. Jeśli wymagane jest sterowanie więcej niż 8 modułów, można stosować układ kaskadowy obsługujący do 64 modułów.

Wyłącznym zadaniem układu 8259A jest zarządzanie przerwaniem. Odbiera on żądanie przerwania od dołączonych modułów, określa przerwanie o najwyższym priorytecie i wysyła sygnał do procesora, wzbudzając linię INTR. Procesor wysyła potwierdzenie przez linię INTA. Skłania to 8259A do umieszczenia odpowiedniego wektora informacji na magistrali danych. Procesor może



RYSUNEK 6.8. Zastosowanie sterownika przerwań 8259A

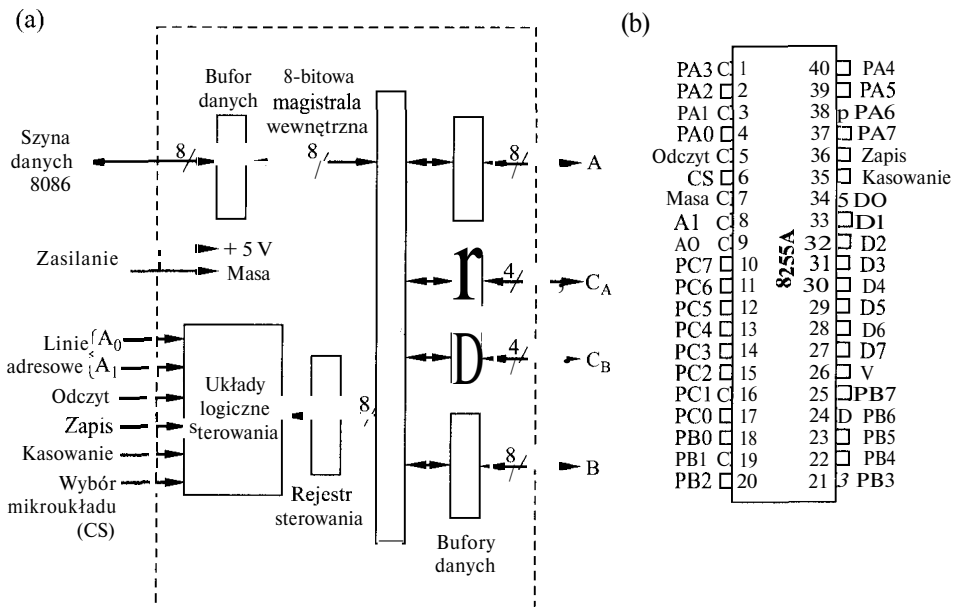
następnie przejść do przetwarzania przerwania i do bezpośredniego komunikowania się z modułem wejścia-wyjścia w celu odczytu lub zapisu danych.

Układ 8259A jest programowalny. Procesor 8086 określa schemat priorytetów, który ma być wykorzystywany przez ustalenie słowa sterowania w sterowniku 8259A. Możliwe są następujące tryby przerwania.

- *W pełni zagnieżdżony.* Zapotrzebowania na przerwanie są porządkowane pod względem priorytetu od 0 (IR0) do 7 (IR7).
- a *Rotacyjny.* W wielu zastosowaniach pewna liczba urządzeń przerywających ma jednakowy priorytet. W tym trybie urządzenie, którego obsługę zakończono, otrzymuje najniższy priorytet w grupie.
- Q *Ze specjalną maską.* Pozwala on procesorowi na selektywne wzbranianie przerw od niektórych urządzeń.

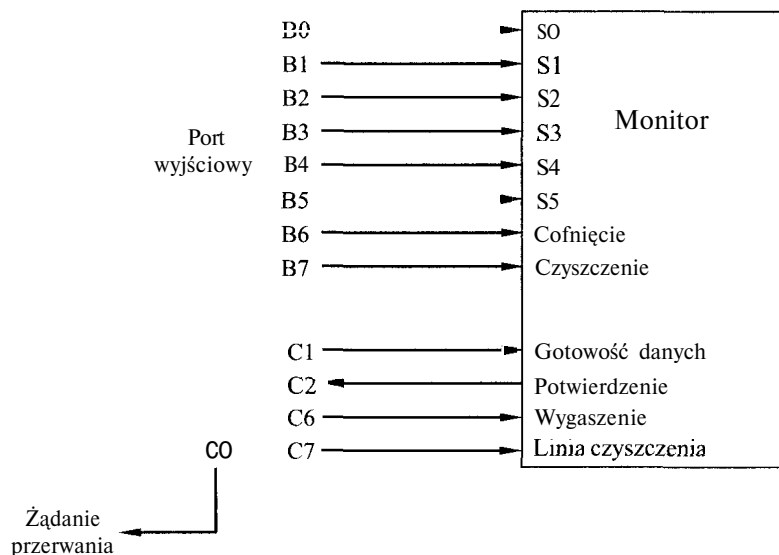
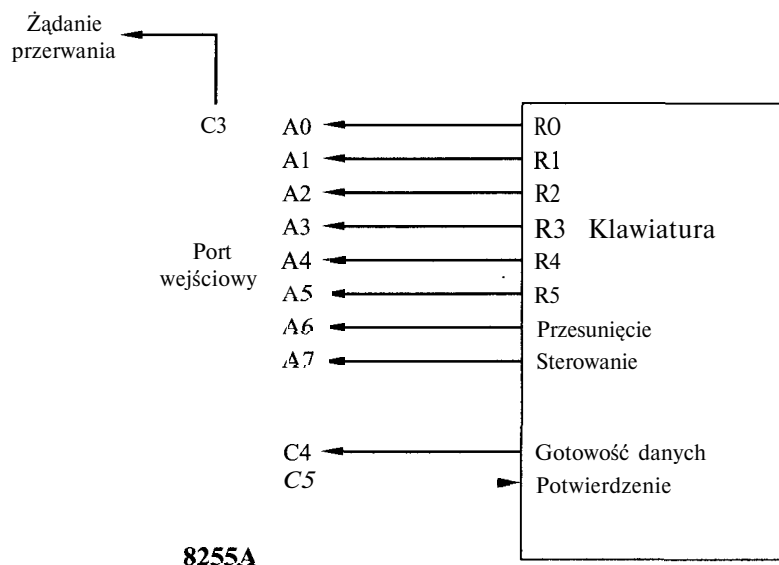
Programowalny interfejs urządzeń peryferyjnych Intel 8255A

Jako przykład modułu wejścia-wyjścia wykorzystywanego przez programowane wejście-wyjście oraz wejście-wyjście sterowane przerwaniami rozpatrzmy programowalny interfejs urządzeń peryferyjnych Intel 8255A. Układ ten jest jednostrukturalnym modułem wejścia-wyjścia o ogólnym przeznaczeniu, zaprojektowanym do wykorzystywania z procesorem Intel 8086. Na rysunku 6.9 widać jego ogólny schemat blokowy oraz przyporządkowanie wyprowadzeń obudowy o 40 końcówkach, w której jest zamykany.



RYSUNEK 6.9. Programowalny interfejs Intel 8255A: (a) schemat blokowy; (b) rozkład wyprowadzeń

Prawa strona schematu blokowego obejmuje interfejs zewnętrzny 8255A. Dwadzieścia cztery linie wejścia-wyjścia są programowalne przez procesor 8086 za pomocą rejestru sterowania. Procesor 8086 może ustalić wartość rejestru sterowania w celu sprecyzowania różnych trybów działania i konfiguracji. Linie wejścia-wyjścia są podzielone na trzy grupy 8-bitowe (A, B i C). Każda grupa może funkcjonować jako 8-bitowy port wejścia-wyjścia. Ponadto grupa C jest



RYSUNEK 6.10. Połączenie klawiatury i monitora z układem 8255A

podzielona na grupy 4-bitowe (C_A i C_B), które mogą być używane w połączeniu z portami A i B. Przy takiej konfiguracji mogą one przenosić sygnały sterowania i stanu.

Lewą stronę schematu blokowego stanowi wewnętrzny interfejs do magistrali procesora 8086. Obejmuje on 8-bitową dwukierunkową magistralę danych (D_0 – D_7) używaną do przesyłania danych do (i z) portów wejścia-wyjścia oraz do transferu informacji sterowania do rejestru sterowania. Transfer ma miejsce, gdy zezwala na to stan linii wyboru mikroukładu (CHIP SELECT) oraz stan jednej z linii odczytu (READ) lub zapisu (WRITE). Linia RESET jest wykorzystywana do ustawiania modułu w stanie początkowym.

Rejestr sterowania jest ładowany przez procesor w celu sterowania trybem działania oraz definiowania sygnałów, jeśli jest taka potrzeba. W trybie działania O trzy grupy po 8 linii zewnętrznych funkcjonują jako trzy 8-bitowe porty wejścia-wyjścia. Każdy z portów może być wyznaczony jako wejściowy lub wyjściowy. W innym przypadku grupy A i B funkcjonują jako porty wejścia-wyjścia, natomiast linie z grupy C służą jako linie sterujące dla A i B. Sygnały sterowania służą dwóm zasadniczym celom: uzgadnianiu i żądaniu przerwania. Uzgadnianie wykorzystuje prosty mechanizm taktowania. Jedna linia sterowania jest używana przez nadawcę jako linia gotowości danych (DATA READY), wskazująca, kiedy dane są obecne na liniach danych wejścia-wyjścia. Inna linia jest używana przez odbiorcę do potwierdzania (ACKNOWLEDGE), wskazując, że dane zostały odczytane i linie danych mogą być wyczyszczone. Jeszcze inna linia może być oznaczona jako linia zapotrzebowania przerwania (INTERRUPT REQUEST) i połączona z magistralą systemową.

Ponieważ układ 8255A jest programowalny za pośrednictwem rejestru sterowania, może on być używany do sterowania różnorodnymi prostymi urządzeniami peryferyjnymi. Rysunek 6.10 ilustruje jego zastosowanie do sterowania terminalem klawiatura/monitor. Klawiatura wykorzystuje wejście 8-bitowe. Dwa z tych bitów, PRZESUNIĘCIE (SHIFT) oraz STEROWANIE (CONTROL) mają znaczenie specjalne dla programu obsługi klawiatury realizowanego przez procesor. Jednakże ta interpretacja jest przezroczysta dla 8255A, który po prostu przyjmuje 8 bitów danych i wprowadza je na systemową magistralę danych. Do współpracy z klawiaturą przewidziane są dwie linie uzgadniania.

Monitor jest również połączony przez 8-bitowy port danych. I znów dwa z tych bitów mają specjalne znaczenie, które jednak jest przezroczyste dla 8255A. Poza dwiema liniami uzgadniania występują jeszcze dwie linie realizujące dodatkowe funkcje sterowania.

6.5

BEZPOŚREDNI DOSTĘP DO PAMIĘCI**Wady programowanego i sterowanego przerwaniemi wejścia-wyjścia**

Wejście-wyjście sterowane przerwaniemi, chociaż jest bardziej efektywne niż proste programowane wejście-wyjście, nadal wymaga aktywnej interwencji procesora przy przesyłaniu danych między pamięcią a modułem wejścia-wyjścia, a każdy transfer danych musi wędrować drogą wiodącą przez procesor. A więc obie te formy wejścia-wyjścia mają dwie nieodłączne wady:

1. Szybkość transferu wejścia-wyjścia jest ograniczana szybkością, z jaką procesor może testować i obsługiwać urządzenie.
2. Procesor jest zajęty zarządzaniem przesyłaniem z wejścia i na wyjście; w przypadku każdego takiego transferu musi być wykonana pewna liczba rozkazów (patrz np. rys. 6.4).

Między tymi niedogodnościami występuje pewna wymiennność. Rozważmy transfer bloku danych. W przypadku prostego programowanego wejścia-wyjścia, procesor jest przypisany określonej zadaniu i może przenosić dane ze stosunkowo dużą szybkością, kosztem rezygnacji z innych zadań. Wejście-wyjście sterowane przerwaniemi uwalnia do pewnego stopnia procesor, kosztem szybkości przesyłania. Niezależnie od tego obie metody mają ujemny wpływ zarówno na pracę procesora, jak i na szybkość transferu z wejścia i na wyjście.

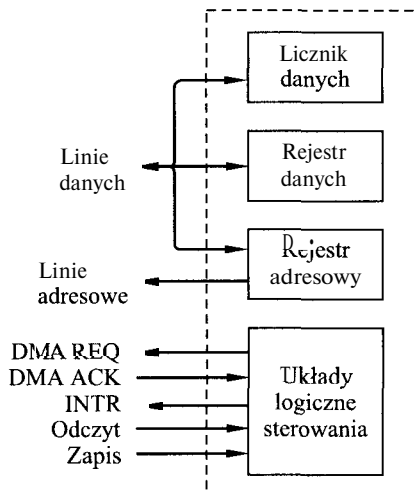
Gdy muszą być przenoszone duże ilości danych, wymagana jest bardziej efektywna metoda: bezpośredni dostęp do pamięci (DMA).

Działanie DMA

Bezpośredni dostęp do pamięci wymaga dodatkowego modułu na magistrali systemowej. Moduł DMA (rys. 6.11) może „udawać” procesor i w rzeczywistości przejmować od procesora sterowanie systemem. Metoda ta działa następująco. Gdy procesor życzy sobie odczytania lub zapisu bloku danych, wydaje rozkaz modułowi DMA, wysyłając mu następujące informacje:

- a czy wymagany jest odczyt, czy zapis;
- a adres niezbędnego urządzenia wejścia-wyjścia;
- a adres początkowej komórki pamięci przewidzianej do odczytania lub do zapisania;
- a liczbę słów, które mają być odczytane lub zapisane.

Procesor następnie kontynuuje inne prace. Zleca tę operację wejścia-wyjścia modułowi DMA i moduł ten zatroszczy się o nią. Moduł DMA przenosi cały



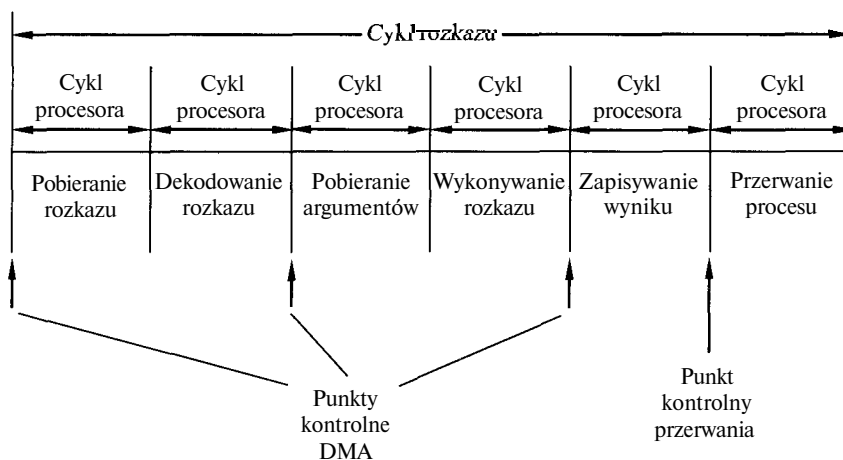
RYСУNEK 6.11. Schemat blokowy typowego modułu DMA

blok danych, słowo po słowie, bezpośrednio z (lub do) pamięci, bez przechodzenia przez procesor. Gdy transfer jest zakończony, moduł DMA wysyła sygnał przerwania do procesora. Dzięki temu procesor jest angażowany tylko na początku i na końcu transferu (rys. 6.4c).

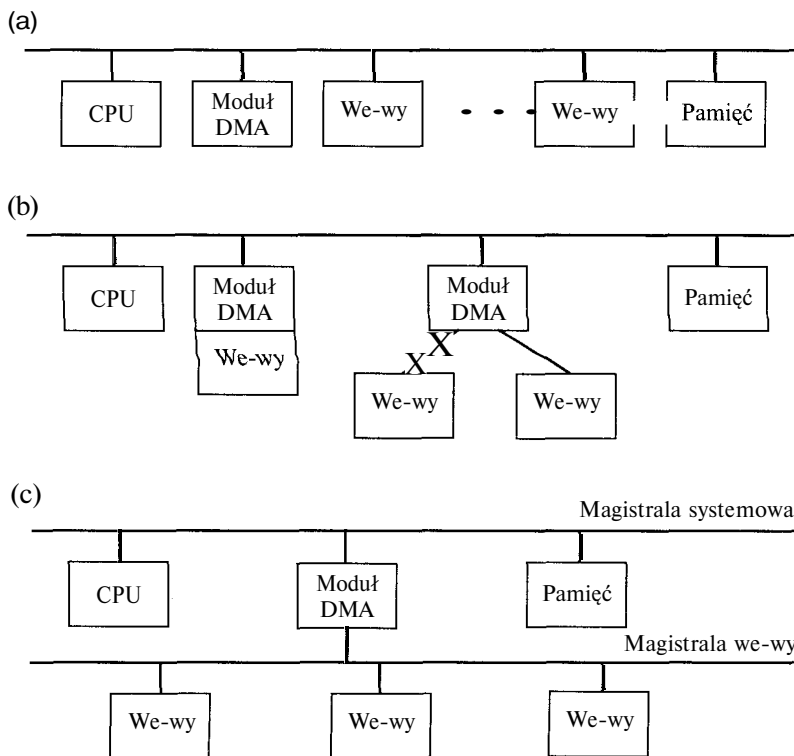
Moduł DMA wymaga przejścia sterowania magistralą w celu przenoszenia danych z (i do) pamięci. W tym celu moduł DMA może używać magistrali tylko wtedy, gdy nie potrzebuje jej procesor, lub może wymusić czasowe zawieszenie operacji procesora. Ta ostatnia metoda jest bardziej powszechna i jest określana jako wykradanie cyklu, ponieważ w wyniku tego moduł DMA zajmuje cykl magistrali.

Na rysunku 6.12 są pokazane miejsca, w których może być zawieszane wykonanie cyklu rozkazu procesora. W każdym przypadku procesor jest zawieszany tuż przed zgłoszeniem zapotrzebowania na magistralę. Moduł DMA przesyła następnie jedno słowo i zwraca sterowanie do procesora. Zauważmy, że nie jest to przerwanie; procesor nie zachowuje kontekstu i nie wykonuje niczego innego. Procesor zatrzymuje się na jeden cykl magistrali. W ogólnym rozrachunku procesor pracuje nieco wolniej. Mimo tego, w przypadku transferu z wejścia lub na wyjście obejmującego wiele słów, metoda DMA jest daleko bardziej efektywna niż wejście-wyjście sterowane przerwaniem lub programowane.

Mechanizm DMA może być konfigurowany na różne sposoby. Niektóre możliwości są pokazane na rys. 6.13. W pierwszym przykładzie wszystkie moduły używają tej samej magistrali systemowej. Moduł DMA, działając jako namiastka procesora, używa programowanego wejścia-wyjścia do wymiany danych między pamięcią a modułem wejścia-wyjścia za pośrednictwem modułu DMA. Konfiguracja ta, chociaż może być niedroga, jest wyraźnie nieefektywna. Podobnie jak w przypadku programowanego wejścia-wyjścia sterowanego przez procesor, każdy transfer słowa pochłania dwa cykle magistrali.



RYSUNEK 6.12. Punkty kontrolne DMA i przerwania podczas cyklu rozkazu



RYSUNEK 6.13. Możliwe konfiguracje modułu DMA: (a) pojedyncza magistrala, odłączalne DMA; (b) pojedyncza magistrala, zintegrowane DMA-wejście-wyjście; (c) magistrala wejścia-wyjścia

Liczba wymaganych cykli magistrali może być istotnie ograniczona przez zintegrowanie funkcji DMA i wejścia-wyjścia. Jak widać na rys. 6.13b, będzie wówczas istniała ścieżka między modułem DMA a jednym lub wieloma modułami wejścia-wyjścia nie angażująca magistrali systemowej. W istocie układ logiczny DMA może być częścią modułu wejścia-wyjścia lub może być oddzielnym modułem, który steruje jednym lub wieloma modułami wejścia-wyjścia. Koncepcja ta może być rozwinięta przez połączenie modułów wejścia-wyjścia z modułem DMA za pomocą magistrali wejścia-wyjścia (rys. 6.13c). Redukuje to liczbę interfejsów wejścia-wyjścia w module DMA do jednego i stanowi łatwo rozszerzalną konfigurację. We wszystkich tych przypadkach (rys. 6.13b i c) magistrala systemowa używana wspólnie przez moduł DMA, procesor i pamięć jest wykorzystywana przez DMA tylko do wymiany danych z pamięcią. Wymiana danych między DMA a modułami wejścia-wyjścia ma miejsce poza magistralą systemową.

6.6

KANAŁY I PROCESORY WEJŚCIA-WYJŚCIA

Ewolucja funkcjonowania wejścia-wyjścia

W miarę ewolucji systemów komputerowych mieliśmy do czynienia z rosnącą złożonością i wyrafinowaniem poszczególnych zespołów. Nigdzie nie jest to bardziej widoczne niż w przypadku funkcjonowania wejścia-wyjścia. Omówiliśmy już część tej ewolucji. Jej etapy można podsumować następująco:

1. Procesor bezpośrednio steruje urządzeniem peryferyjnym. Można to zaobserwować w prostych urządzeniach sterowanych mikroprocesorami.
2. Dodany jest sterownik lub moduł wejścia-wyjścia. Procesor używa programowanego wejścia-wyjścia bez przerw. Na tym etapie procesor oddala się nieco od specyficznych szczegółów interfejsu urządzenia zewnętrznego.
3. Konfiguracja jest taka sama, jak na etapie 2, jednak teraz używane są przerwy. Procesor nie musi tracić czasu, czekając na zakończenie operacji wejścia-wyjścia, dzięki czemu zwiększa się jego wydajność.
4. Moduł wejścia-wyjścia uzyskuje bezpośredni dostęp do pamięci poprzez DMA. Może teraz przenosić bloki danych z (i do) pamięci bez angażowania jednostki centralnej, z wyjątkiem początku i końca transferu.
5. Moduł wejścia-wyjścia zostaje wzbogacony i sam staje się procesorem o wyspecjalizowanej liście rozkazów dostosowanej do zadań wejścia-wyjścia. Jednostka centralna skłania procesor wejścia-wyjścia do wykonania programu zawartego w pamięci. Procesor wejścia-wyjścia pobiera i wykonuje rozkazy bez interwencji jednostki centralnej. Pozwala to procesorowi na ustalanie sekwencji działań wejścia-wyjścia, a przerwanie może nastąpić tylko po zrealizowaniu całej sekwencji.

6. Moduł wejścia-wyjścia uzyskuje własną pamięć i w rzeczywistości sam staje się komputerem. W przypadku tej architektury dużym zestawem urządzeń wejścia-wyjścia można sterować przy minimalnym zaangażowaniu jednostki centralnej. Powszechnie wykorzystuje się taką architekturę do sterowania komunikacją z terminalami konwersacyjnymi. Procesor wejścia-wyjścia przejmuje większość zadań związanych ze sterowaniem terminalami.

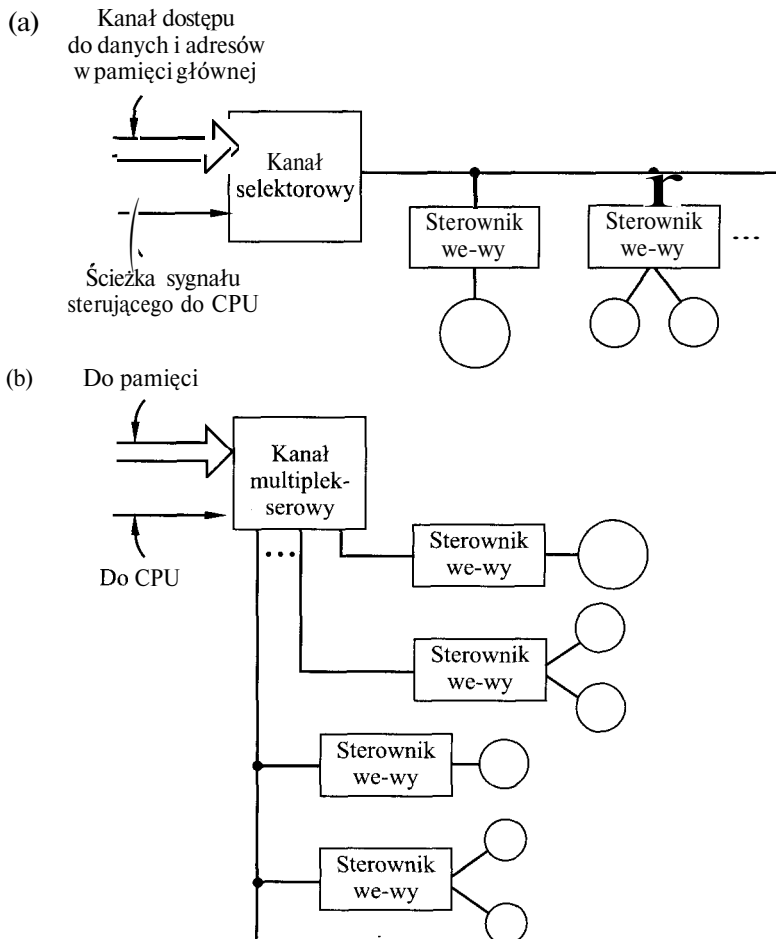
W miarę posuwania się po tej drodze ewolucji coraz więcej funkcji wejścia-wyjścia realizuje się bez angażowania procesora, który jest stopniowo uwalniany od zadań związanych z wejściem-wyjściem, co zwiększa jego wydajność. Na dwóch ostatnich etapach (5 i 6) następuje poważna zmiana, polegająca na wprowadzeniu koncepcji modułu wejścia-wyjścia zdolnego do wykonywania programu. Na etapie 5 moduł wejścia-wyjścia jest często określany jako **kanał wejścia-wyjścia**. Na etapie 6 jest stosowany termin **procesor wejścia-wyjścia**. Czasem oba terminy bywają stosowane w obu sytuacjach. W dalszym ciągu będziemy posługiwali się terminem kanał wejścia-wyjścia.

Własności kanałów wejścia-wyjścia

Kanał wejścia-wyjścia stanowi rozszerzenie koncepcji DMA. Może on wykonywać rozkazy wejścia-wyjścia, a co za tym idzie - w pełni sterować operacjami wejścia-wyjścia. W systemie komputerowym wyposażonym w takie urządzenie jednostka centralna nie wykonuje rozkazów wejścia-wyjścia. Rozkazy takie są przechowywane w pamięci głównej i są przeznaczone do realizacji przez specjalizowany procesor w samym kanale wejścia-wyjścia. Jednostka centralna inicjuje więc transfer z wejścia i na wyjście, przekazując kanałowi wejścia-wyjścia rozkaz wykonania programu zawartego w pamięci. Za pomocą programu są określone potrzebne urządzenia lub urządzenie, obszar lub obszary pamięci przewidziane do użycia, priorytety oraz działania, które mają być podejmowane w określonych warunkach wystąpienia błędu. Kanał wejścia-wyjścia wykonuje te rozkazy i steruje przesyłaniem danych.

Powszechnie spotykane są dwa typy kanałów wejścia-wyjścia, zilustrowane na rys. 6.14.

Kanał wybiórczy (ang. *selector channel*) steruje wieloma szybkimi urządzeniami i w określonym czasie zajmuje się transferem danych z jednego spośród tych urządzeń. Kanał wejścia-wyjścia wybiera więc urządzenie i realizuje transfer danych. Każde urządzenie lub niewielki zespół urządzeń jest sterowany za pomocą sterownika lub modułu wejścia-wyjścia, podobnego do modułów wejścia-wyjścia omawianych powyżej. Tak więc kanał wejścia-wyjścia zastępuje procesor w czynności kontrolowania tych sterowników wejścia-wyjścia. **Kanał multiplekserowy** (ang. *multiplexor channel*) może jednocześnie współpracować z wieloma urządzeniami wejścia-wyjścia. W przypadku urządzeń powolnych **multiplekser bajtowy** odbiera lub przekazuje znaki do wielu urządzeń



RYSUNEK 6.14. Architektura kanału wejścia-wyjścia: (a) selektor; (b) multiplexer

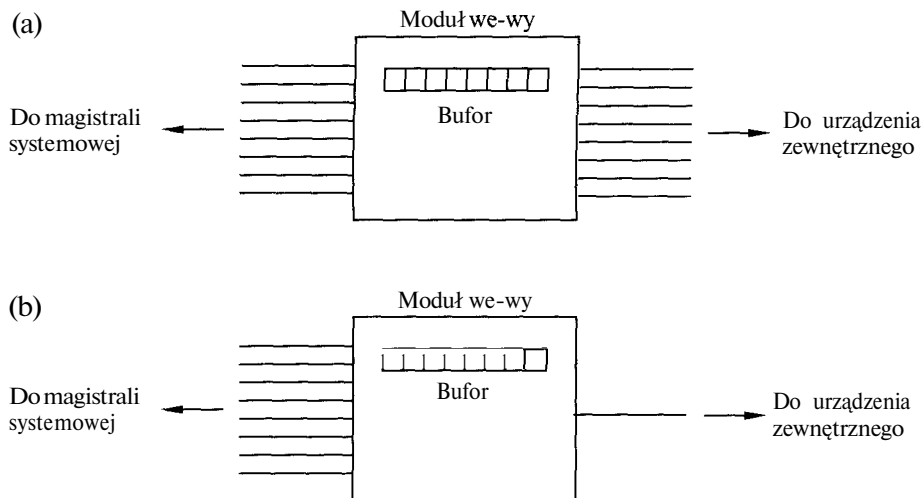
tak szybko, jak tylko jest to możliwe. Na przykład, wynikowy strumień znaków pochodzących od trzech urządzeń o różnych szybkościach, wysyłających indywidualne strumienie $A_1 A_2 A_3 A_4 \dots$, $B_1 B_2 B_3 B_4 \dots$ i $Q C_2 C_3 C_4 \dots$, może wyglądać następująco: $A_1 B_1 C_1 A_2 C_2 A_3 B_3 C_3 A_4$ itd. W przypadku urządzeń szybkich multiplexer blokowy przeplata bloki danych z kilku urządzeń.

6.7

INTERFEJS ZEWNĘTRZNY

Rodzaje interfejsów

Interfejs między urządzeniem peryferyjnym a modułem wejścia-wyjścia musi być dostosowany do natury i działania urządzenia peryferyjnego. Jedną z głównych cech tego interfejsu jest to, czy jest on szeregowy, czy równoległy



RYСУNEK 6.15. Równoległe (a) i szeregowe (b) wejście-wyjście

(rys. 6.15). W *interfejsie równoległym* występuje wiele linii łączących moduł wejścia-wyjścia z urządzeniem peryferyjnym i wiele bitów jest przesyłanych jednocześnie, podobnie jak wszystkie bity słowa są przenoszone równocześnie przez szynę danych. W *interfejsie szeregowym* do transmisji danych służy tylko jedna linia i w określonym momencie jest transmitowany tylko jeden bit. Interfejs równoległy jest powszechnie wykorzystywany do współpracy z szybkimi urządzeniami peryferyjnymi, takimi jak pamięć taśmowa lub dyskowa. Interfejs szeregowy jest bardziej powszechny w przypadku drukarek i terminali.

W każdym przypadku moduł wejścia-wyjścia musi się angażować w dialog z urządzeniem peryferyjnym. Ogólnie rzecz biorąc, dialog w przypadku operacji zapisu wygląda następująco:

1. Moduł wejścia-wyjścia wysyła sygnał sterowania domagający się zgody na wysłanie danych.
2. Urządzenie peryferyjne potwierdza żądanie.
3. Moduł wejścia-wyjścia przesyła dane (słowo lub blok, zależnie od urządzenia peryferyjnego).
4. Urządzenie peryferyjne potwierdza otrzymanie danych.

Operacja odczytu przebiega podobnie.

Kluczem do działania modułu wejścia-wyjścia jest wewnętrzny bufor, w którym mogą być przechowywane dane przechodzące między urządzeniem peryferyjnym a resztą systemu. Bufor ten umożliwia modułowi wejścia-wyjścia kompensowanie różnic szybkości między magistralą systemową a swoimi liniami zewnętrznymi.

Konfiguracje dwupunktowe i wielopunktowe

Połączenie między modułem wejścia-wyjścia w systemie komputerowym a urządzeniami zewnętrznymi może być dwupunktowe lub wielopunktowe. W przypadku interfejsu dwupunktowego między modułem wejścia-wyjścia a urządzeniem zewnętrznym jest przewidziana specjalistyczna linia. W małych systemach (komputery osobiste, stacje robocze) typowe łączy dwupunktowe dotyczą klawiatury, drukarki i modemu zewnętrznego. Typowym przykładem takiego interfejsu jest specyfikacja EIA-232 (jej opis zawarto w [STAL94]).

Wzrasta znaczenie wielopunktowych interfejsów zewnętrznych wykorzystywanych do obsługi zewnętrznych urządzeń pamięci masowej (napędów dyskowych lub taśmowych) oraz urządzeń multimedialnych (CD-ROM, wideo, audio). Interfejsy wielopunktowe są w istocie zewnętrznymi magistralami i mają takie same rodzaje rozwiązań logicznych, co magistrale przedyskutowane w rozdz. 3. W tym podrozdziale omówimy dwa kluczowe przykłady: SCSI oraz P1394.

Interfejs małych systemów komputerowych (SCSI)

Dobrym przykładem interfejsu urządzeń peryferyjnych jest interfejs SCSI. Został on najpierw spopularyzowany w komputerach Macintosh w roku 1984. Obecnie jest szeroko stosowany w systemach komputerów osobistych kompatybilnych zarówno z produktami Macintosh, jak i IBM, a także w wielu stacjach roboczych. SCSI jest standardowym interfejsem napędów CD-ROM, urządzeń audio i zewnętrznych pamięci masowych. Wykorzystuje on interfejs równoległy z 8, 16 lub 32 liniami.

Konfiguracja SCSI jest ogólnie określana jako magistrala, chociaż w rzeczywistości urządzenia są połączone łańcuchowo. Każde urządzenie SCSI ma dwa łączy: jedno na wejściu i jedno na wyjściu. Wszystkie urządzenia są połączone łańcuchowo, przy czym jeden z końców łańcucha jest połączony z komputerem macierzystym. Wszystkie urządzenia funkcjonują niezależnie i mogą wymieniać dane zarówno między sobą, jak i z systemem macierzystym. Na przykład, pamięć na dyskach stałych może współpracować z pamięcią taśmową, nie angażując systemu macierzystego. Dane są przenoszone w postaci pakietów komunikatów, co opisujemy poniżej.

Wersje SCSI

Oryginalna specyfikacja SCSI, nazywana obecnie SCSI-1, powstała we wczesnych latach osiemdziesiątych. SCSI-1 odnosi się do 8 linii danych i pracuje z szybkością zegara 5 MHz, przenosząc dane z szybkością 5 MB/s. Umożliwia on łańcuchową konfigurację do 7 urządzeń oraz połączenie ich z systemem macierzystym.

W roku 1991 wprowadzono zrewidowaną specyfikację pod nazwą SCSI-2. Najbardziej godnymi uwagi zmianami były: opcjonalne rozszerzenie linii danych do 16 lub 32 oraz zwiększenie szybkości zegara do 10 MHz. W rezultacie uzyskano maksymalną szybkość transferu danych równą 20 lub 40 MB/s.

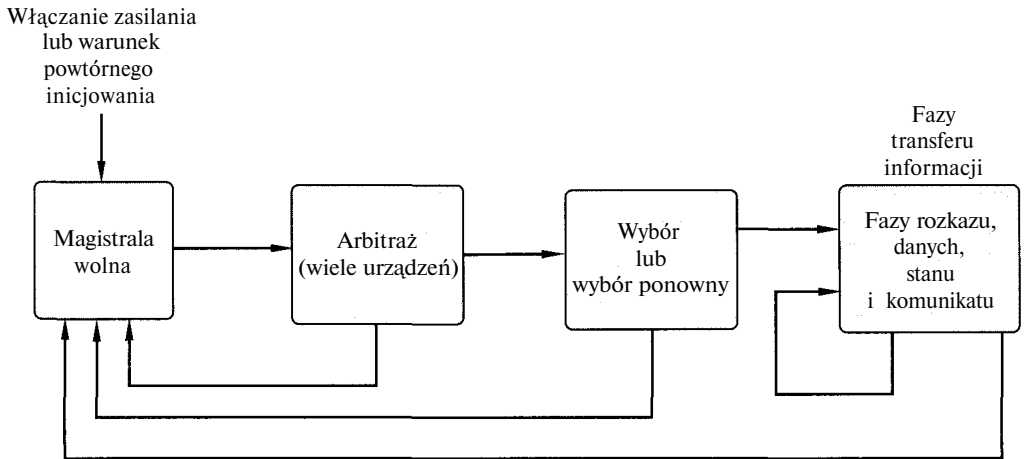
Prowadzone są obecnie prace nad specyfikacją SCSI-3, która umożliwi pracę z jeszcze większą szybkością.

Sygnały i fazy

Wszystkie wymiany na magistrali SCSI odbywają się między inicjatorem a adresatem. Zwykle inicjatorem jest system macierzysty, a adresatem sterownik urządzenia peryferyjnego. Jednak niektóre urządzenia mogą pełnić obie funkcje. W każdym przypadku wszystkie działania na magistrali następują w postaci sekwencji faz. Fazy te są następujące:

- a **Magistrala wolna.** Wskazuje, że żadne urządzenie nie korzysta z magistrali i że jest ona dostępna do użytku.
- **Arbitraż.** Umożliwienie jednemu urządzeniu przejęcia sterowania magistralą, dzięki czemu może ono zainicjować lub zakończyć proces wejścia-wyjścia.
- Q **Wybór.** Umożliwienie inicjatorowi wybrania adresata do wykonania rozkazu odczytu lub zapisu.
- **Powtórny wybór.** Umożliwienie adresatowi ponownego połączenia się z inicjatorem w celu zakończenia operacji rozpoczętej uprzednio przez inicjatora, lecz zawieszzonej przez adresata.
- **Rozkaz.** Umożliwienie adresatowi zgłoszenia inicjatorowi rozkazu żądania.
- Q **Dane.** Umożliwienie adresatowi zgłoszenia żądania transferu danych albo od adresata do inicjatora (wejście danych), albo od inicjatora do adresata (wyjście danych).
- Q **Stan.** Umożliwienie adresatowi zgłoszenia żądania przesłania informacji o stanie od adresata do inicjatora.
- Q **Komunikat.** Umożliwienie adresatowi zgłoszenia żądania transferu jednego lub wielu komunikatów albo od adresata do inicjatora (wejście komunikatu), albo od inicjatora do adresata (wyjście komunikatu).

Na rysunku 6.16 jest pokazany porządek następowania faz na magistrali SCSI. Po sprowadzeniu do stanu początkowego lub włączeniu zasilania magistrala rozpoczyna fazę „magistrala wolna”. Po niej następuje faza „arbitraż” lub „powtórny wybór”, w której zachodzi ustalenie urządzeń będących inicjatorem i adresatem w tej wymianie. Po wyznaczeniu obu urządzeń następuje jedna lub wiele faz transferu informacji („rozkaz”, „dane”, „stan”, „komunikat”) angażujących te urządzenia. Ostatnią fazą transferu informacji jest zwykle faza „wejście komunikatu”, w której do inicjatora jest przesyłany komunikat „rozłącz” lub „rozkaz zakończony”, po czym następuje faza „magistrala wolna”.



RYSUNEK 6.16. Fazy magistrali SCSI

Ważną cechą SCSI jest zdolność powtórnego wyboru. Jeśli wykonanie wydanego rozkazu wymaga pewnego czasu, to adresat może uwolnić magistralę i później ponownie połączyć się z inicjatorem. Na przykład, system macierzysty może skierować do napędu dysków rozkaz sformatowania dysku, napęd zaś może przeprowadzić tę operację bez zajmowania magistrali.

Specyfikacja SCSI-1 określa, że kabel składa się z 18 linii sygnałowych, 9 linii sterowania i 9 linii danych (8 linii danych i 1 parzystości). Linie sterowania są następujące:

- a BSY. Ustawiana przez inicjatora lub adresata w celu pokazania, że magistrala jest zajęta.
- a SEL. Używana przez inicjatora w celu wybrania adresata do wykonania rozkazu lub przez adresata do ponownego wyboru inicjatora, od którego jest on odłączony. Sygnał wejścia-wyjścia umożliwia rozróżnienie wyboru i ponownego wyboru.
- C/D. Używana przez inicjatora w celu wskazania, czy sygnały na magistrali reprezentują sterowanie (fazy: „rozkaz”, „stan” lub „komunikat”), czy też są właściwymi danymi.
- Q I/O. Używana przez adresata do sterowania kierunkiem ruchu danych na szynie danych.
- MSG. Używana przez adresata w celu pokazania inicjatorowi, czy przesyłana informacja jest komunikatem.
- a REQ. Używana przez adresata do zgłaszania żądania transferu danych. W odpowiedzi na sygnał REQ inicjator przyjmuje dane z magistrali w fazie „wejście danych” lub umieszcza dane na magistrali w fazie „wyjście danych”.
- a ACK. Używana przez inicjatora do potwierdzania sygnału REQ od adresata. Sygnał ACK wskazuje, że inicjator umieścił informację na szynie danych

w fazie „wyjście danych” lub przyjął dane z szyny w fazie „wejście danych”.

- a ATN. Używana przez inicjatora do informowania adresata, że dostępny jest komunikat przeznaczony do przesłania. Inicjator może potwierdzać ten sygnał w fazie „wybór” lub w dowolnym momencie po przejściu sterowania magistralą przez inicjatora.
- a RST. Używany do przywracania stanu początkowego magistrali.

W tabeli 6.4 są podane zależności między sygnałami a fazami magistrali. W przypadku SCSI-2 występują dodatkowe linie danych i parzystości oraz dodatkowa para linii REQ/ACK.

TABELA 6.4. Sygnały magistrali SCSI

Faza magistrali	Sygnały kabla A					Sygnały kabla B		
	BSY	SEL	C/D, I/O MSG, REQ	ACK, ATN	DB(7÷0) DB(P)	REOP	ACKB	DB(31÷8), DB(P1), DB(P2), DB(P3)
Magistrala wolna	brak	brak	brak	brak	brak	brak	brak	brak
Arbitraż	wszystkie	wygr.	brak	brak	S ID	brak	brak	brak
Wybór	I&C	inicjator	brak	inicjator	inicjator	brak	brak	brak
Ponowny wybór	I&C	cel	cel	inicjator	cel	brak	brak	brak
Rozkaz	cel	brak	cel	inicjator	inicjator	brak	brak	brak
Wejście danych	cel	brak	cel	inicjator	cel	cel	inicjator	cel
Wyjście danych	cel	brak	cel	inicjator	inicjator	cel	inicjator	inicjator
Stan	cel	brak	cel	inicjator	cel	brak	brak	brak
Wejście komunikatu	cel	brak	cel	inicjator	cel	brak	brak	brak
Wyjście komunikatu	cel	brak	cel	inicjator	inicjator	brak	brak	brak

- wszystkie - sygnał jest aktywowany przez wszystkie przyrządy SCSI biorące czynny udział w arbitrażu,
- S ID - unikatowy bit danych (identyfikator SCSI), aktywowany przez każdy przyrząd SCSI biorący czynny udział w arbitrażu,
- I&C - sygnał jest aktywowany przez przyrząd inicjujący, docelowy lub przez jeden i drugi, zgodnie ze specyfikacją dotyczącą fazy wyboru i ponownego wyboru,
- inicjator - jeśli jest aktywowany, to tylko przez przyrząd inicjujący,
- brak - sygnał nie jest aktywowany,
- wygr. - sygnał jest aktywowany przez przyrząd wygrywający arbitraż,
- cel - jeśli jest aktywowany, to tylko przez czynny przyrząd docelowy.

Przebiegi czasowe sygnałów na magistrali SCSI

Na rysunku 6.17 widać typowy przykład przebiegu czasowego sygnałów na magistrali SCSI, którego można użyć do wyjaśnienia różnych faz i sygnałów magistrali. Przykład ten dotyczy rozkazu odczytu, który powoduje przeniesienie danych od adresata do inicjatora. Gdy wszystkie linie są wolne, sekwencja rozpo-

czyna się od fazy „magistrala wolna”. Następnie ma miejsce faza „arbitraż”, w której jedno lub wiele urządzeń rywalizuje o sterowanie magistralą. Każde z urządzeń potwierdza linię BSY i jedną z linii danych. Każde z ośmiu urządzeń (jedno macierzyste oraz do 7 innych) ma unikatowy identyfikator ID od O do 7 i każde potwierdza jedną z linii danych odpowiadających swojemu ID. Identyfikatorom przypisano priorytet, przy czym 7 ma najwyższy priorytet, a O - najniższy. Jeśli więcej niż jedno urządzenie potwierdza swój ID w fazie arbitrażu, to wygrywa urządzenie o najwyższym priorytecie. Pozostałe urządzenia rozpoznają to, obserwując linie danych i poddają się arbitrażowi.

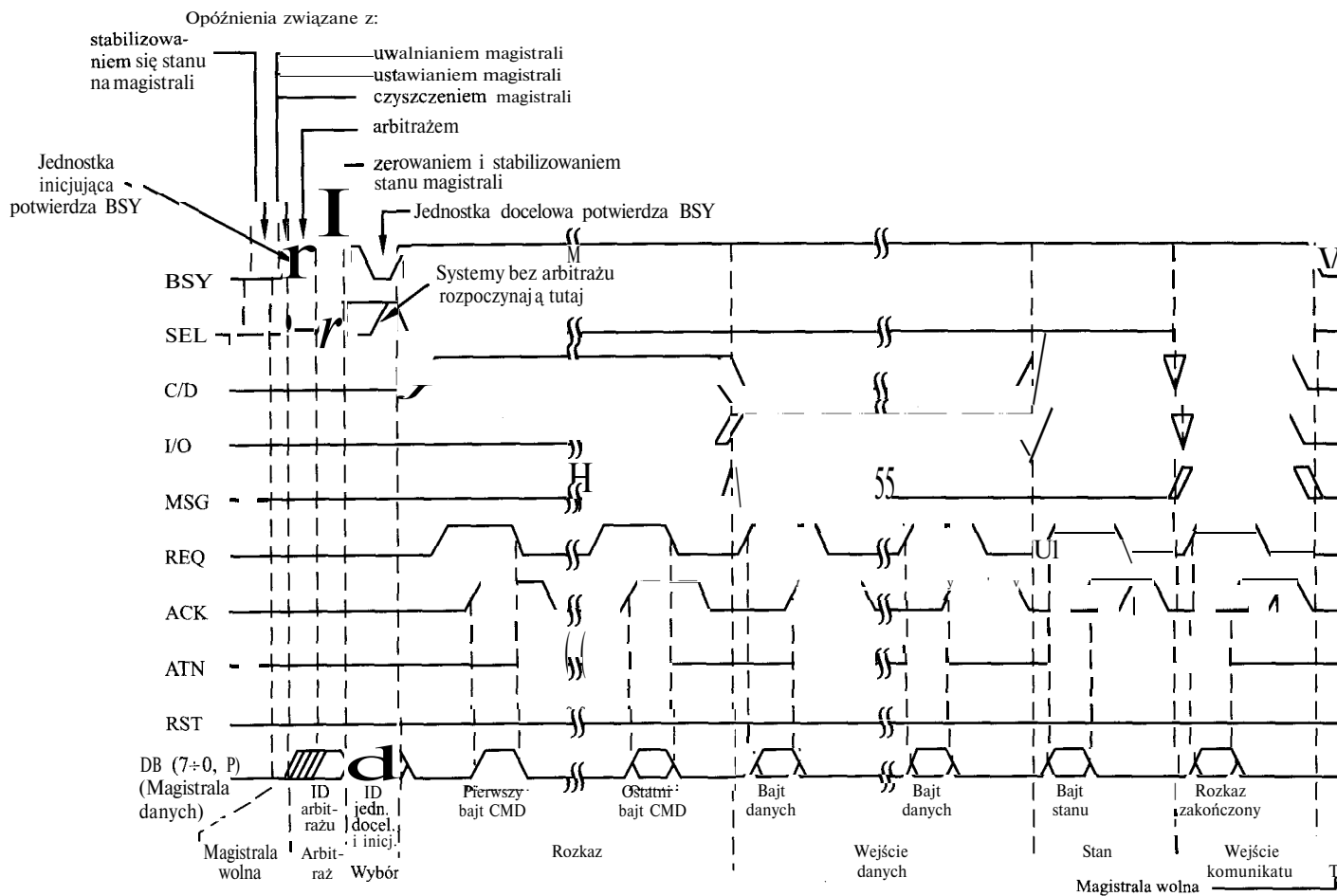
Urządzenie, które wygrało, staje się inicjatorem. Otwiera ono fazę „wybór” przez potwierdzenie sygnału SEL. Podczas tej fazy inicjator potwierdza swój ID oraz ID adresata na dwóch odpowiednich liniach danych. Po pewnym opóźnieniu neguje on sygnał BSY. Gdy wybrany adresat wykrywa potwierdzony sygnał SEL, a BSY i I/O - zanegowane, oraz rozpoznaje swój ID, potwierdza sygnał BSY. Gdy inicjator wykrywa BSY, uwalnia magistralę danych i neguje SEL.

Następnie adresat pokazuje rozpoczęcie fazy „rozkaz” przez potwierdzenie linii C/D; linia ta pozostaje potwierdzona podczas tej fazy. Następnie zgłasza zapotrzebowanie na pierwszy bajt rozkazu inicjatora, potwierdzając REQ. Inicjator umieszcza pierwszy bajt rozkazu na magistrali danych i potwierdza ACK. Adresat, po odczytaniu bajta, neguje REQ. Następnie inicjator neguje ACK. Pierwszy bajt rozkazu zawiera kod operacyjny rozkazu wskazujący, ile bajtów pozostało do przekazania. Te dodatkowe bajty są przekazywane za pomocą takiej samej procedury uzgadniania REQ/ACK przed i po każdym przesłaniu bajta.

Po otrzymaniu i zinterpretowaniu rozkazu przez adresata, rozpoczyna on fazę „wejście danych” magistrali przez zanegowanie linii C/D (co oznacza, że magistrala danych zawiera dane) i potwierdza linię I/O (co oznacza kierunek transferu od adresata do inicjatora). Adresat umieszcza pierwszy bajt żądanych danych na magistrali danych i potwierdza linię REQ. Po odczytaniu bajta inicjator potwierdza linię ACK. Pozostałe bajty danych są przekazywane za pomocą takiej samej procedury uzgadniania REQ/ACK przed i po każdym przesłaniu bajta.

Po przekazaniu wszystkich żądanych danych adresat rozpoczyna fazę „stan” magistrali i przekazuje inicjatorowi bajt stanu, wskazując na udane zakończenie transferu. W tym przypadku linia C/D jest znów potwierdzona, a I/O pozostaje potwierdzona. Inicjator i adresat posługują się uzgadnianiem REQ/ACK w celu skoordynowania transferu bajta stanu.

Na zakończenie adresat otwiera fazę „wejście komunikatu” magistrali przez potwierdzenie linii MSG i wysyła bajt komunikatu zawierający komunikat „rozkaz zakończony”. Kiedy inicjator odbierze ten bajt, adresat uwalnia wszystkie linie sygnałowe magistrali, powodując w ten sposób wejście magistrali w fazę „magistrala wolna”.



RYSUNEK 6.17. Przykładowy przebieg czasowy sygnałów na magistrali SCSI

Transakcja pokazana na rys. 6.17 jest przykładem transferu asynchronicznego. W przypadku transferu asynchronicznego dla każdego przekazywanego bajta jest wymagane uzgadnianie REQ/ACK. Standard SCSI umożliwia także transfer synchroniczny, który jest mniej czasochłonny. Tryb przesyłania synchronicznego jest stosowany wyłącznie w fazach „wejście danych” i „wyjście danych”. Ponieważ pozostała część transferu jest asynchroniczna, tryb synchroniczny musi być negocjowany. Adresat wysyła inicjatorowi komunikat „zapotrzebowanie na synchroniczny transfer danych”, który zawiera minimalny czas trwania przesyłania i jego maksymalne opóźnienie dozwolone między REQ a odpowiadającym mu ACK. Inicjator odpowiada za pomocą takiego samego komunikatu, pokazując swój minimalny czas przesyłania oraz maksymalne opóźnienie REQ/ACK.

Synchroniczny tryb transferu ustalony po takiej wymianie komunikatów charakteryzuje się dłuższym z dwóch zgłoszonych minimalnych czasów transferu oraz mniejszym spośród dwóch zgłoszonych maksymalnych opóźnień REQ/ACK. Przesyłanie danych przebiega potem następująco. Nadawca wysyła bajty danych oddzielone od siebie przynajmniej minimalnym czasem transferu, sygnalizuje każdy bajt za pomocą pulsującego sygnału REQ. Odbiorca nie musi potwierdzać ACK natychmiast po każdym odebranych bajcie, jednak musi to zrobić przed upłynięciem czasu opóźnienia. Nadawca może wysyłać ciągły strumień bajtów tak długo, jak długo otrzymuje odpowiednie sygnały ACK w ramach czasu opóźnienia. Jeśli sygnał ACK jest spóźniony, nadawca musi czekać na potwierdzenie.

Komunikaty

Wymiana komunikatów między inicjatorami a adresatami ma na celu zarządzanie interfejsem SCSI. Występują trzy formaty komunikatów: jednobajtowe, dwubajtowe i komunikaty rozszerzone do trzech i więcej bajtów. Oto przykłady komunikatów:

- a Rozkaz zakończony.** Wysyłany przez adresata do inicjatora w celu pokazania, że wykonywanie rozkazu zostało zakończone i że do inicjatora została wysłana ważna informacja o stanie.
- Rozłącz.** Wysyłany przez adresata w celu poinformowania inicjatora, że bieżące połączenie ma być przerwane, ale do zakończenia bieżącej operacji będzie potrzebne późniejsze ponowne połączenie.
- a Błąd wykryty przez inicjatora.** Wysyłany przez inicjatora w celu poinformowania adresata, że wystąpił błąd (np. parzystości), który nie wyklucza adresata z ponownej próby wykonania operacji.
- Przerwanie (zawieszenie awaryjne).** Wysyłany przez inicjatora do adresata w celu skasowania bieżącej operacji.
- Synchroniczny transfer danych.** Wymieniany między inicjatorem a adresatem w celu ustanowienia synchronicznego transferu danych.

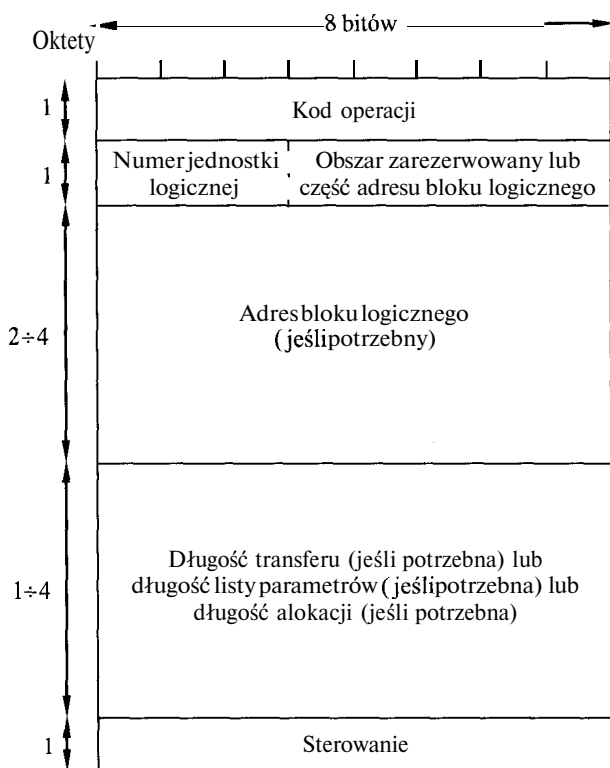
Rozkazy

Głównym składnikiem protokołu SCSI jest lista rozkazów. Rozkaz jest wydawany przez inicjatora w celu spowodowania wykonania przez adresata pewnych operacji. Rozkaz może obejmować pobranie danych od adresata (odczyt), wysłanie danych do adresata (zapis) lub inne działanie specyficzne dla danego urządzenia peryferyjnego. W każdym przypadku wykonanie rozkazu obejmuje niektóre (lub wszystkie) spośród niżej wymienionych kroków:

- ☐ Adresat przyjmuje i dekoduje rozkaz.
- ☐ Dane są przenoszone do lub od adresata (dotyczy nie wszystkich rozkazów).
- ☐ Adresat generuje i przekazuje informację o stanie.

Rozkaz jest określany w **bloku opisu rozkazu** (ang. *command descriptor block* - CDB) przygotowanym przez inicjatora. Po ustanowieniu połączenia między inicjatorem i adresatem inicjator wysyła adresatowi CDB za pomocą magistrali danych, aby zapoczątkować wykonywanie rozkazu.

Na rysunku 6.18 jest pokazany ogólny format bloku CDB. Oto pola występujące w bloku CDB:



RYСУNEK 6.18. Format bloku deskryptora rozkazu SCSI

- a **Kod operacji.** Kod ten określa konkretny rozkaz i format pozostałej części bloku CDB.
- a **Numer jednostki logicznej.** Identyfikuje fizyczne lub wirtualne urządzenie połączone z adresatem. Numery jednostek logicznych są używane do adresowania wielu urządzeń peryferyjnych obsługiwanych przez wspólny sterownik lub wielu jednostek logicznych na dysku.
- a **Adres bloku logicznego.** W przypadku operacji odczytu/zapisu blok CDB zawiera początkowy adres transferu.
- a **Długość transferu.** W przypadku operacji odczytu/zapisu pole to precyzuje liczbę następujących po sobie bloków logicznych przewidzianych do przesłania.
- a **Długość listy parametrów.** Precyzuje liczbę bajtów wysyłanych w fazie „wyjście danych”. Pole to jest zwykle używane dla parametrów wysyłanych do adresata (np. parametry trybu, diagnostyki lub dziennika).
- a **Długość alokacji.** Precyzuje maksymalną liczbę bajtów, którą inicjator przydzielił danym zwrótnym.
- **Sterowanie.** Pole to zawiera bity połączenia i znaczniki sterujące mechanizmem łączenia rozkazów. Jeśli ustawiony jest bit połączenia, to bieżący rozkaz nie kończy się wraz z fazą „magistrala wolna”, lecz łączy się z następnym rozkazem, uruchamiając jego fazę „rozkaz”. Gdy ustawiony jest bit znacznika, adresat wysyła komunikat „połączony rozkaz zakończony”, jeśli wykonywanie rozkazu zakończyło się sukcesem, i zwraca tę samą wartość znacznika, która została przekazana przez CDB. Zwykle wartość 1 bitu znacznika jest używana przez inicjatora w celu wywołania przerwy między połączonymi rozkazami.

Specyfikacja SCSI definiuje szeroki zakres rodzajów rozkazów. Większość z nich ma charakter specyficzny, odnoszący się do określonego rodzaju urządzeń. Standard SCSI-2 obejmuje rozkazy dla następujących typów urządzeń:

- a urządzenia o dostępie bezpośrednim,
- urządzenia o dostępie sekwencyjnym,
- a drukarki,
- a procesorów,
- urządzenia z jednokrotnym zapisem,
- a napędu CD-ROM,
- skanera,
- a urządzenia pamięci optycznej,
- a urządzenia przełączania ośrodków,
- a urządzenia komunikacyjnego.

Ponadto występuje lista 17 rozkazów, które mają zastosowanie do wszystkich rodzajów urządzeń. Cztery spośród nich są obligatoryjne i muszą być wdrożone we wszystkich urządzeniach:

- a **Zapytanie.** Wymaga wysłania inicjatorowi parametrów adresata i połączonych z nim urządzeń peryferyjnych.
- a **Żądanie informacji odczytanej.** Wymaga wysłania inicjatorowi informacji odczytanej przez adresata. Odczytane dane obejmują warunki błędu (np. brak papieru), informacje o położeniu (np. koniec nośnika) oraz informacje o stanie logicznym (np. że rozkaz bieżący doszedł do znacznika ostatniego zapisu w pliku).
- a **Wysianie diagnostyki.** Wymaga wykonania przez adresata testu diagnostycznego samego siebie lub dołączonych urządzeń peryferyjnych, albo jednego i drugiego.
- a **Test gotowości jednostki.** Dostarcza środków do sprawdzenia, czy jednostka logiczna znajduje się w stanie gotowości.

Repertuar rozkazów SCSI jest najsilniejszą stroną tej specyfikacji. Umożliwia znormalizowany sposób obsługi większości urządzeń dołączanych do komputerów osobistych i stacji roboczych, upraszczając zadanie pisania oprogramowania wejścia-wyjścia dla systemów macierzystych.

Magistrala szeregową P1394

Wobec szybkości procesora sięgających 100 MHz i urządzeń pamięciowych przechowujących wiele gigabajtów danych wymagania wejścia-wyjścia odnoszące się do komputerów osobistych, stacji roboczych i serwerów są przerażające. Wciąż jeszcze techniki kanałów wejścia-wyjścia opracowane dla dużych komputerów i superkomputerów są zbyt drogie i zbyt obszerne dla tych mniejszych systemów. Wobec tego pojawiło się wielkie zainteresowanie opracowaniem szybkiego, alternatywnego systemu w stosunku do SCSI i innych interfejsów wejścia-wyjścia przeznaczonych dla małych systemów. Wyniki mają postać normy ANSI P1394 dotyczącej magistrali szeregową o wysokiej wydajności.

Standard P1394 ma wiele zalet w stosunku do SCSI i innych interfejsów wejścia-wyjścia. Magistrala P1394 jest bardzo szybka, tania i łatwa do wdrożenia. W rzeczywistości występuje ona nie tylko w systemach komputerowych, ale także w wyrobach elektroniki konsumpcyjnej, takich jak cyfrowe aparaty fotograficzne, magnetowidy i telewizory. W tych wyrobach magistrala P1394 służy do przekazywania obrazów wideo, które coraz częściej pochodzą ze źródeł zdyskretyzowanych.

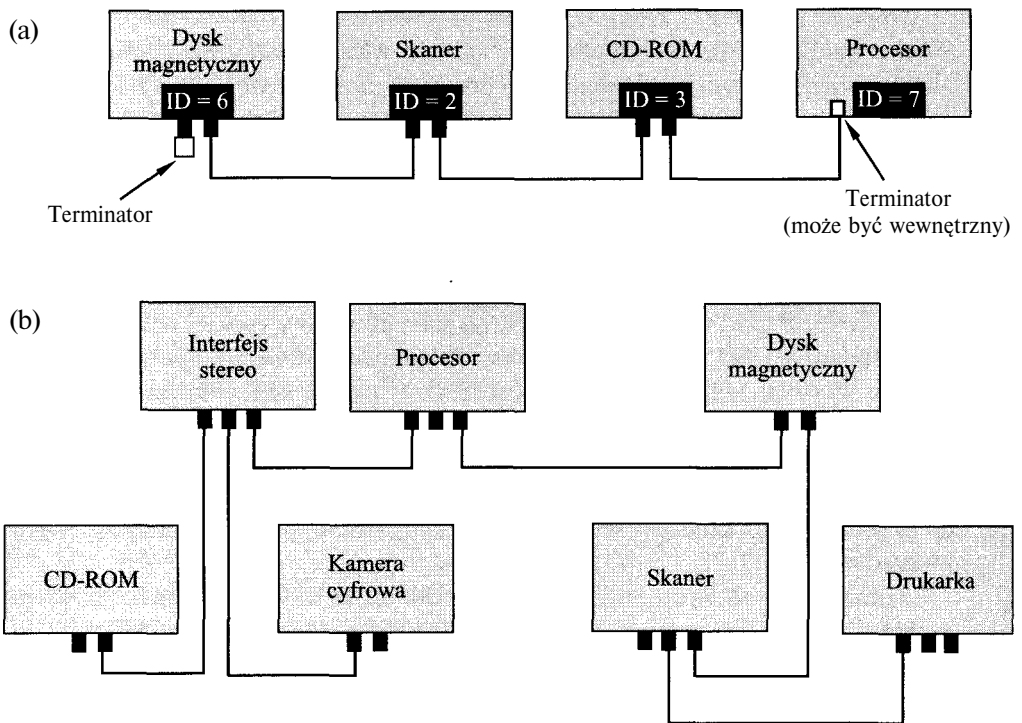
Jedną z silnych stron interfejsu P1394 jest to, że używa on transmisji szeregową (bit po bicie) zamiast równoległej. Interfejsy równoległe, takie jak SCSI, wymagają wielu przewodów, co oznacza szersze i droższe kable oraz szersze i droższe złącza z większą liczbą końcówek do wygięcia lub wyłamania. Kabel z większą liczbą przewodów wymaga ekranowania w celu zapobieżenia interferencji elektrycznej między przewodami. W przypadku interfejsów równoległych wymagana jest także synchronizacja, który to problem staje się coraz trudniejszy w miarę wzrostu długości kabla.

Ponadto komputery stają się fizycznie mniejsze, nawet jeśli zwiększa się ich moc obliczeniowa i wymagania co do wejścia-wyjścia. Komputery podręczne i kieszonkowe mają mało miejsca na złącza, wymagają jednak dużych szybkości transferu danych w celu operowania obrazami i informacjami wideo.

Magistrale P1394 opracowano w celu zapewnienia pojedynczego interfejsu wejścia-wyjścia z prostym złączem, mogącego współpracować z licznymi urządzeniami przez pojedynczy port, do którego mogą być podłączone myszka, drukarka laserowa, urządzenia SCSI, zewnętrzny napęd dyskowy, urządzenie audio oraz sieć lokalna. Samo złącze zostało zainspirowane przez złącze używane w Nintendo Gameboy. Jest tak wygodne, że użytkownik może sięgnąć za urządzenie i wetknąć wtyk bez patrzenia.

Konfiguracje P1394

Magistrala P1394 używa konfiguracji łańcuchowej, przy czym do pojedynczego portu można dołączyć do 63 urządzeń. Ponadto możliwe jest łączenie za pomocą mostów do 1022 magistral P1394, co umożliwia systemowi obsługę dowolnej wymaganej liczby urządzeń peryferyjnych.

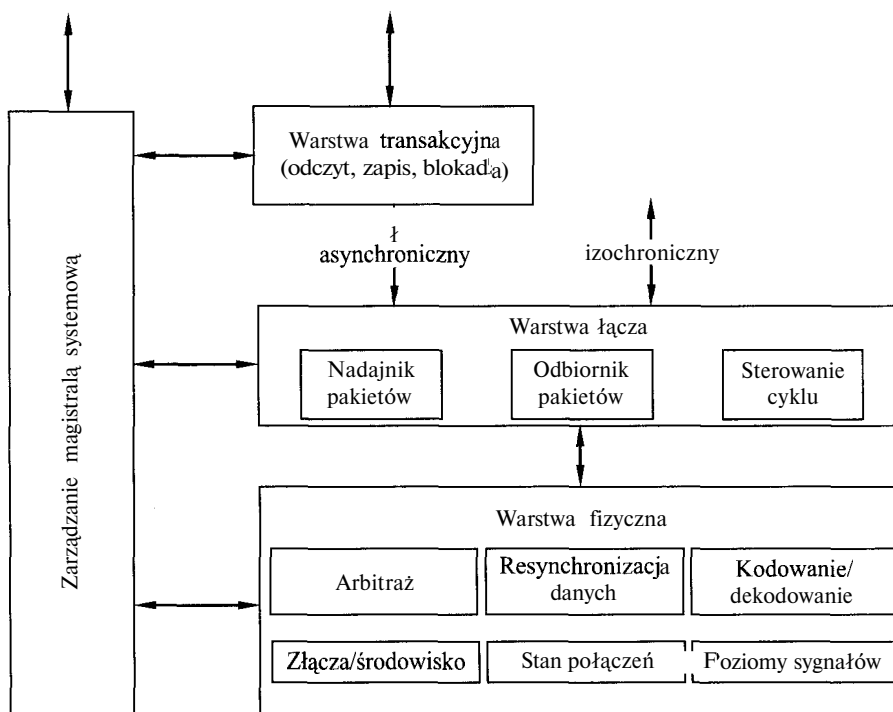


RYSunEK 6.19. Porównanie konfiguracji SCSI i P1394 (na podstawie „P1394 Technical summary”, Apple Computer, 1994): (a) przykładowa konfiguracja SCSI; (b) przykładowa konfiguracja P1394

Standard P1394 przewiduje możliwość tzw. aktywnego łączenia, dzięki czemu urządzenia peryferyjne mogą być dołączane i odłączane bez wyłączania zasilania systemu komputerowego i ponownego konfigurowania systemu. Zapewnia on również automatyczne konfigurowanie; nie jest konieczne ręczne ustawianie identyfikatorów ID lub interesowanie się wzajemnym położeniem urządzeń. Na rysunku 6.19 porównano konfigurację P1394 z konfiguracją SCSI. W przypadku SCSI oba końce magistrali muszą mieć terminatory, a każdemu urządzeniu musi być przypisany unikatowy adres jako część konfiguracji. Magistrala P1394 nie wymaga terminatorów, a system automatycznie realizuje funkcję konfiguracji, przypisując adresy. Zauważmy także, że magistrala P1394 nie musi mieć struktury ściśle łańcuchowej. Możliwa jest konfiguracja drzewiasta.

Ważną zaletą standardu P1394 jest to, że specyfikuje on zestaw trzech warstw protokołów normalizujących sposób współpracy systemu macierzystego z urządzeniami peryferyjnymi poprzez magistralę szeregową. Stos ten widać na rys. 6.20. Ma on trzy warstwy.

- a *Warstwa fizyczna.* Określa dopuszczalne ośrodki transmisji oraz ich charakterystyki elektryczne i sygnałowe.
- ❑ *Warstwa łącza.* Opisuje transmisję danych w pakietach.
- ❑ *Warstwa transakcyjna.* Definiuje protokół żądanie-odpowiedź, który ukrywa przed aplikacjami szczegóły niższych warstw P1394.



RYСУNEK 6.20. Stos protokołów P1394

Warstwa fizyczna

Warstwa fizyczna P1394 określa kilka alternatywnych ośrodków transmisji i ich złącza, różniące się własnościami fizycznymi i transmisyjnymi. Określono szybkości transferu danych od 25 do 400 Mbit/s. Warstwa fizyczna określa konwersję danych binarnych na sygnały elektryczne dla różnych ośrodków fizycznych. Umożliwia także obsługę arbitrażu, która gwarantuje, że w określonym momencie tylko jedno urządzenie będzie transmitowało dane.

Standard P1394 przewiduje dwie formy arbitrażu. Najprostsza forma jest oparta na wspomnianym wcześniej drzewiastym układzie węzłów magistrali P1394. Szczególnym przypadkiem tej struktury jest liniowa organizacja łańcuchowa. Warstwa fizyczna zawiera układy logiczne umożliwiające samokonfigurację wszystkich dołączonych urządzeń, dzięki czemu jeden węzeł jest określany jako korzeń a pozostałe są organizowane topologicznie jako drzewo. Gdy konfiguracja ta jest już ustalona, węzeł-korzeń działa jako arbiter centralny i przetwarza żądanie dostępu do magistrali metodą pierwszy zgłoszony - pierwszy obsłużony. W przypadku zapotrzebowań jednoczesnych dostęp uzyskuje węzeł o najwyższym priorytecie naturalnym. Priorytet naturalny wynika z bliskości rywalizujących węzłów do korzenia, a w przypadku równych odległości od korzenia o wygranej decyduje najniższy numer ID.

Powyższa metoda arbitrażu jest uzupełniona przez dwie dodatkowe funkcje: arbitrażu równorzędnego i arbitrażu pilnego. W przypadku arbitrażu równorzędnego czas magistrali jest organizowany w postaci przedziałów **równorzędności** (ang. *fairness intervals*). Na początku każdego przedziału każdy węzeł ustawia znacznik zezwolenia arbitrażu. W ramach przedziału każdy węzeł może rywalizować o dostęp do magistrali. Węzeł, po uzyskaniu dostępu do magistrali, ustawia swój znacznik zezwolenia na arbitraż i nie może w ramach tego samego przedziału ponownie starać się o dostęp do magistrali. Schemat ten czyni arbitraż bardziej sprawiedliwym, ponieważ zapobiega zmonopolizowaniu dostępu do magistrali przez jedno lub wiele „zapracowanych” urządzeń o wysokim priorytecie.

Obok schematu równorzędnego niektóre urządzenia mogą być skonfigurowane z priorytetem „pilny”. Węzły takie mogą przejmować sterowanie magistralą wielokrotnie w ramach przedziału równorzędności. W każdym węźle o wysokim priorytecie jest stosowany licznik umożliwiający wykorzystanie przez wszystkie węzły o wysokim priorytecie 75% dostępnego czasu magistrali. Na każdy pakiet, który jest transmitowany jako niepilny, trzy mogą być wysyłane jako pilne.

Warstwa łącza

Warstwa łącza określa transmisję danych w postaci pakietów. Obsługiwane są dwa rodzaje transmisji:

- *Asynchroniczna.* Zmienna ilość danych oraz kilka bajtów informacji warstwy transakcyjnej są wysyłane jako pakiet pod określonym adresem, po czym wraca potwierdzenie.
- a *Izochroniczna.* Zmienna ilość danych jest wysyłana w postaci szeregu pakietów o ustalonym rozmiarze w regularnych odstępach czasu. W tej formie transmisji stosowane jest uproszczone adresowanie i nie występuje potwierdzenie.

Transmisja asynchroniczna jest stosowana w przypadku danych, które nie wymagają ustalonej szybkości przesyłania. W przypadku transmisji asynchronicznej mogą być używane formy arbitrażu równorzędnego i opartego na pilności. Mniej przydatną metodą jest arbitraż równorzędny. Urządzenia wymagające znacznej części pojemności magistrali lub mające wysokie wymagania dotyczące opóźnień używają metody arbitrażu opartej na pilności. Na przykład, szybki węzeł zbierający dane w czasie rzeczywistym może wymagać pilnego arbitrażu, gdy bufor danych krytycznych są zapełnione co najmniej w połowie.

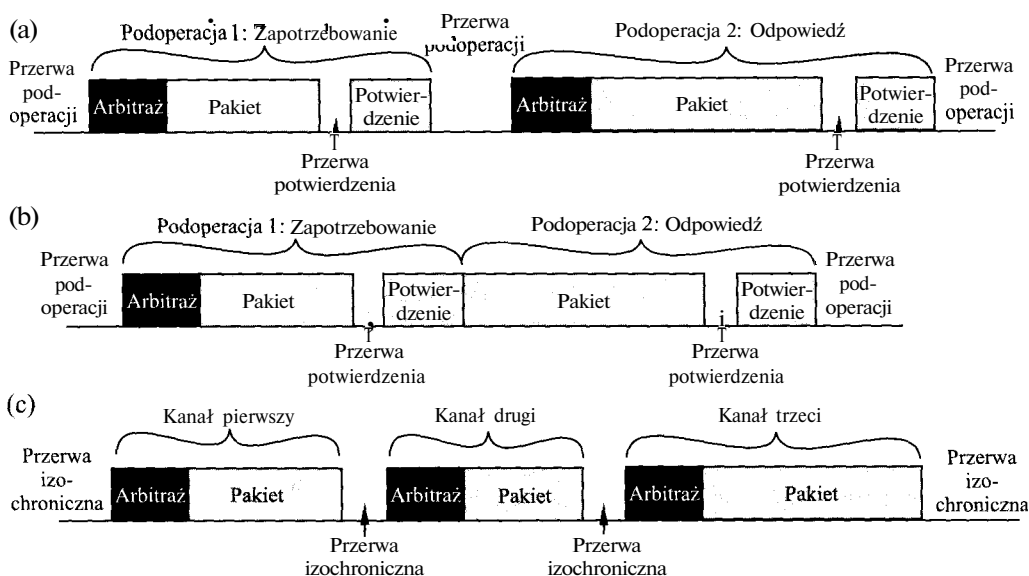
Na rysunku 6.21a jest pokazana typowa transakcja asynchroniczna. Proces dostarczania jednego pakietu jest nazywany **podoperacją** (ang. *subaction*). Podoperacja obejmuje pięć przedziałów czasowych. Są to:

- a **Sekwencja arbitrażu.** Polega na wymianie sygnałów wymaganej do uzyskania sterowania magistralą przez jedno urządzenie.
- a **Transmisja pakietu.** Każdy pakiet ma nagłówek zawierający identyfikator ID źródła i miejsca przeznaczenia. Zawiera on także informację o rodzaju pakietu, sumę kontrolną CRC * oraz informację o parametrach specyficznego rodzaju pakietu. Pakiet może także obejmować blok danych zawierających dane użytkownika i inne sumy CRC.
- a **Przerwa na potwierdzenie.** Jest to opóźnienie przewidziane dla odbiorcy w celu odbioru i zdekodowania pakietu oraz wygenerowania potwierdzenia.
- a **Potwierdzenie.** Odbiorca pakietu zwraca pakiet potwierdzenia wraz z kodem wskazującym na podejmowane przez siebie działanie.
- a **Przerwa podoperacji.** Jest to wymuszony okres jałowy, mający na celu zapewnienie, żeby inne węzły magistrali nie rozpoczęły arbitrażu, zanim nie zostanie wysłany pakiet potwierdzenia.

W momencie wysyłania potwierdzenia sterowanie magistralą należy do węzła potwierdzającego. Jeżeli więc wymiana polega na współpracy typu zapotrzebowanie-odpowiedź między dwoma węzłami, to węzeł odpowiadający może natychmiast transmitować pakiet odpowiedzi bez przechodzenia przez sekwencję arbitrażu (rys. 6.21b).

W przypadku urządzeń, które regularnie wysyłają lub przyjmują dane, takich jak cyfrowe urządzenia audio i wideo, przewidziano dostęp izochroniczny. Me-

* CRC - skrót od ang. *cyclic redundancy check* - cykliczna kontrola nadmiarowa (przyp. tłum.).



Rysunek 6.21. Podoperacje P1394: (a) przykładowe podoperacje asynchroniczne; (b) połączone podoperacje asynchroniczne; (c) przykładowe podoperacje izochroniczne

toda ta gwarantuje, że dane zostaną dostarczone z nieprzekraczalnym opóźnieniem z zagwarantowaną szybkością.

W celu opanowania mieszanego przepływu danych ze źródeł izochronicznych i asynchronicznych jeden z węzłów jest wyznaczany jako **węzeł nadrzędny** cyklu (ang. *cycle master*). Węzeł nadrzędny okresowo wydaje pakiet początku cyklu. Sygnalizuje to pozostałym węzłom, że rozpoczął się cykl izochroniczny. Podczas tego cyklu mogą być wysyłane tylko pakiety izochroniczne (rys. 6.21c). Każde izochroniczne źródło danych ubiega się o dostęp do magistrali. Węzeł wygrywający natychmiast transmituje pakiet. Pakiet ten nie podlega potwierdzaniu, dzięki czemu inne izochroniczne źródła danych natychmiast po przesłaniu poprzedniego pakietu rywalizują o magistralę. W rezultacie występuje niewielka przerwa między transmisją jednego pakietu a okresem arbitrażu dla następnego pakietu, podyktowana opóźnieniem na magistrali. Opóźnienie to, nazywane przerwą izochroniczną, jest mniejsze niż przerwa podoperacji.

Po zrealizowaniu transmisji przez wszystkie źródła izochroniczne magistrala pozostaje bezczynna przez czas wystarczający do wystąpienia przerwy podoperacji. Jest to sygnałem dla źródeł asynchronicznych, że mogą teraz rywalizować o dostęp do magistrali. Źródła asynchroniczne mogą następnie używać magistrali, aż do początku następnego cyklu izochronicznego.

Pakiety izochroniczne są etykietowane za pomocą 8-bitowych numerów kanału, przypisanych uprzednio podczas dialogu między dwoma węzłami, które mają wymieniać dane izochroniczne. Nagłówek, który jest krótszy niż w przypadku pakietów asynchronicznych, zawiera także pole długości danych i CRC nagłówka.

6.8.

POLECANA LITERATURA

Marcowy numer czasopisma *Computer* z roku 1994 jest poświęcony podsystemowi wejścia-wyjścia. Teksty dotyczące problemów wejścia-wyjścia są zawarte w [PATT94], [FELD94] i [MANO93]. Wnikliwą dyskusję modułów wejścia-wyjścia firmy Intel i ich architektury można znaleźć w [BREY95].

Dwoma niewielkimi książkami, które zawierają dobre wprowadzenie do SCSI, są [DEDE94] i [NCR90].

BREY95 Brey B.: *The Intel 32-bit Microprocessors: 80386, 80486, and Pentium*. Englewood Cliffs, Prentice Hall, 1995.

DEDE94 Dedek J.: *Basics of SCSI*. Menlo Park, Ancot Corp., 1994.

FELD94 Feldman J., Retter C.: *Computer Architecture*. New York, McGraw-Hill, 1994.

MANO93 Mano M.: *Computer System Architecture*. Englewood Cliffs, Prentice Hall, 1993.

NCR90 NCR Corp.: *SCSI: Understanding the Small Computer System Interface*. Englewood Cliffs, Prentice Hall, 1990.

PATT94 Patterson D., Hennessy J.: *Computer Organization and Design: the Hardware/Software Interface*. San Mateo, Morgan Kaufmann, 1994.

65

PROBLEMY DO ROZWIĄZANIA

- 6.1. W punkcie 6.3 wymieniliśmy jedną wadę i jedną zaletę wejścia-wyjścia odwzorowanego w pamięci w porównaniu z izolowanym wejściem-wyjściem. Wymień dwie inne wady i dwie inne zalety.
- 6.2. Dlaczego w przypadku przerwania wektorowych modułów wejścia-wyjścia umieszcza wektor na liniach danych, a nie na liniach adresu?
- 6.3. Praktycznie we wszystkich systemach zawierających moduły DMA dostęp DMA do pamięci głównej ma wyższy priorytet niż dostęp procesora do tej pamięci. Dlaczego tak jest?
- 6.4. Rozważ system dyskowy opisany w problemach 5.7 i 5.8. Procesor odczytuje jeden sektor dysku, używając wejścia-wyjścia sterowanego przerwaniem, przy jednym przerwaniu przypadającym na bit. Jeśli przetworzenie każdego przerwania zabiera 2,5 μ s, to jaką część czasu zajmie procesorowi obsługa wejścia-wyjścia (pomijając czas przeszukiwania)?
- 6.5. Powtórz problem 6.4, stosując DMA i zakładając jedno przerwanie na sektor.
- 6.6. Moduł DMA przesyła do pamięci znaki z urządzenia transmitującego z szybkością 9600 bit/s, stosując zajmowanie cyklu. Procesor pobiera rozkazy z szybkością miliona rozkazów na sekundę. O ile procesor będzie spowolniony przez moduł DMA?
- 6.7. Komputer 32-bitowy ma dwa kanały wybiórcze i jeden multiplekserowy. Każdy kanał wybiórczy obsługuje dwie jednostki dysków magnetycznych

i dwie jednostki taśmowe. Kanał multiplekserowy obsługuje zaś dwie drukarki wierszowe, dwa czytniki kart i dziesięć terminali wizyjnych. Założmy następujące szybkości transferu:

napęd dysków	800 KB/s
napęd taśmowy	200 KB/s
drukarka wierszowa	6,6 KB/s
czytnik kart	1,2 KB/s
terminal wizyjny	1 KB/s.

Oszacuj maksymalną łączną szybkość transferu w tym systemie.

- 6.8. Komputer składa się z procesora i urządzenia wejścia-wyjścia połączonego z pamięcią główną za pomocą wspólnej magistrali o szerokości 1 słowa. Procesor może wykonywać maksymalnie 10^5 rozkazów na sekundę. Przebieżny rozkaz wymaga 5 cykli maszynowych, z których 3 używają magistrali pamięci. Operacja odczytu lub zapisu pamięci zajmuje 1 cykl maszynowy. Załóżmy, że procesor w sposób ciągły realizuje programy podstawowe wykorzystujące 95% jego zdolności wykonywania rozkazów, jednak bez rozkazów wejścia-wyjścia. Urządzenie wejścia-wyjścia ma być teraz użyte do przesłania bardzo dużych bloków danych z (i do) pamięci głównej. Załóżmy też, że czas cyklu maszynowego jest równy czasowi cyklu magistrali.

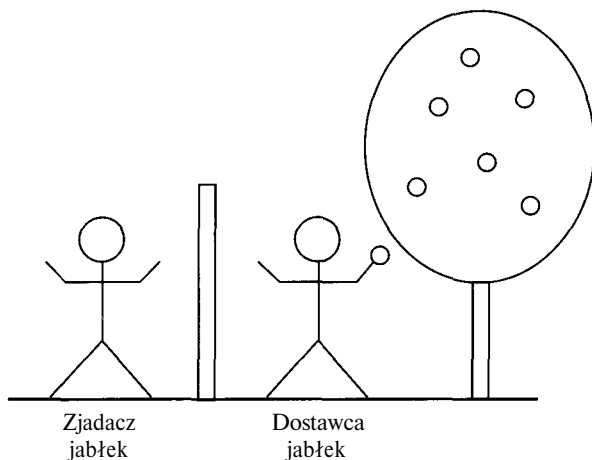
- (a) Jeśli stosowane jest programowane wejście-wyjście, a przesłanie każdego słowa wymaga od procesora wykonania 2 rozkazów, oszacuj maksymalną szybkość transferu danych z wejścia i na wyjście w słowach na sekundę, możliwą do uzyskania przez to urządzenie wejścia-wyjścia.
- (b) Oszacuj tę samą szybkość, jeśli używane jest DMA.

- 6.9. Źródło danych dostarcza 7-bitowych znaków ASCII, przy czym do każdego z nich jest dołączony bit parzystości. Wyprowadź wzór na maksymalną efektywną szybkość transferu danych (bitów danych ASCII) przez linię R (w bitach na sekundę) dla następujących przypadków:

- (a) transmisja asynchroniczna z bitem stopu 1,5 jednostki;
- (b) transmisja bitowo-synchroniczna, przy czym porcja danych zawiera 48 bitów sterowania i 128 bitów danych;
- (c) to samo co w (b), jednak pole informacyjne obejmuje 1024 bity;
- (d) transmisja znakowo-synchroniczna z 9 znakami sterowania i 16 znakami informacji na porcję;
- (e) to samo co w (d), jednak ze 128 znakami na porcję.

- 6.10. Następujący problem wykorzystuje ilustrację mechanizmu wejścia-wyjścia zawartą w [ECKE90] (rys. 6.22):

Dwaj chłopcy bawią się po obu stronach wysokiego ogrodzenia. Jeden z nich, nazwany Dostawcą jabłek, dysponuje piękną jabłonką obsypaną smacznymi jabłkami, rosnącą po jego stronie ogrodzenia. Jest on szczer-



Rysunek 6.22

śliwy, mogąc podawać jabłka drugiemu chłopcu, gdy tylko ten ich potrzebuje. Drugi chłopiec, nazwany Zjadaczem jabłek, lubi jeść jabłka, jednak nie ma żadnego. W rzeczywistości, może on jeść jabłka z ustaloną szybkością („jedno jabłko dziennie, a unikniesz wizyty u lekarza”). Jeśli jadłby jabłka szybciej, zachorowałby. Jeśli jadłby je wolniej, byłby niedożywiony. Żaden z chłopców nie może mówić, tak więc problem polega na dostarczaniu jabłek Zjadaczowi przez Dostawcę z właściwą szybkością.

- (a) Załóżmy, że na szczycie ogrodzenia znajduje się budzik umożliwiający wielokrotny alarm. Jak można użyć budzika do rozwiązania tego problemu? Narysuj wykres taktowania ilustrujący to rozwiązanie.
- (b) Załóżmy teraz, że nie ma budzika. Zamiast niego Zjadacz ma flagę, którą powiewa, gdy potrzebuje jabłka. Zasugeruj nowe rozwiązanie. Czy byłoby pomocne, gdyby Dostawca również dysponował flagą? Jeśli tak, to uwzględnij to w rozwiązaniu. Przedyskutuj wady tego rozwiązania.
- (c) Usuńmy teraz flagę i załóżmy istnienie długiego sznurka. Zasugeruj rozwiązanie wykorzystujące sznurek, które byłoby lepsze od (b).

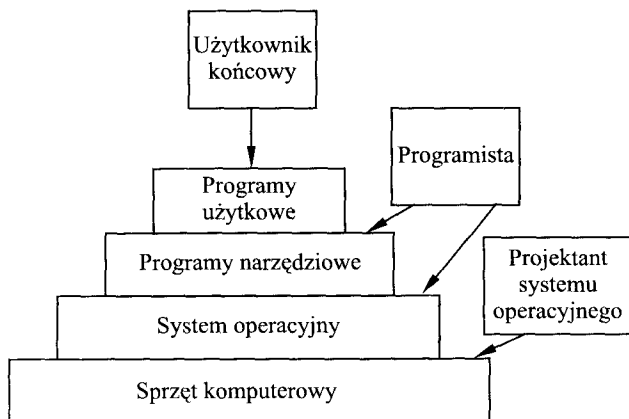
6.11. Załóżmy, że jeden 16-bitowy i dwa 8-bitowe mikroprocesory są dołączone do magistrali systemowej. Określone są następujące szczegóły:

1. Wszystkie mikroprocesory mają rozwiązania konieczne do wykonania dowolnego rodzaju transferu danych (programowane wejście-wyjście, wejście-wyjście sterowane przerwaniem i DMA).
2. Wszystkie mikroprocesory mają 16-bitową szynę adresową.
3. Dwa moduły pamięci, każdy o pojemności 64 KB, są dołączone do magistrali. Projektant chce użyć pamięci wspólnej tak dużej, jak tylko jest to możliwe.

System operacyjny jako interfejs użytkownik-komputer

W rozdziale I omówiliśmy hierarchiczną naturę systemu komputerowego, odnosząc się do sprzętu. Obraz ten może być poszerzony o oprogramowanie stosowane w komputerze, co widać na rys. 7.1. W większości przypadków ostatecznym przeznaczeniem komputerów jest wykonywanie jednego lub wielu programów użytkowych (aplikacji). Użytkownik tych aplikacji jest nazywany *użytkownikiem końcowym* i na ogół nie interesuje go architektura komputerów. Użytkownik końcowy patrzy na system komputerowy poprzez swój program użytkowy. Program ten może być wyrażony w języku programowania i jest przygotowywany przez *programistę tworzącego oprogramowanie użytkowe*. W dalszym ciągu stanie się jasne, że jeśli ktoś miałby opracować program użytkowy jako zestaw rozkazów maszynowych w pełni odpowiedzialnych za sterowanie sprzętem komputerowym, stałby przed przytłaczająco złożonym zadaniem. Dla ułatwienia tego zadania przewidziano zestaw *programów systemowych*. Niektóre spośród tych programów są określane jako *programy narzędziowe*. Umożliwiają one realizowanie często używanych funkcji, które wspomagają tworzenie programów, zarządzanie plikami oraz sterowanie urządzeniami wejścia-wyjścia. Programista wykorzystuje te ułatwienia, przygotowując program użytkowy, a program ten podczas pracy uruchamia programy narzędziowe w celu wykonania pewnych funkcji. Najważniejszym programem systemowym jest system operacyjny. System operacyjny maskuje przed programistą szczegóły sprzętowe i dostarcza mu wygodnego interfejsu z systemem komputerowym. Działa jako pośrednik, ułatwiając programiście i programom użytkowym dostęp do tych udogodnień i usług.

Mówiąc w skrócie, system operacyjny zapewnia zwykle usługi należące do następujących obszarów:



RYСУNEK 7.1. Warstwy i punkty postrzegania systemu komputerowego

- a *Tworzenie programów.* System operacyjny dostarcza wielu ułatwień i usług wspomagających programistę przy tworzeniu programów. Są to tak zwane programy narzędziowe (ang. *Utilities*).
- *Wykonywanie programów.* Aby program był wykonany, musi być zrealizowanych wiele zadań. Rozkazy i dane muszą być załadowane do pamięci głównej, urządzenia wejścia-wyjścia i pliki muszą być zainicjowane, potrzebne jest też przygotowanie pozostałych zasobów. Tym wszystkim zajmuje się system operacyjny.
- a *Dostęp do urządzeń wejścia-wyjścia.* Każde urządzenie wejścia-wyjścia do działania wymaga własnego, specyficznego zestawu rozkazów lub sygnałów sterowania. System operacyjny zajmuje się tymi szczegółami, dzięki czemu programista może myśleć w kategoriach prostych odczytów i zapisów.
- a *Kontrolowany dostęp do plików.* W przypadku plików sterowanie musi być dostosowane nie tylko do natury urządzeń wejścia-wyjścia (napędów dyskowych, napędów taśmowych), lecz także do formatu plików na nośniku przechowującym. I znów, o szczegóły troszczy się system operacyjny. Ponadto w przypadku systemów z wieloma jednoczesnymi użytkownikami system operacyjny może zapewnić mechanizmy ochrony, kontrolujące dostęp do wspólnych zasobów, takich jak pliki.
- a *Dostęp do systemu.* W przypadku systemów wspólnych lub publicznych system operacyjny kontroluje dostęp do systemu jako całości oraz do określonych zasobów systemu.

Nie będziemy dalej zajmować się usługami systemu operacyjnego. Przedmiotem naszego zainteresowania w tym rozdziale jest funkcja zarządzania zasobami przez system operacyjny oraz jej implikacje w stosunku do organizacji i architektury komputera.

System operacyjny jako program zarządzający zasobami

Komputer jest zbiorem zasobów służących do przenoszenia, przechowywania i przetwarzania danych oraz do sterowania tymi funkcjami. System operacyjny jest odpowiedzialny za zarządzanie tymi zasobami.

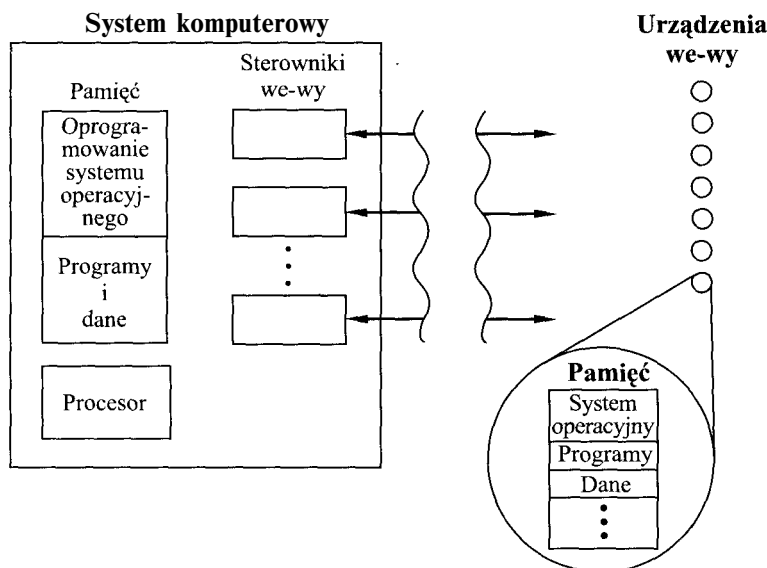
Czy możemy powiedzieć, że to system operacyjny steruje przenoszeniem, przechowywaniem i przetwarzaniem danych? Z pewnego punktu widzenia odpowiedź brzmi - tak, przez zarządzanie zasobami komputera system operacyjny steruje podstawowymi funkcjami komputera. Jednakże sterowanie to jest realizowane w dość osobliwy sposób. Zwykle traktujemy mechanizm sterowania jako coś zewnętrznego w stosunku do sterowanego obiektu, a przynajmniej jako wyraźnie wydzieloną część tego obiektu. Na przykład domowy system ogrzewania jest sterowany za pomocą termostatu, który jest oddzielony od urządzeń generujących i rozprowadzających ciepło. Nie jest tak w przypadku systemu

operacyjnego, który jako mechanizm sterujący jest niezwykle pod dwoma względami:

- Q System operacyjny funkcjonuje w ten sam sposób, jak zwykłe oprogramowanie komputera; to znaczy jest on programem wykonywanym przez procesor.
- System operacyjny często wyrzeka się sterowania i musi polegać na procesorze, aby odzyskać sterowanie.

System operacyjny nie jest w rzeczywistości niczym więcej, jak tylko programem komputerowym. Podobnie jak inne programy komputerowe dostarcza on rozkazy do procesora. Jedyną różnicą jest intencja programu. System operacyjny kieruje procesorem w zakresie używania pozostałych zasobów systemu oraz synchronizowania wykonywania przez procesor innych programów. Żeby jednak procesor mógł wykonywać te czynności, musi zaprzestać realizowania programu systemu operacyjnego i zacząć wykonywać inne programy. System operacyjny wyrzeka się więc sterowania procesorem, umożliwiając mu wykonanie użytecznej pracy, po czym wznawia sterowanie z wyprzedzeniem wystarczającym do przygotowania następnej pracy. Wykorzystywany tu mechanizm zostanie wyjaśniony w dalszej części rozdziału.

Na rysunku 7.2 są pokazane główne zasoby, którymi zarządza system operacyjny. Część systemu operacyjnego znajduje się w pamięci głównej. Należy do niej jądro, które zawiera najczęściej używane funkcje systemu operacyjnego, oraz - w określonym momencie - inne części systemu operacyjnego używane na bieżąco. Pozostała część pamięci głównej zawiera inne programy i da-



RYSunek 7.2. System operacyjny jako zarządca zasobów

ne. Jak zobaczymy, przydzielanie tych zasobów (pamięci głównej) jest sterowane wspólnie przez system operacyjny i sprzętowy mechanizm zarządzania pamięcią. System operacyjny decyduje, kiedy może być użyte urządzenie wejścia-wyjścia oraz steruje dostępem do plików i ich wykorzystaniem.

Rodzaje systemów operacyjnych

Do różnicowania systemów operacyjnych służą pewne kluczowe własności. Własności te mogą być rozpatrywane w dwóch niezależnych wymiarach. W pierwszym wymiarze precyzuje się, czy system jest **wsadowy**, czy **konwersacyjny**. W przypadku systemu **konwersacyjnego** (ang. *interactive*) ma miejsce konwersacja między użytkownikiem (programistą) a komputerem, zwykle za pośrednictwem terminala klawiatura/monitor, mająca na celu zgłoszenie zapotrzebowania na wykonanie określonej pracy lub przeprowadzenie transakcji. Ponadto użytkownik może, zależnie od natury zastosowania, komunikować się z komputerem podczas wykonywania zadania. System **wsadowy** (ang. *batch*) jest przeciwieństwem systemu konwersacyjnego. Programy wielu użytkowników są łączone (grupowane); powstaje wsad, który jest uruchamiany przez operatora komputera. Po zakończeniu wykonania programów wyniki są drukowane i przekazywane użytkownikom. Czysto wsadowe systemy są dzisiaj rzadkością. Jednak zwięzłe przeanalizowanie systemów wsadowych będzie użyteczne dla przedstawienia współczesnych systemów operacyjnych.

W drugim, niezależnym wymiarze precyzuje się, czy system stosuje **wieloprogramowanie**, czy nie. Za pomocą wieloprogramowania czynione jest staranie o maksymalne możliwe obciążenie procesora pracą, poprzez jednoczesne wykonywanie więcej niż jednego programu. Do pamięci ładuje się kilka programów, a procesor „przeskakuje” szybko między nimi. Alternatywą jest system **jednoprogramowy**, który w określonym czasie wykonuje tylko jeden program.

Posługując się tymi wymiarami, możemy przedstawić cztery ogólne rodzaje systemów operacyjnych, zebrane w tabeli 7.1. Najlepszym sposobem opisanie tych rodzajów jest przypomnienie krótkiej historii systemów operacyjnych.

TABELA 7.1. Podział systemów operacyjnych

	Wsadowy	Konwersacyjny
Jednozadaniowy	pojedynczy wsad	system wyspecjalizowany
Wieloprogramowy	złożony wsad	z podziałem czasu

Wczesne systemy

W przypadku najwcześniejszych komputerów programista współpracował bezpośrednio ze sprzętem komputerowym. Maszyny były kierowane z konsoli, zawierającej lampki wskaźnikowe, przełączniki dwustabilne, pewną postać urządzenia wejściowego i drukarkę. Programy w kodzie maszynowym były ładowa-

ne przez urządzenie wejściowe (np. czytnik kart). Jeśli błąd spowodował zatrzymanie programu, było to sygnalizowane przez lampki.

W miarę upływu czasu opracowano dodatkowy sprzęt i oprogramowanie. Uzupełnienia sprzętowe obejmowały magnetyczną pamięć taśmową i szybkie drukarki wierszowe. Dodatkowe oprogramowanie zawierało kompilatory, asembler i biblioteki powszechnie używanych funkcji. Powszechnie używane funkcje można było włączać do programów użytkowych bez konieczności ich ponownego programowania.

W tych wczesnych systemach występowały dwa główne problemy:

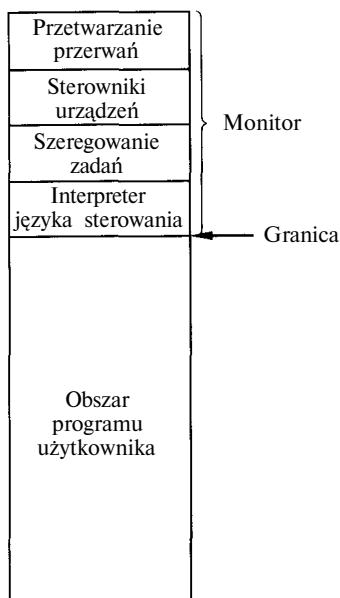
- a *Planowanie.* W większości systemów używano kartki papieru w celu zarezerwowania czasu maszyny, zwykle w wielokrotnościach pół godziny. Użytkownik mógł wystąpić o godzinę, a zakończyć pracę w ciągu 45 minut; wynikiem tego był tracony czas komputera. Użytkownik mógł też natrafić na problemy, nie ukończyć pracy w przydzielonym czasie i przerwać ją przed rozwiązaniem zadania.
- a *Czas na przygotowania.* Pojedynczy program, nazywany zadaniem, mógł obejmować ładowanie do pamięci kompilatora oraz programu w języku wysokiego poziomu (programu źródłowego), zachowanie programu skompilowanego (programu wynikowego), a następnie ładowanie i łączenie ze sobą programu wynikowego i powszechnie używanych funkcji. Każdy z tych kroków mógł zawierać zakładanie i wyjmowanie taśm oraz przygotowywanie paczek kart. W rezultacie tracono dużo czasu na przygotowanie programu do pracy.

Proste systemy wsadowe

Wczesne maszyny były bardzo kosztowne, dlatego tak ważne było maksymalne ich wykorzystanie. Czas tracony z powodu (sztywnego) planowania i przygotowań był nie do zaakceptowania.

W celu poprawienia wykorzystania opracowano proste wsadowe systemy operacyjne. W takim systemie, zwanym również **monitorem**, użytkownik nie miał już możliwości bezpośredniego dostępu do maszyny. Zamiast tego przekazywał zadanie w postaci kart lub taśmy operatorowi komputera, który porządkował sekwencyjnie zadania, tworząc wsad, po czym umieszczał wsad w urządzeniu wejściowym, do wykorzystania przez monitor (program zarządzający).

Aby zrozumieć funkcjonowanie tego schematu, rozpatrzmy jego działanie z dwóch punktów widzenia: programu zarządzającego i procesora. Z punktu widzenia programu zarządzającego to on właśnie steruje sekwencją zdarzeń. Żeby tak mogło być, program ten zawsze znajduje się w pamięci głównej (dlatego jest często nazywany **rezydentnym programem zarządzającym**) i zawsze jest gotowy do wykonywania (rys. 7.3). Program zarządzający wczytuje zadania, jedno w jednym czasie. Po wczytaniu bieżące zadanie jest lokowane w obszarze programu użytkowego, a sterowanie jest przekazywane temu właśnie zadaniu. Gdy



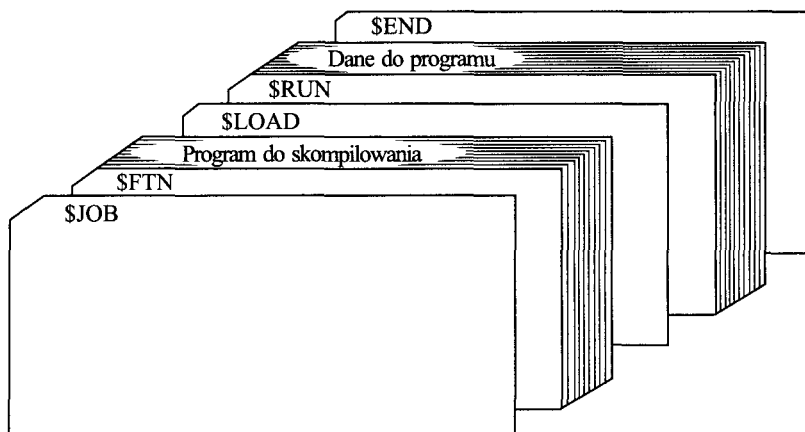
RYSUNEK 7.3. Rozkład pamięci w przypadku rezydentnego programu zarządzającego (monitora)

zadanie jest zakończone, następuje przerwanie (wewnętrzne w stosunku do procesora), które powoduje zwrócenie sterowania programowi zarządzającemu. Ten z kolei natychmiast wczytuje następne zadanie. Wyniki każdego zadania są drukowane i dostarczane użytkownikowi.

Rozważmy teraz tę sekwencję z punktu widzenia procesora. W pewnym momencie procesor wykonuje rozkazy pochodzące z tej części pamięci głównej, w której jest zawarty program zarządzający. Rozkazy te sprawiają, że do innej części pamięci jest wczytywane następne zadanie. Po wczytaniu zadania procesor napotyka w programie zarządzającym rozkaz rozgałęzienia, który nakazuje mu kontynuowanie pracy określonej w innej komórce pamięci (początek programu użytkownika). Procesor następnie wykonuje rozkazy programu użytkownika, aż do zakończenia lub natrafienia na błąd. Dowolne z tych zdarzeń skłoni procesor do pobrania następnego rozkazu z programu zarządzającego. Tak więc zdanie „sterowanie zostało przekazane zadaniu” oznacza po prostu, że procesor pobiera teraz i wykonuje rozkazy pochodzące z programu użytkownika. Natomiast zdanie „sterowanie jest zwrócone programowi zarządzającemu” oznacza, że procesor pobiera i wykonuje rozkazy z programu zarządzającego.

Widoczne jest, że program zarządzający rozwiązuje problem planowania (szeregowania). Zadania stanowiące wsad są ustawiane w kolejce i są realizowane tak szybko, jak to jest możliwe, bez czasu jałowego związanego z interwencjami.

Co się natomiast dzieje z problemem przygotowywania? Również i to rozwiązuje program zarządzający. Każdemu zadaniu towarzyszą rozkazy sformu-



RYSUNEK 7.4. Paczka kart dziurkowanych w przypadku prostego systemu wsadowego

łowane w języku sterowania zadaniami. Jest to specjalny rodzaj języka programowania używany do dostarczania instrukcji programowi zarządzającemu. Na rysunku 7.4 widać prosty przykład, w którym zadanie jest wprowadzane za pomocą kart. W tym przykładzie użytkownik dostarcza program napisany w FORTRAN-ie oraz dane przeznaczone do wykorzystania przez program. Jako uzupełnienie paczka kart zawiera rozkazy sterowania zadaniami, oznaczone na początku znakiem \$.

Widzimy, że program zarządzający (lub wsadowy system operacyjny) jest po prostu programem komputerowym. Umożliwia on procesorowi pobieranie rozkazów z różnych części pamięci głównej w celu albo przejmowania sterowania, albo rezygnowania z niego. Wymagane są również pewne inne własności sprzętowe:

- *Ochrona pamięci.* Wykonywanie programu użytkownika nie może powodować zmian w obszarze pamięci zajęтым przez program zarządzający. Jeśli ma miejsce takie użycie, to procesor wykrywa błąd i przekazuje sterowanie programowi zarządzającemu. Program ten porzuca zadanie, drukuje komunikat o błędzie i ładuje następne zadanie.
- Q *Czasomierz.* Czasomierz jest używany dla zapobiegania monopolizacji systemu przez pojedyncze zadanie. Jest on ustawiany na początku każdego zadania. Po dojściu do określonego wskazania, następuje przerwanie i sterowanie wraca do programu zarządzającego.
- Q *Rozkazy uprzywilejowane.* Pewne rozkazy są oznaczone jako uprzywilejowane i mogą być wykonywane tylko przez program zarządzający. Należą do nich rozkazy wejścia-wyjścia, a więc program zarządzający zachowuje sterowanie wszystkimi urządzeniami wejścia-wyjścia. Zapobiega to na przykład temu, żeby program użytkownika przypadkowo przyjął rozkazy sterowania pochodzące z następnego zadania. Jeśli z programu użytkownika

wynika potrzeba operacji wejścia-wyjścia, to musi on zwrócić się do programu zarządzającego, aby ten wykonał w jego imieniu tę operację. Jeśli procesor napotyka rozkaz uprzywilejowany podczas wykonywania programu użytkownika, to traktuje go jako błąd i przekazuje sterowanie programowi zarządzającemu.

Czas maszynowy jest więc dzielony między wykonywanie programów użytkownika a wykonywanie programu zarządzającego. Towarzyszą temu dwie straty: część pamięci głównej została oddana programowi zarządzającemu, a pewna część czasu maszynowego jest zużywana przez ten program. Nawet z tymi stratami, stosując prosty system wsadowy, poprawia się wykorzystanie komputera.

Wyrafinowane systemy wsadowe

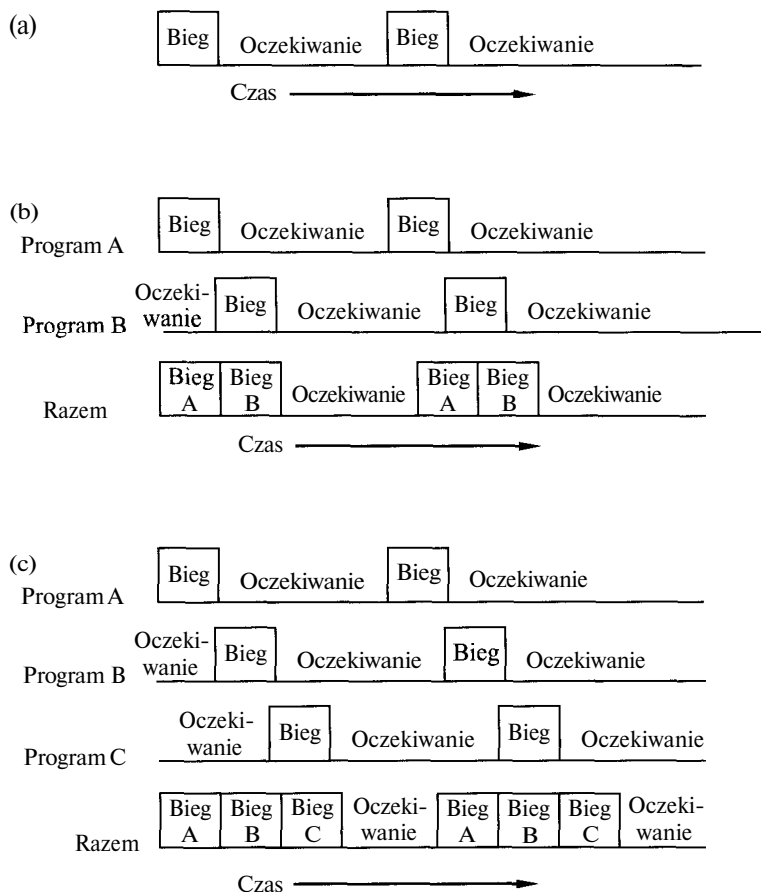
Nawet przy automatycznym szeregowaniu zadań realizowanym przez prosty wsadowy system operacyjny procesor często pozostaje niewykorzystany. Problem polega na tym, że urządzenia wejścia-wyjścia są powolne w porównaniu z procesorem. Na rysunku 7.5 są przedstawione reprezentatywne obliczenia. Dotyczą one programu, który przetwarza pliki rekordów i wykonuje przeciętnie 100 rozkazów maszynowych na 1 rekord. W tym przykładzie komputer spędza ponad 96% czasu, czekając na zakończenie transferu danych przez urządzenia wejścia-wyjścia! Na rysunku 7.6a zilustrowano tę sytuację. Procesor przez pewien czas wykonuje rozkazy, aż napotka rozkaz wejścia-wyjścia. Musi następnie czekać na zakończenie wykonywania rozkazu wejścia-wyjścia, zanim podejmie dalszą pracę.

Ta nieefektywność nie jest konieczna. Wiemy już, że pojemność pamięci musi być wystarczająca do pomieszczenia rezydentnego programu zarządzającego i jednego programu użytkownika. Załóżmy, że pojemność byłaby wystarczająca dla systemu operacyjnego i dwóch programów użytkowych. W takim przypadku, gdy jedno z zadań musi czekać na wejście lub wyjście, procesor może się przełączyć na inne zadanie, które prawdopodobnie nie czeka na wejście-wyjście (rys. 7.6b). Ponadto możemy rozszerzyć pamięć, umożliwiając zmieszczenie trzech, czterech lub większej liczby programów, po czym przełączać się między nimi (rys. 7.6c). Proces ten jest znany jako **wieloprogramowanie**. Jest on głównym problemem nowoczesnych systemów operacyjnych.

Odczytanie jednego rekordu	0,0015 s
Wykonanie 100 rozkazów	0,0001 s
Zapisanie jednego rekordu	0,0015 s
RAZEM	0,0031 s

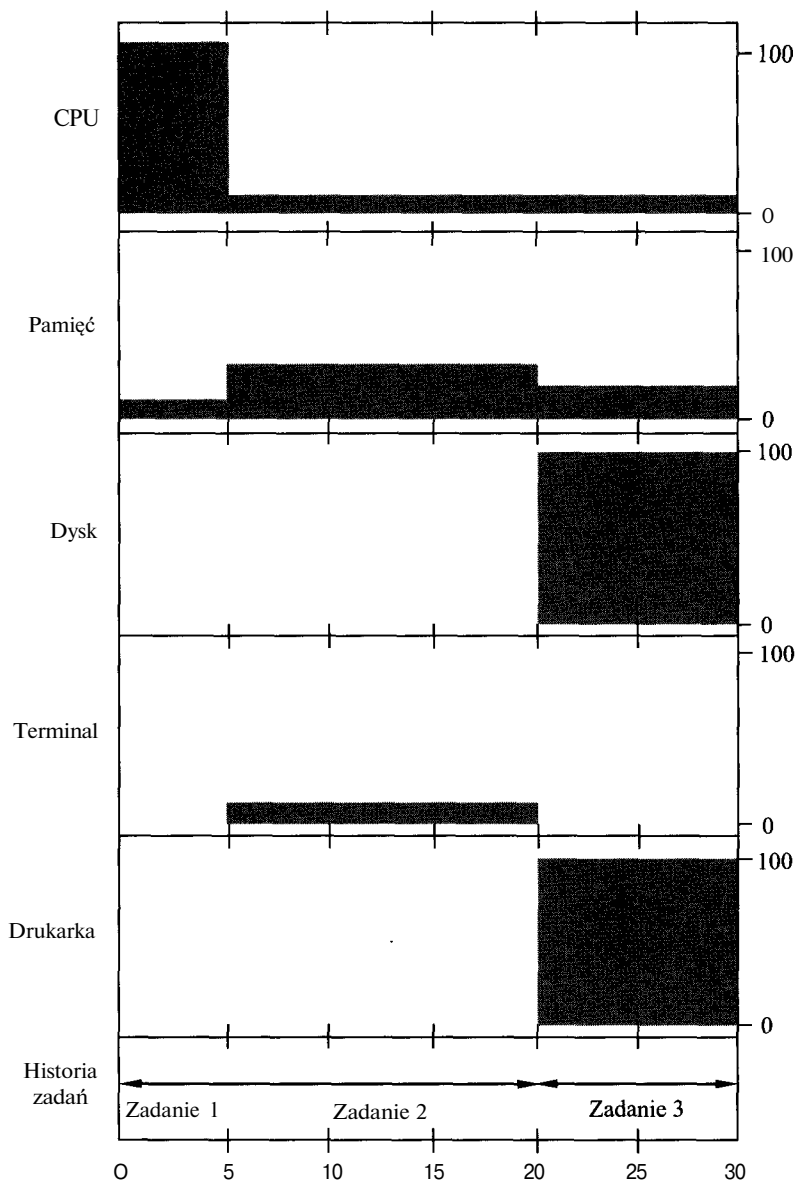
$$\text{Procentowe wykorzystanie CPU} = \frac{0,0001}{0,0031} = 0,032 = 3,2\%$$

RYSUNEK 7.5. Przykład wykorzystania systemu



RYСУNEK 7.6. Wieloprogramowanie: (a) jeden program; (b) dwa programy; (c) trzy programy

Aby zilustrować korzyści z wieloprogramowości, rozważmy przykład oparty na pracy [TURN86]. Dysponujemy komputerem o dostępnej (nie zajmowanej przez system operacyjny) pojemności pamięci 256 K słów, dyskiem, terminalem i drukarką. Trzy programy: ZADANIE1, ZADANIE2 i ZADANIE3, zostały skierowane do wykonywania w tym samym czasie. Mają one własności wymienione w tabeli 7.2. Załóżmy, że ZADANIE2 i ZADANIE3 mają minimalne wymagania w odniesieniu do procesora, a ZADANIE3 wymaga ciągłego używania dysku i drukarki. W środowisku jednoprogramowym zadania te byłyby realizowane kolejno, jedno po drugim. ZADANIE1 jest wykonywane w ciągu 5 minut. ZADANIE2 musi odczekać 5 minut, po czym jest wykonywane w ciągu 15 minut. ZADANIE3 jest więc rozpoczynane po 20 minutach, a jego realizacja pochłania dalsze 10 minut. Przeciętne wykorzystanie zasobów, przepustowość i czasy odpowiedzi są zestawione w kolumnie dotyczącej jednoprogramo-



RYSUNEK 7.7. Histogram wykorzystania w warunkach pojedynczego programu

Wieloprogramowe systemy operacyjne są dość złożone w porównaniu z *jednoprogramowymi*. Aby dysponować kilkoma zadaniami gotowymi do uruchomienia, muszą one być przechowywane w pamięci, wymagając tym samym pewnej formy *zarządzania pamięcią*. Ponadto, jeśli kilka zadań czeka w stanie gotowości, procesor musi zdecydować, które ma być realizowane, co wymaga pewnego *algorytmu szeregowania*. Koncepcje te są przedyskutowane w dalszym ciągu tego rozdziału.

W dzisiejszych czasach wymaganie przetwarzania konwersacyjnego może być - i często jest - zaspokajane przez użycie mikrokomputerów wyspecjalizowanych. Opcja ta nie była osiągalna w latach sześćdziesiątych, kiedy w większości komputery były duże i kosztowne. Zamiast tego wprowadzono koncepcję **podziału czasu**.

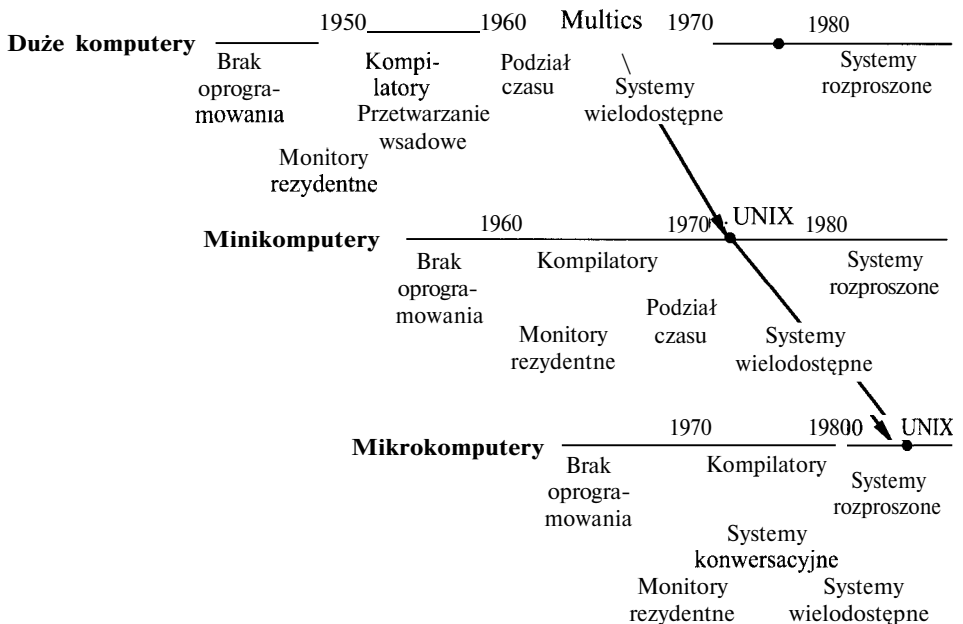
Wieloprogramowość umożliwia procesorowi jednoczesne wykonywanie wielu zadań wsadowych, może ona być także użyta do wielu zadań konwersacyjnych. W tym ostatnim przypadku metoda ta jest określana jako **tryb z podziałem czasu** (*time sharing*), co odzwierciedla fakt, że czas procesora jest dzielony między wielu użytkowników. Zarówno wieloprogramowe przetwarzanie wsadowe, jak i system z podziałem czasu posługują się wieloprogramowością. Podstawowe różnice między nimi są wymienione w tabeli 7.4.

TABELA 7.4. Porównanie programowania wsadowego z programowaniem w systemie z podziałem czasu

	Wieloprogramowanie wsadowe	System z podziałem czasu
Cel główny	maksymalne wykorzystanie procesora	minimalny czas odpowiedzi
Źródło rozkazów dla systemu operacyjnego	rozkazy sterujące wykonywaniem zadania dostarczane wraz z zadaniem	rozkazy wprowadzane z terminala

Klasy komputerów

Większość prac badawczo-rozwojowych dotyczących systemów operacyjnych była prowadzona dla potrzeb dużych komputerów. Gdy powstały pierwsze mi-



RYСУNEK 7.9. Zmiany koncepcji i własności systemów operacyjnych

nikomputery a następnie mikrokomputery i wciąż rosły możliwości sprzętowe tych systemów, technologia opracowana dla dużych komputerów była przenoszona do tych nowych klas komputerów.

Zjawisko to jest zilustrowane na rys. 7.9. Dobrym przykładem migracji koncepcji i cech jest ewolucja systemu UNIX od systemu Multics. Multics jest systemem operacyjnym dużych komputerów, opracowanym wspólnie przez MIT., Bell Laboratories i oddział General Electric, który następnie stał się częścią firmy Honeywell. Ten system operacyjny jest nadal sprzedawany przez firmę Honeywell i nadal jest uważany za system zaawansowany, szczególnie w zakresie zabezpieczeń. Wiele idei opracowanych dla systemu Multics zastosowano następnie w Bell Laboratories w pracach nad projektowaniem systemu UNIX, który stał się jednym z najbardziej popularnych minikomputerowych systemów operacyjnych i jest obecnie oferowany dla wielu mikrokomputerów.

72

SZEREGOWANIE CZASOWE

Głównym celem nowoczesnych systemów operacyjnych jest wieloprogramowanie (z wyjątkiem wyspecjalizowanych mikrokomputerów). W przypadku wieloprogramowania wiele zadań lub programów użytkowych pozostawia się w pamięci. Każde zadanie jest przełączane między używaniem centralnego procesora a oczekiwaniem na realizację operacji wejścia-wyjścia. Procesor pozostaje zajęty, wykonując jedno z zadań, podczas gdy inne oczekują.

TABELA 7.5. Szeregowanie w przypadku wieloprogramowania

Szeregowanie wysokiego poziomu	Decyzja dotycząca dodania do puli programów przeznaczonych do wykonywania
Szeregowanie krótkookresowe	Wybór tego spośród dostępnych procesów, który będzie wykonywany przez procesor
Szeregowanie operacji wejścia-wyjścia	Wybór tego spośród zawieszonych żądań wejścia-wyjścia, które ma być realizowane przez dostępne urządzenie wejścia-wyjścia

Kluczem do wieloprogramowania jest szeregowanie. Stosowane są zwykle trzy rodzaje szeregowania (tabela 7.5). Przeanalizujemy je teraz. Najpierw jednak wprowadzimy koncepcję **procesu** (ang. *process*). Wyrażenie to było użyte po raz pierwszy przez projektantów systemu Multics w latach sześćdziesiątych. Jest terminem nieco bardziej ogólnym niż zadanie. Podano wiele definicji terminu proces, w tym:

- a realizowany program,
- „animowany duch” programu,
- jednostka, do której przypisany jest procesor.

Koncepcja ta stanie się jaśniejsza w dalszym ciągu.

Szeregowanie wysokiego poziomu

Program szeregujący wysokiego poziomu określa, które programy są dopuszczone do przetwarzania przez system. Program ten steruje więc *stopniem wieloprogramowości* (liczbą procesów w pamięci). Po dopuszczeniu zadanie lub program staje się procesem i jest dodawane do kolejki zarządzanej przez krótkookresowy program szeregujący. Im więcej tworzy się procesów, tym mniejsza część czasu może być poświęcona na realizację każdego z nich. Długookresowy program szeregujący może ograniczyć stopień wieloprogramowości w celu zapewnienia zadowalającej obsługi bieżącemu zespołowi procesów.

W przypadku systemu wsadowego nowe zadania są kierowane na dysk i trzymane w kolejce. Program szeregujący wysokiego poziomu pobiera zadania z kolejki, gdy jest to możliwe. Muszą tu być podjęte dwie decyzje. Po pierwsze, program szeregujący musi zdecydować, czy może pobrać jeden czy wiele dodatkowych procesów. Decyzja ta jest zwykle podejmowana za każdym razem po zakończeniu istniejącego procesu. Po drugie, program szeregujący musi zdecydować, które zadanie lub zadania mają być akceptowane i zamienione na procesy. Stosowane kryteria mogą obejmować priorytet, czas realizacji i wymagania co do wejścia-wyjścia.

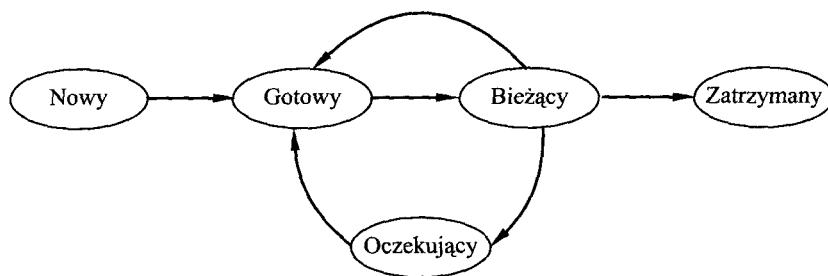
W przypadku podziału czasu zapotrzebowanie na proces jest generowane przez działanie użytkownika, który chce się dołączyć do systemu. Użytkownicy systemu z podziałem czasu nie są po prostu ustawiani w kolejce i trzymani w niej, aż do momentu akceptacji ich przez system. System operacyjny będzie raczej akceptował wszystkich autoryzowanych użytkowników aż do nasycenia systemu. Wówczas żądanie dołączenia się do systemu spotka się z odpowiedzią, że system jest w pełni wykorzystany, a użytkownik powinien spróbować ponownie później.

Szeregowanie krótkookresowe

Program szeregujący wysokiego poziomu działa stosunkowo rzadko i podejmuje ogólne decyzje, czy przyjąć nowy proces i który proces ma być przyjęty. Krótkookresowy program szeregujący, znany również jako **dyspozytor** (ang. *dispatcher*), działa często i podejmuje bardziej szczegółowe decyzje, które zadanie ma być realizowane jako następne.

Stany procesu

Aby zrozumieć działanie dyspozytora, musimy rozważyć koncepcję stanu procesu. Podczas czasu życia procesu, jego status zmienia się wielokrotnie. Status procesu w dowolnym czasie jest nazywany stanem. Użyty został termin „stan”, ponieważ oznacza on jednocześnie, że istnieje pewna informacja definiująca status w tym momencie. Definiuje się zwykle pięć stanów procesu (rys. 7.10):



RYSUNEK 7.10. Stany procesu

- a **Nowy.** Program jest przyjęty przez program szeregujący wysokiego poziomu, lecz nie jest gotowy do realizacji. System operacyjny inicjuje proces, przesuwając go w ten sposób do stanu gotowości.
- a **Gotowy.** Proces jest gotowy do wykonywania i czeka na dostęp do procesora.
- a **Bieżący.** Proces jest realizowany przez procesor.
- a **Oczekujący.** Realizacja procesu jest zawieszona podczas oczekiwania na pewne zasoby systemu, np. wejście-wyjście.
- a **Zatrzymany.** Proces został zakończony i zostanie zniszczony przez system operacyjny.

Dla każdego procesu w systemie system operacyjny musi utrzymywać informację o stanie, wskazującą na status procesu oraz na inne cechy istotne do realizacji procesu. W tym celu każdy proces jest reprezentowany w systemie operacyjnym przez **blok kontrolny procesu** (rys. 7.11). W bloku kontrolnym procesu są podane:

- Q Identyfikator.** Każdy aktualny proces ma unikatowy identyfikator.
- Q Stan.** Aktualny stan procesu (nowy, gotowy itd.).
- a **Priorytet.** Względny poziom priorytetu.
- a **Licznik programu.** Adres następnego rozkazu w programie, który ma być wykonywany.
- ☐ **Znaczniki pamięci.** Początkowa i końcowa komórka zajmowane przez proces w pamięci.
- a **Dane dotyczące kontekstu.** Są to dane obecne w rejestrach procesora podczas realizacji procesu; będą one przedyskutowane w części III. Teraz wystarczy stwierdzić, że dane te reprezentują „kontekst” procesu. Kontekst i licznik programu są zachowywane, gdy proces przestaje być w stanie gotowości. Są pobierane przez procesor, gdy wznawia on realizację procesu.
- Q Informacja o stanie wejścia-wyjścia.** Obejmuje wyróżniające się żądania wejścia-wyjścia, urządzenia wejścia-wyjścia (np. napędy taśmowe) przypisane do procesu, listę plików przypisanych do procesu itd.

Identyfikator
Stan
Priorytet
Licznik rozkazów
Wskaźniki pamięci
Dane kontekstowe
Informacje o stanie we-wy
Informacja ewidencyjna
⋮

RYSUNEK 7.11. Blok kontrolny procesu

a Informacja ewidencyjna. Może obejmować wymagany czas procesora i czas zegara, ograniczenia czasowe, liczby ewidencyjne itd.

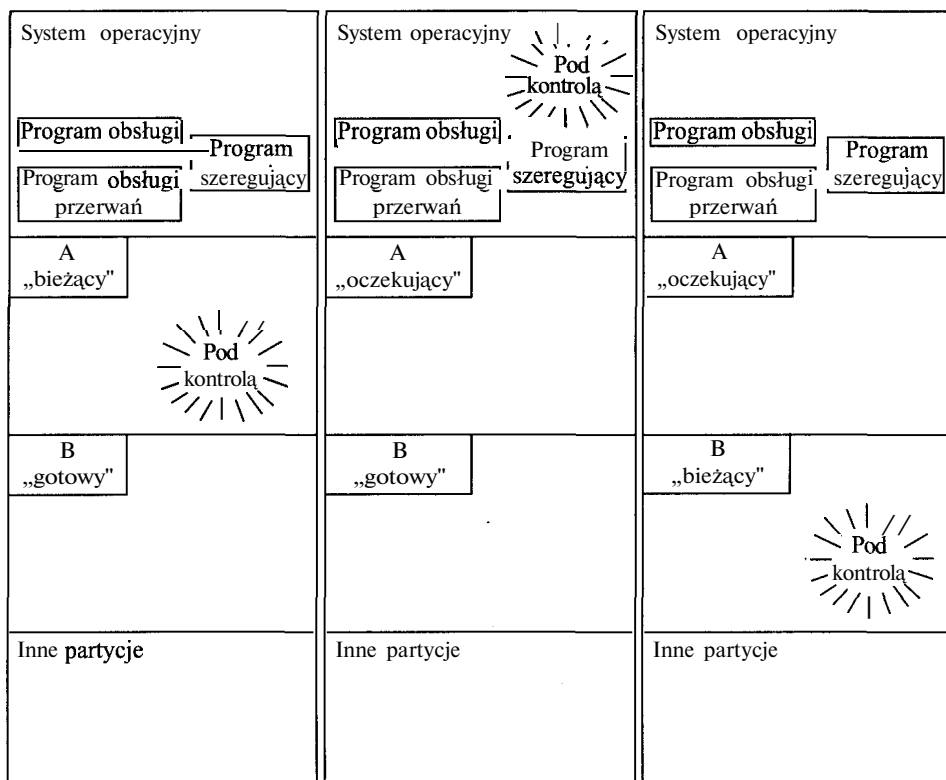
Gdy procesor akceptuje do wykonania nowe zadanie lub żądanie użytkownika, tworzy czysty blok kontrolny procesu i umieszcza związany z nim proces w stanie „nowy”. Po wypełnieniu przez system bloku kontrolnego procesu, proces jest przenoszony do stanu „gotowy”.

Metody szeregowania

Aby zrozumieć, jak system operacyjny zarządza szeregowaniem różnych zadań w pamięci, rozpocznijmy od rozważenia prostego przykładu z rys. 7.10. Na rysunku widać, jak w określonym momencie jest podzielona pamięć. Oczywiście, jądro systemu operacyjnego zawsze jest rezydentne. Ponadto występuje pewna liczba aktywnych procesów, w tym A i B, z których każdy jest rozmieszczony w części pamięci.

Rozpocznijmy w momencie (rys. 7.12a), gdy jest realizowany proces A. Procesor wykonuje rozkazy programu zawartego w części pamięci przypisanej do A. W pewnym późniejszym momencie (rys. 7.12b) procesor zaprzestaje wykonywania rozkazów procesu A i rozpoczyna wykonywanie rozkazów przechowywanych w obszarze systemu operacyjnego. Następuje to z jednego z trzech powodów:

1. Proces A przekazuje wywołanie obsługi (np. żądanie wejścia-wyjścia) systemowi operacyjnemu. Wykonywanie procesu A jest więc zawieszone do czasu spełnienia żądania przez system operacyjny.



RYSUNEK 7.12. Przykład szeregowania

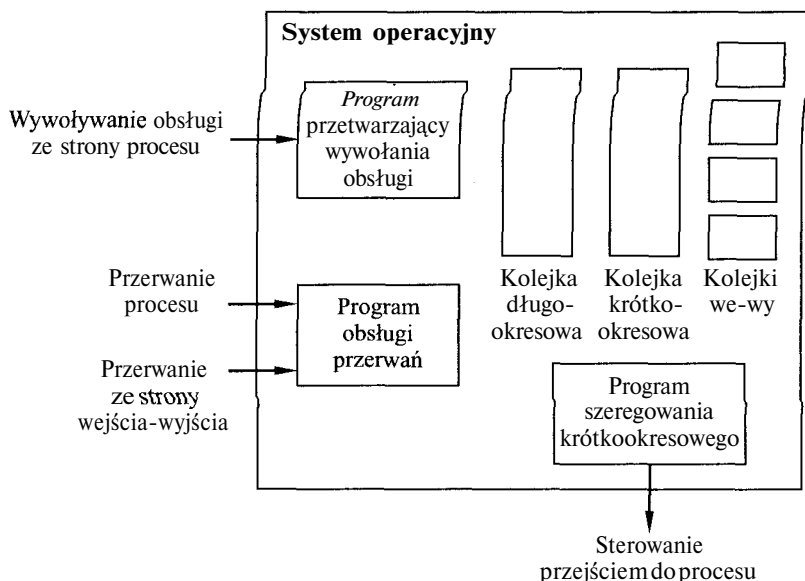
- Proces A powoduje *przerwanie*. Przerwanie jest sygnałem wysyłanym do procesora generowanym sprzętowo. Gdy sygnał ten zostaje wykryty, procesor zaprzestaje wykonywania procesu A i przenosi się do programu obsługi przerwania w systemie operacyjnym. Różne zdarzenia związane z procesem A mogą spowodować przerwanie. Jedną z możliwości jest błąd, taki jak przystąpienie do wykonywania rozkazu uprzywilejowanego. Innym przykładem jest wyczerpanie się czasu; w celu zapobieżenia monopolizacji procesora przez jeden proces, każdy proces uzyskuje dostęp do procesora tylko na krótki czas.
- Pewne zdarzenie nie związane z procesem A, lecz wymagające uwagi, powoduje przerwanie. Przykładem jest zakończenie operacji wejścia-wyjścia.

W każdym przypadku wynik jest następujący. Procesor zapisuje bieżące dane kontekstowe i licznik programu A w bloku kontrolnym procesu A, po czym przystępuje do działania w systemie operacyjnym. System operacyjny może wykonać pewną pracę, taką jak zainicjowanie operacji wejścia-wyjścia. Potem należący do systemu operacyjnego program szeregowania krótkookresowego de-

cyduje, który proces powinien być realizowany jako następny. W tym przykładzie został wybrany proces B. System operacyjny poleca procesorowi odnowienie danych kontekstu B i przystąpienie do realizowania procesu B w punkcie, w którym był on pozostawiony (rys. 7.12c).

Ten prosty przykład ujawnia podstawową funkcję programu szeregowania krótkookresowego. Na rysunku 7.13 są pokazane główne elementy systemu operacyjnego zaangażowane w wieloprogramowość oraz szeregowanie procesów. System operacyjny uzyskuje sterowanie procesorem przez swój program obsługi przerwań, jeśli nastąpiło przerwanie, lub przez program obsługi wezwań serwisowych, jeśli wystąpiło wezwanie do obsługi. Gdy przerwanie lub wezwanie serwisu jest przetwarzane, program szeregowania krótkoterminowego jest wzywany do wybrania procesu przeznaczonego do realizacji.

W celu wykonania tego zadania system operacyjny utrzymuje pewną liczbę kolejek. Każda kolejka jest po prostu listą procesów czekających na pewne zasoby. **Kolejka długookresowa** jest listą zadań czekających na użycie systemu. Gdy warunki pozwolą, program szeregujący wysokiego poziomu dokona alokacji pamięci i utworzy proces z jednej spośród czekających jednostek. **Kolejka krótkookresowa** składa się ze wszystkich procesów będących w stanie gotowości. Każdy z tych procesów mógłby korzystać z procesora jako następny. Do programu szeregowania krótkookresowego należy wybranie jednego z nich. Na ogół jest to robione za pomocą algorytmu cyklicznego, co powoduje przydzielenie czasu kolejno każdemu procesowi. Mogą też być użyte poziomy priorytetu. Jest wreszcie **kolejka wejścia-wyjścia** związana z każdym urządzeniem wej-



RYСУNEK 7.13. Główne składniki systemu operacyjnego w przypadku wieloprogramowania

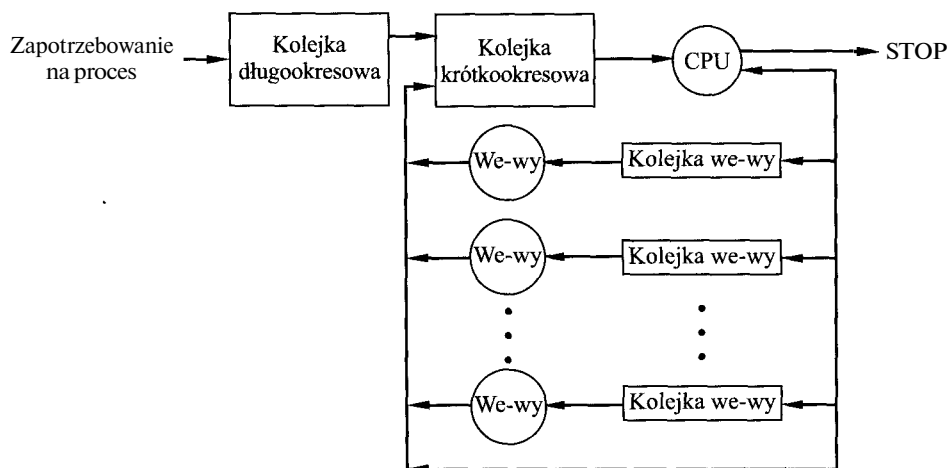
ścia-wyjścia. Wiele procesów może domagać się użycia tego samego urządzenia wejścia-wyjścia. Wszystkie procesy czekające na użycie określonego urządzenia są ustawiane w jego kolejce.

Na rysunku 7.14 jest pokazane, co dzieje się z procesem w komputerze pod kontrolą systemu operacyjnego. Każde zapotrzebowanie na proces (zadanie wsadowe, zadanie konwersacyjne definiowane przez użytkownika) jest umieszczane w kolejce długookresowej. Gdy zasoby stają się osiągalne, zapotrzebowanie na proces staje się procesem, jest przenoszone do stanu gotowości i umieszczane w kolejce krótkookresowej. Procesor wykonuje na przemian rozkazy systemu operacyjnego oraz procesów użytkowych. Gdy sterowanie należy do systemu operacyjnego, decyduje on, który spośród procesów czekających w kolejce krótkookresowej ma być realizowany jako następny. Gdy system operacyjny zakończy swoje pilne zadania, zleca procesorowi realizację wybranego zadania.

Jak wspomnieliśmy wcześniej, realizowany proces może być z wielu powodów zawieszony. Jeśli jest zawieszony, ponieważ zgłosił żądanie wejścia-wyjścia, to zostaje umieszczony w odpowiedniej kolejce wejścia-wyjścia. Jeśli natomiast powodem zawieszenia było wyczerpanie się czasu lub konieczność przejścia systemu operacyjnego do pilniejszych zadań, to jest przenoszony do stanu gotowości i powraca do kolejki krótkookresowej.

Zauważmy na koniec, że system operacyjny zarządza również kolejkami wejścia-wyjścia. Gdy operacja wejścia-wyjścia jest zakończona, system operacyjny usuwa zaspokojony już proces z kolejki wejścia-wyjścia i umieszcza go w kolejce krótkoterminowej. Następnie wybiera inny czekający proces (jeśli taki istnieje) i sygnalizuje urządzeniu wejścia-wyjścia zapotrzebowanie zgłoszone przez ten proces.

Stwierdziliśmy, że głównym zadaniem systemu operacyjnego jest zarządzanie zasobami systemu. Na rysunku 7.14 widać, jak system operacyjny zarządza



czasem procesora i urządzeniami wejścia-wyjścia za pomocą kolejek. Pozostaje jeszcze problem zarządzania zasobami pamięci, czym zajmiemy się w dalszym ciągu.

73

ZARZĄDZANIE PAMIĘCIĄ

W systemie jednoprogramowym pamięć główna jest podzielona na dwie części: jedną przeznaczoną dla systemu operacyjnego (rezydentnego programu zarządzającego) i drugą dla programu aktualnie realizowanego. W systemie wieloprogramowym część pamięci przypisana użytkownikowi musi być dalej podzielona w celu przyjęcia wielu procesów. Zadanie tego dalszego podziału jest realizowane dynamicznie przez system operacyjny i nosi nazwę **zarządzania pamięcią**.

Sprawne zarządzanie pamięcią jest niezwykle ważne w systemie wieloprogramowym. Jeśli w pamięci znajduje się niewiele procesów, to znaczną część czasu wszystkich procesów zabierze oczekiwanie na wejście-wyjście i procesor będzie „bezrobotny”. Wobec tego pamięć musi być sprawnie zarządzana w celu upakowania w pamięci możliwie dużej liczby procesów.

Wymiana

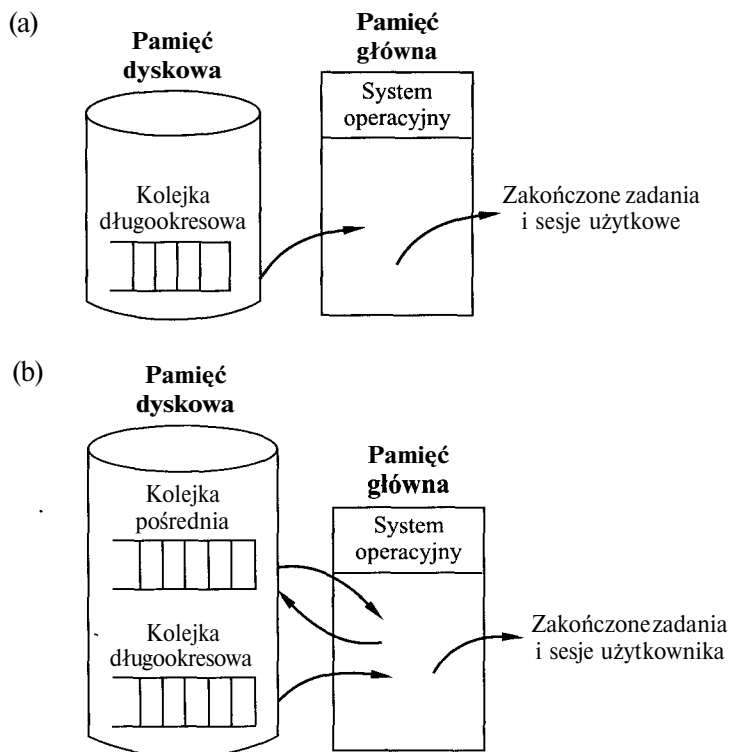
Wróćmy do rysunku 7.14. Przedyskutowaliśmy trzy rodzaje kolejek: długookresową kolejkę zapotrzebowań na nowe procesy, kolejkę krótkookresową procesów gotowych do użycia procesora i różne kolejki wejścia-wyjścia procesów, które nie są gotowe do użycia procesora. Zauważmy, że przyczyną tej rozbudowanej struktury jest to, że operacje wejścia-wyjścia są o wiele wolniejsze niż obliczenia i dlatego właśnie procesor w systemie jednoprogramowym pozostaje bezczynny przez większość czasu.

Jednak schemat przedstawiony na rys. 7.14 nie rozwiązuje całkowicie problemu. Prawdą jest, że w tym przypadku pamięć zawiera wiele procesów i procesor może przenieść się do innego procesu, jeśli jeden proces jest w stanie oczekiwania. Jednak procesor jest na tyle szybszy od wejścia-wyjścia, że często wszystkie procesy w pamięci będą czekały na wejście lub wyjście. Wobec tego nawet w warunkach wieloprogramowości procesor może być bezczynny przez większość czasu.

Co więc robić? Można poszerzyć pamięć główną, dzięki czemu będzie ona mogła przyjąć większą liczbę procesów. To rozwiązanie ma jednak dwie wady. Po pierwsze, pamięć główna jest kosztowna, nawet dzisiaj. Po drugie, apetyt programów na pamięć wzrasta tak samo szybko, jak maleje koszt pamięci. Większa pamięć oznacza więc większe procesy, a nie więcej procesów.

Innym rozwiązaniem jest **wymiana** (ang. *swapping*), pokazana na rys. 7.15. Mamy tutaj długookresową kolejkę zapotrzebowań na procesy, zwykle przechowywaną na dysku. Procesy są wprowadzane jeden po drugim, w miarę pojawiania się wolnej przestrzeni. Gdy są zakończone, wyprowadza się je z pamięci głównej. Załóżmy teraz, że wystąpiła sytuacja, w której żaden z procesów w pamięci nie znajduje się w stanie gotowości. Zamiast pozostawać bezczynnym, procesor przekazuje jeden z tych procesów na dysk, do **kolejki pośredniej**. Jest to kolejka procesów, które zostały czasowo usunięte z pamięci głównej. Następnie system operacyjny pobiera inny proces z kolejki pośredniej lub honoruje nowe zapotrzebowanie na proces z kolejki długookresowej, po czym realizowany jest nowo przybyły proces.

Wymiana jest jednakże operacją wejścia-wyjścia, istnieje więc potencjalna możliwość pogorszenia, a nie poprawienia sytuacji. Ponieważ jednak dyski są zwykle najszybszymi urządzeniami wejścia-wyjścia w systemie (np. w porównaniu z pamięcią taśmową lub drukarką), wymiana na ogół powoduje poprawę wydajności. Bardziej wyrafinowany schemat, obejmujący pamięć wirtualną, umożliwia jeszcze znaczącą poprawę wydajności niż prosta wymiana. Omówimy to pokrótce. Najpierw jednak musimy przygotować grunt poprzez wyjaśnienie partycjonowania i stronicowania.



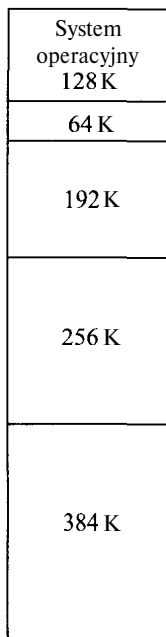
RYСУNEK 7.15. Zastosowanie wymiany: (a) proste szeregowanie zadań; (b) wymiana

Partycjonowanie

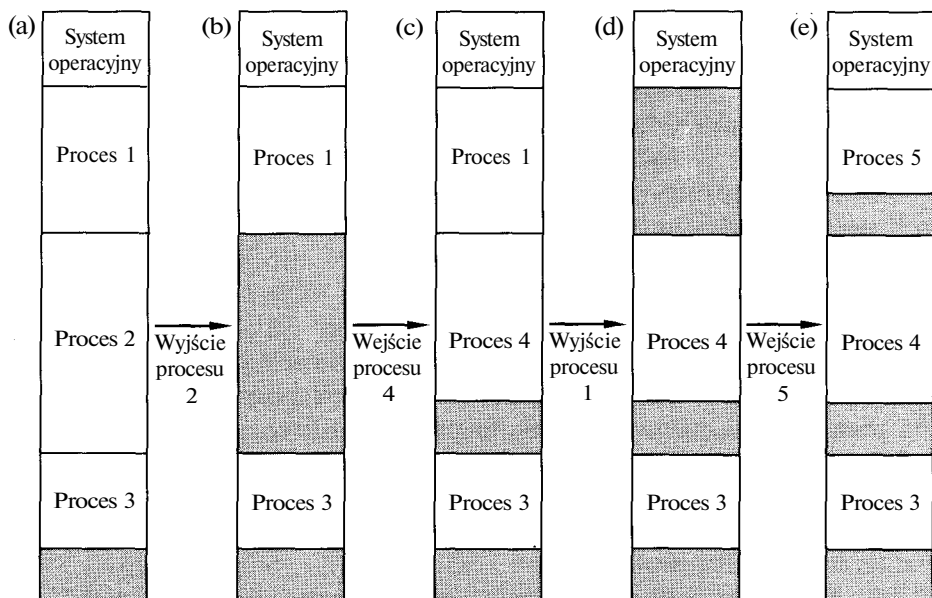
System operacyjny zajmuje ustaloną część pamięci głównej. Reszta pamięci jest partycjonowana (dzielona), aby mogło z niej korzystać wiele procesów. Najprostszym schematem partycjonowania dostępnej pamięci jest stosowanie partycji (części) o *ustalonym rozmiarze*, co widać na rys. 7.16. Zauważmy, że chociaż partycje mają ustalone rozmiary, jednak nie są one równe. Gdy proces jest wprowadzany do pamięci, jest umieszczany w najmniejszej dostępnej partycji, która może go pomieścić.

Nawet jeżeli używa się nierównych partycji o ustalonych rozmiarach, wystąpią straty pojemności pamięci. W większości przypadków proces nie będzie wymagał dokładnie takiej pojemności pamięci, jaką zapewnia partycja. Na przykład proces wymagający 128 KB pamięci zostanie umieszczony w partycji 192 KB (rys. 7.16), co oznacza stratę 64 KB pojemności, która mogłaby być użyta przez inny proces.

Bardziej efektywnym podejściem jest użycie partycji o *zmiennych rozmiarach*. Gdy proces jest wprowadzany do pamięci, jest umieszczany w obszarze o dokładnie takiej pojemności, jaka jest niezbędna. Przykład jest pokazany na rys. 7.17. Początkowo pamięć główna jest pusta, z wyjątkiem części zajętej przez system operacyjny. Ładowane są pierwsze trzy procesy, począwszy od miejsca zakończenia systemu operacyjnego (rys. 7.17a). Pozostaje luka na końcu pamięci, która jest za mała, aby zmieścić czwarty proces. Gdy proces 2 zostaje



RYSUNEK 7.16. Partycjonowanie na bloki o ustalonym rozmiarze



RYSUNEK 7.17. Wynik partycjonowania

usunięty (rys. 7.17b), jest już miejsce na proces 4. Ponieważ proces 4 jest mniejszy niż proces 2, powstaje następna mała luka. Jak pokazuje ten przykład, metoda ta rozpoczyna się bardzo dobrze, jednak może prowadzić do sytuacji, w której występuje w pamięci wiele małych luk. W miarę upływu czasu pamięć jest coraz bardziej dzielona na części i wykorzystanie pamięci jest coraz mniej efektywne. Metodą pokonywania tego problemu jest **upakowanie** (ang, *compaction*). Od czasu do czasu system operacyjny przesuwa procesy w pamięci w celu skumulowania wolnej pamięci w jednym bloku. Jest to procedura czasochłonna, powodująca stratę czasu procesora.

Zanim rozpatrzmy sposoby postępowania z wadami partycjonowania, musimy wyjaśnić jeszcze pewien problem. Jeśli czytelnik wróci na chwilę do rys. 7.17, to zauważy z pewnością, że najprawdopodobniej proces nie zostanie załadowany na to samo miejsce w pamięci głównej za każdym razem, gdy powraca w ramach wymiany. Proces w pamięci składa się z rozkazów i danych. Rozkazy zawierają adresy komórek pamięciowych dwóch rodzajów:

- a adresy jednostek danych,
- a adresy rozkazów używanych w przypadku rozgałęzienia.

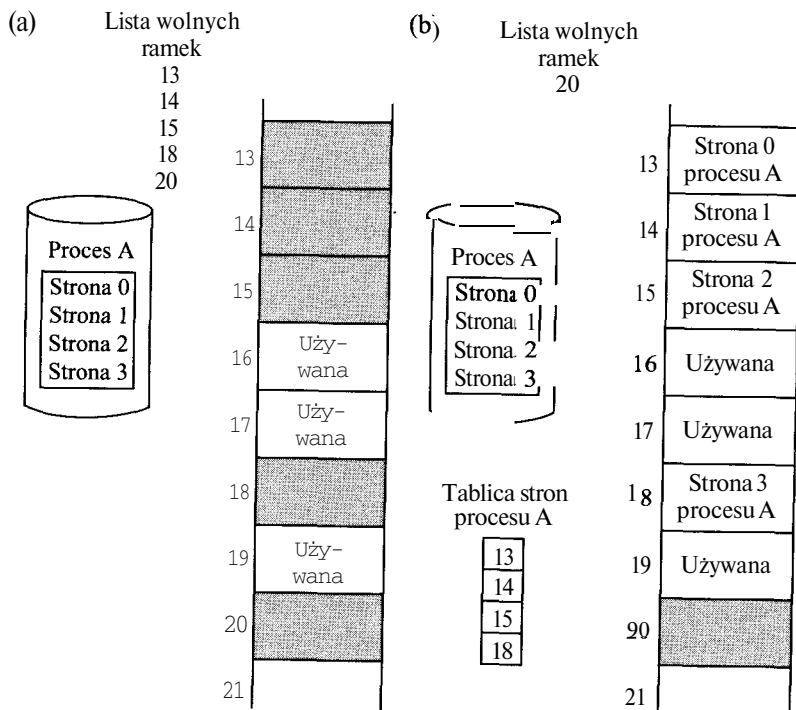
Zauważmy teraz, że te adresy nie są stałe! Zmieniają się one za każdym razem, gdy proces jest wprowadzany w ramach wymiany. W celu rozwiązania tego problemu, czynione jest rozróżnienie między adresem logicznym a fizycznym. **Adres logiczny** jest wyrażany jako położenie względem początku programu. **Adres fizyczny** jest oczywiście aktualnym położeniem w pamięci głównej.

Gdy procesor realizuje proces, dokonuje automatycznej konwersji adresów logicznych na fizyczne poprzez dodanie aktualnego położenia początku procesu, nazywanego jego **adresem bazowym**, do każdego adresu logicznego. Jest to jeszcze jeden przykład własności procesora zaprojektowanej w celu zaspokojenia wymagań systemu operacyjnego. Dokładna natura tej własności sprzętowej zależy od stosowanej strategii zarządzania pamięcią. W dalszej części tego podrozdziału znajdziemy kilka przykładów tej zależności.

Stronicowanie

Zarówno nierówne partycje o ustalonych rozmiarach, jak i partycje o zmiennych rozmiarach nie są wydajnym sposobem używania pamięci. Załóżmy jednak, że pamięć jest podzielona na małe fragmenty o ustalonym rozmiarze. Wtedy fragmenty programu, nazywane **stronami**, mogą być przypisane dostępnym fragmentom pamięci, nazywanym **ramkami** lub **ramkami stron**. W najgorszej sytuacji stracona pojemność pamięci w przypadku tego programu będzie tylko częścią ostatniej strony.

Na rysunku 7.18 jest pokazany przykład zastosowania stron i ramek. W określonym momencie niektóre ramki w pamięci są używane, inne zaś są wolne. Lista wolnych ramek jest utrzymywana przez system operacyjny. Proces A, przecho-

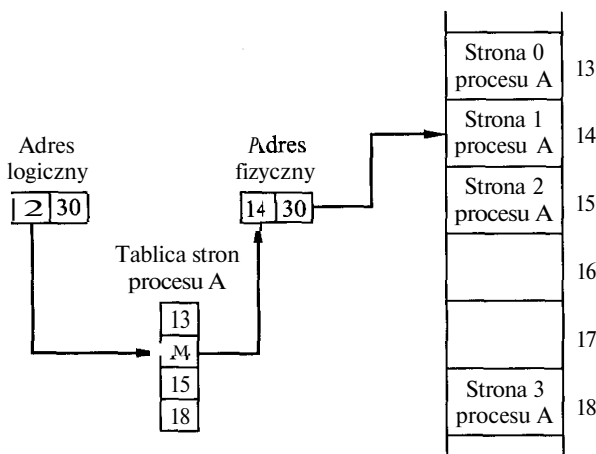


RYСУNEK 7.18. Przydział wolnych ramek: (a) przed przydziałem; (b) po

wywany na dysku, składa się z 4 stron. Gdy przychodzi czas ładowania tego procesu, system operacyjny odnajduje 4 wolne ramki i ładuje 4 strony procesu A do tych ramek.

Załóżmy teraz, że, jak w tym przykładzie, nie ma wystarczającej liczby nieużywanych, sąsiadujących ze sobą ramek, aby zmieścić ten proces. Czy to uniemożliwi systemowi operacyjnemu załadowanie procesu A? Odpowiedź brzmi - nie, ponieważ znów możemy użyć koncepcji adresu logicznego. Prosty adres bazowy nie będzie w tym przypadku odpowiedni. Zamiast tego system operacyjny tworzy **tablicę stron** dla każdego procesu. Tablica stron pokazuje położenie ramek każdej strony procesu. Wewnątrz programu każdy adres logiczny składa się z numeru strony i względnego adresu wewnątrz strony. Przypomnijmy, że w przypadku prostego partycyjnego wania adres logiczny jest położeniem słowa w stosunku do początku programu; procesor tłumaczy to na adres fizyczny. W przypadku stronicowania tłumaczenie adresów z logicznych na fizyczne jest nadal wykonywane przez sprzęt (procesor). Procesor musi teraz wiedzieć, jak uzyskać dostęp do tablicy stron aktualnego procesu. Otrzymując adres logiczny (numer strony, adres względny), procesor używa tablicy stron do utworzenia adresu fizycznego (numer ramki, adres względny). Przykład widać na rys. 7.19.

Podejście to stanowi rozwiązanie opisanych wcześniej problemów. Pamięć główna jest dzielona na wiele małych ramek o równych rozmiarach. Każdy proces jest dzielony na strony o rozmiarach ramek: mniejsze procesy wymagają mniejszej liczby ramek, a większe procesy - większej. Gdy proces jest wprowadzany, jego strony są ładowane do dostępnych ramek i ustalana jest tablica stron.



RYSunEK 7.19. Adresy logiczne i fizyczne

Pamięć wirtualna

Stronicowanie na żądanie

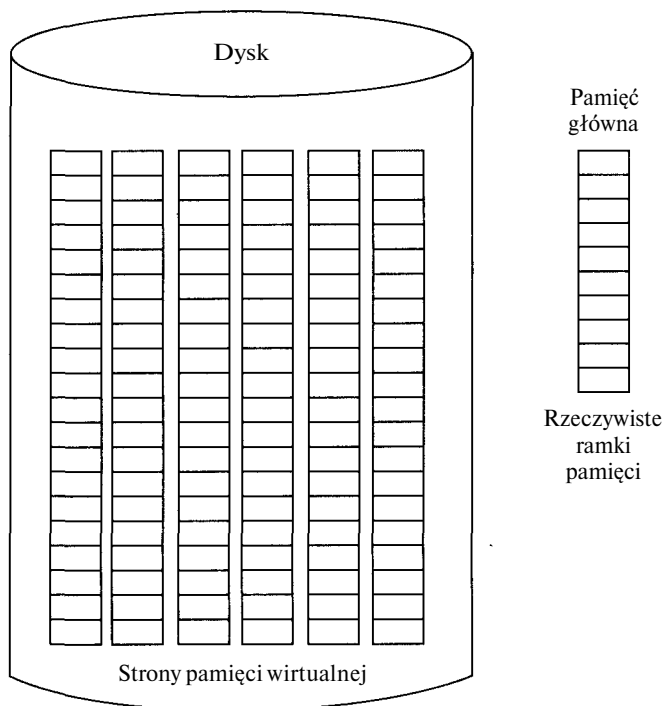
W wyniku zastosowania stronicowania stały się możliwe prawdziwie efektywne systemy wieloprogramowe. Równie ważne jest to, że prosta taktyka podziału procesu na strony doprowadziła do powstania innej ważnej koncepcji - pamięci wirtualnej.

Aby zrozumieć na czym polega pomysł pamięci wirtualnej, musimy dodać pewne ulepszenie do schematu stronicowania, który właśnie przedyskutowaliśmy. Ulepszeniem tym jest **stronicowanie na żądanie**, które oznacza po prostu, że każda strona procesu jest wprowadzana tylko wtedy, gdy jest potrzebna, to znaczy na żądanie.

Rozważmy duży proces, składający się z długiego programu oraz z pewnej liczby zespołów danych. Przez pewien czas wykonywanie programu może być ograniczone do małej jego części (np. do podprogramu standardowego) i mogą być używane tylko jeden lub dwa zespoły danych. Jest to zgodne z zasadą lokalności, którą wprowadziliśmy w dodatku 4A do rozdz. 4. Byłoby oczywiście rozrzutnością ładowanie wielu stron procesu, gdy tylko kilka stron będzie używanych przed zawieszeniem programu. Lepiej wykorzystamy pamięć, ładując tylko kilka stron. Następnie, jeśli program rozgałęzi się do rozkazu znajdującego się na stronie poza pamięcią lub jeśli program odwoła się do danych znajdujących się na stronie poza pamięcią, zostanie zasygnalizowany **błąd strony**. Skłoni to system operacyjny do dostarczenia żądanej strony.

Tak więc w określonym momencie tylko kilka stron danego procesu znajduje się w pamięci i dzięki temu więcej procesów może pozostawać w pamięci. Ponadto oszczędzany jest czas, ponieważ nieużywane strony nie są wprowadzane i wyprowadzane z pamięci. Jednak system operacyjny musi potrafić zarządzać tym schematem. Gdy wprowadza jedną stronę, musi wyrzucić inną. Jeśli wyrzuci stronę, która wkrótce ma być użyta, musi ją prawie natychmiast wprowadzić ponownie. Zbyt wiele takich sytuacji prowadzi do sytuacji zwanej **szamotaniem** (ang. *trashing*): procesor spędza większość czasu na wymienianiu stron, zamiast na wykonywaniu rozkazów. Zapobieganie szamotaniu było ważnym obszarem badań w latach siedemdziesiątych i doprowadziło do wielu złożonych, lecz efektywnych algorytmów. W istocie, system operacyjny próbuje zgadnąć, na podstawie najnowszej historii, które strony najprawdopodobniej będą użyte w bliskiej przyszłości.

W przypadku stronicowania na żądanie nie jest konieczne ładowanie całego procesu do pamięci głównej. Fakt ten ma doniosłe konsekwencje: *istnieje możliwość, żeby proces był większy od całej pamięci głównej*. Usunięte zostało jedno z najbardziej fundamentalnych ograniczeń w programowaniu. Bez stronicowania na żądanie programista musiał ciągle zwracać uwagę na to, ile pozostało dostępnej pamięci. Jeśli program został napisany jako zbyt obszerny, programista mu-



RYSUNEK 7.20. Pamięć wirtualna

siał znaleźć sposoby jego podziału na części, które mogły być jednorazowo załadowane. W przypadku stronicowania na żądanie zadanie to jest pozostawiane systemowi operacyjnemu i sprzętowi. Jeśli chodzi o programistę, ma on do czynienia z potężną pamięcią o pojemności związanej z pamięcią dyskową. System operacyjny używa stronicowania na żądanie w celu ładowania części tego procesu do pamięci głównej.

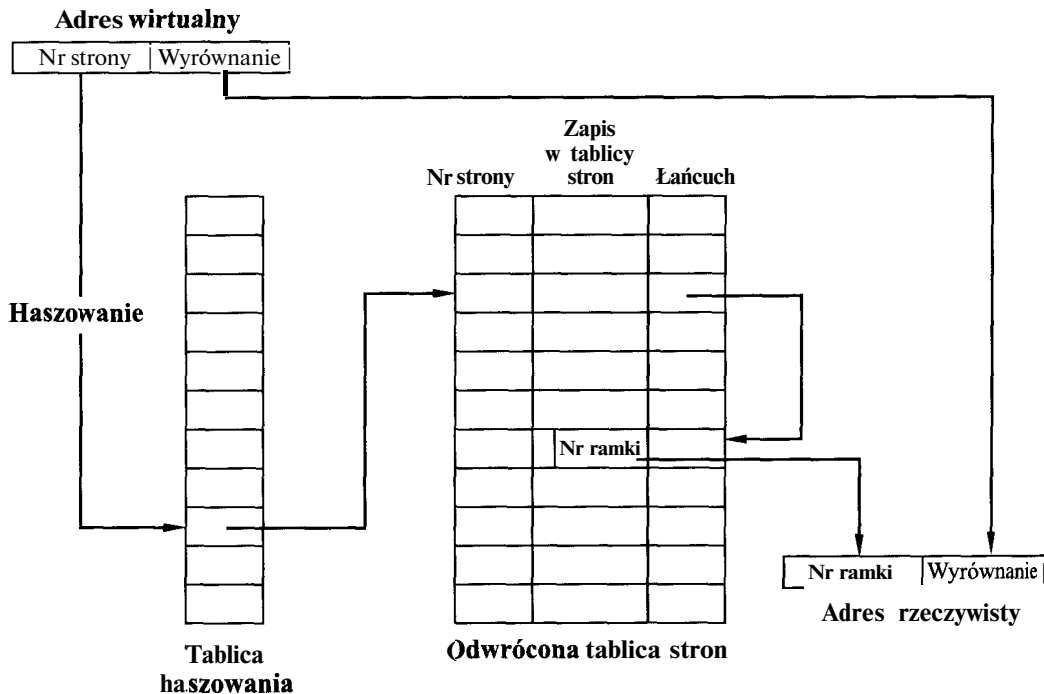
Ponieważ proces jest realizowany tylko w pamięci głównej, pamięć ta jest określana jako **pamięć rzeczywista**. Jednak programista lub użytkownik dysponuje o wiele większą pamięcią - tą, która znajduje się na dysku. Ta ostatnia jest wobec tego nazywana **pamięcią wirtualną** (rys. 7.20). Pamięć wirtualna umożliwia bardzo efektywne wieloprogramowanie i uwalnia użytkownika od niepożądanych ciasnych ograniczeń pamięci głównej.

Struktura tablicy stron

Podstawowy mechanizm odczytu słowa z pamięci obejmuje przetłumaczenie adresu wirtualnego (lub logicznego), składającego się z numeru strony i adresu względnego, na adres fizyczny, składający się z numeru ramki i adresu względnego, za pomocą tablicy stron. Ponieważ tablica stron ma zmienną długość, zależną od rozmiaru procesu, nie możemy spodziewać się trzymywania jej w reje-

strach. Zamiast tego musi się ona znajdować w pamięci głównej. Na rysunku 7.19 jest pokazana propozycja rozwiązania sprzętowego tego schematu. Gdy jest realizowany określony proces, rejestr zachowuje adres początkowy tablicy stron tego procesu. Numer strony adresu wirtualnego służy do indeksowania tej tablicy oraz do znalezienia odpowiedniego numeru ramki. Po połączeniu go z adresem względnym, stanowiącym część adresu wirtualnego, można otrzymać żądany adres rzeczywisty.

Rozważmy wymaganą liczbę zapisów tablic stron. W większości systemów występuje jedna tablica stron na proces. Jednak każdy proces może zajmować znaczną część pamięci wirtualnej. Na przykład, w przypadku architektury VAX każdy proces może zajmować do $2^{31} = 2$ GB pamięci wirtualnej. Przy użyciu stron $2^0 = 512$ -bajtowych, oznacza to, że na jeden proces może przypadać 2^{22} zapisów tablicy stron. Jest jasne, że pojemność pamięci przeznaczona na same tylko tablice stron byłaby niedopuszczalnie duża. Aby uporać się z tym problemem, w większości przypadków tablice stron są przechowywane raczej w pamięci wirtualnej niż w rzeczywistej. Oznacza to, że tablice stron również podlegają stronicowaniu. Gdy proces jest realizowany, przynajmniej część jego tablicy stron musi się znajdować w pamięci głównej, w tym zapis tablicy stron odnoszący się do aktualnie realizowanej strony. Niektóre procesory stosują dwupoziomowy schemat organizacji dużych tablic stron. W takim przypadku występuje katalog stron, w którym każdy zapis oznacza tablicę stron. Jeśli dłu-



RYSUNEK 7.21. Struktura odwróconej tablicy stron

gość katalogu stron wynosi X a maksymalna długość tablicy stron jest równa Y , to liczba stron procesu może wynosić do $X \times Y$. Zwykle maksymalna długość tablicy stron jest ograniczona do jednej strony. Zobaczmy przykład takiego dwu-poziomowego podejścia, gdy w dalszej części tego rozdziału będziemy rozważać Pentium.

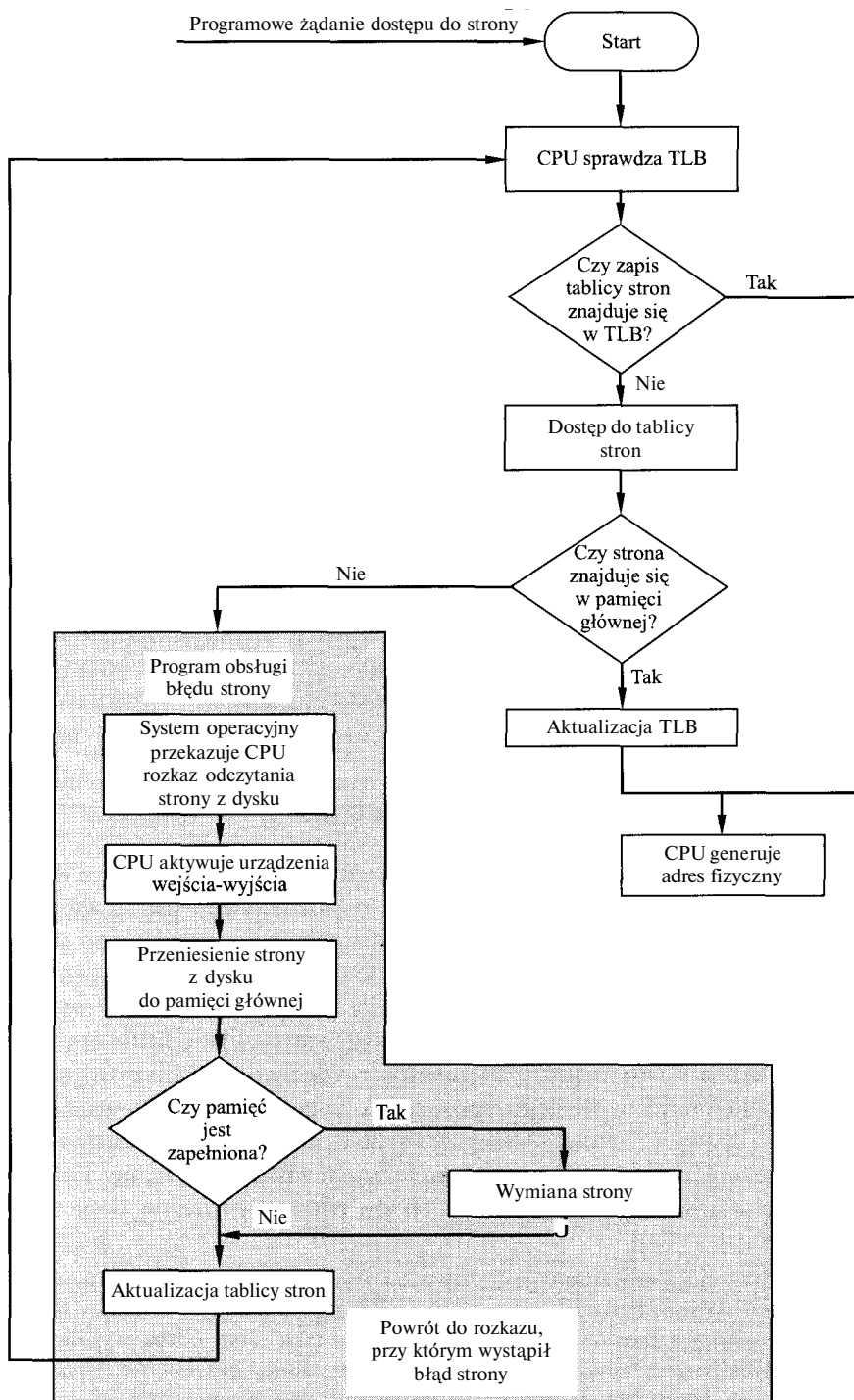
Alternatywnym rozwiązaniem w stosunku do stosowania jedno- i dwupoziomowych tablic stron jest użycie struktury **odwróconej tablicy stron** (rys. 7.21). Rozwiązanie to jest stosowane w komputerze AS/400 firmy IBM, a także we wszystkich wyrobach RISC tej firmy, łącznie z PowerPC.

W tym przypadku numer strony będący częścią adresu wirtualnego jest odwzorowywany w tablicy haszowania (ang. *hash table*) za pomocą prostej funkcji haszującej¹. Tablica haszowania zawiera wskaźnik odnoszący się do odwróconej tablicy stron, zawierającej zapisy tablicy stron. Przy użyciu tej struktury w tablicy haszowania i w odwróconej tablicy stron znajduje się jeden zapis dla każdej strony pamięci rzeczywistej, a nie dla każdej strony pamięci wirtualnej. Dzięki temu dla tablic jest wymagana ustalona część pamięci rzeczywistej, niezależnie od obsługiwanej liczby procesów lub stron wirtualnych. Ponieważ więcej niż jeden adres wirtualny może być odwzorowany w tym samym zapisie tablicy haszowania, wynikające stąd przepełnienie jest opanowywane za pomocą metody łańcuchowej. Metoda haszowania prowadzi do łańcuchów, które są zwykle krótkie i obejmują jeden lub dwa zapisy.

Bufor translacji adresów tablic stron

W zasadzie każde odniesienie do pamięci wirtualnej wywołuje dwa dostępy do pamięci fizycznej: jeden w celu pobrania odpowiedniego zapisu tablicy stron, a drugi w celu pobrania żądanych danych. Wobec tego przyjęcie prostego schematu pamięci wirtualnej powoduje podwojenie czasu dostępu do pamięci. Aby uporać się z tym problemem, w większości przypadków stosuje się specjalną pamięć podręczną zapisów tablicy stron, nazywaną zwykle **buforem translacji adresów tablic stron** (ang. *translation lookaside buffer* TLB). Ta pamięć podręczna funkcjonuje w ten sam sposób, jak pamięć podręczna współpracująca z pamięcią główną i zawiera te zapisy tablicy stron, które były ostatnio używane. Na rysunku 7.22 widać sieć działań, która ilustruje stosowanie TLB. Wobec zasady lokalności większość odniesień do pamięci wirtualnej dotyczy miejsc

¹ Funkcja haszująca odwzorowuje liczby z zakresu 0 do M na liczby w zakresie 0 do N , gdzie $M > N$. Wynik funkcji haszującej jest używany jako indeks w tablicy haszowania. Ponieważ więcej niż jedna liczba wejściowa daje taką samą liczbę wyjściową, liczba wejściowa może przy odwzorowywaniu natrafić w tablicy haszowania zapis, który jest już zajęty. W takim przypadku nowa liczba musi być przeniesiona na inne miejsce tablicy haszowania. Zwykle nowa liczba jest umieszczana w pierwszym najbliższym pustym obszarze, a do skojarzenia zapisów służy znacznik położenia oryginalnego. Bardziej szczegółowy opis tablic haszowania zawiera [STAL95].

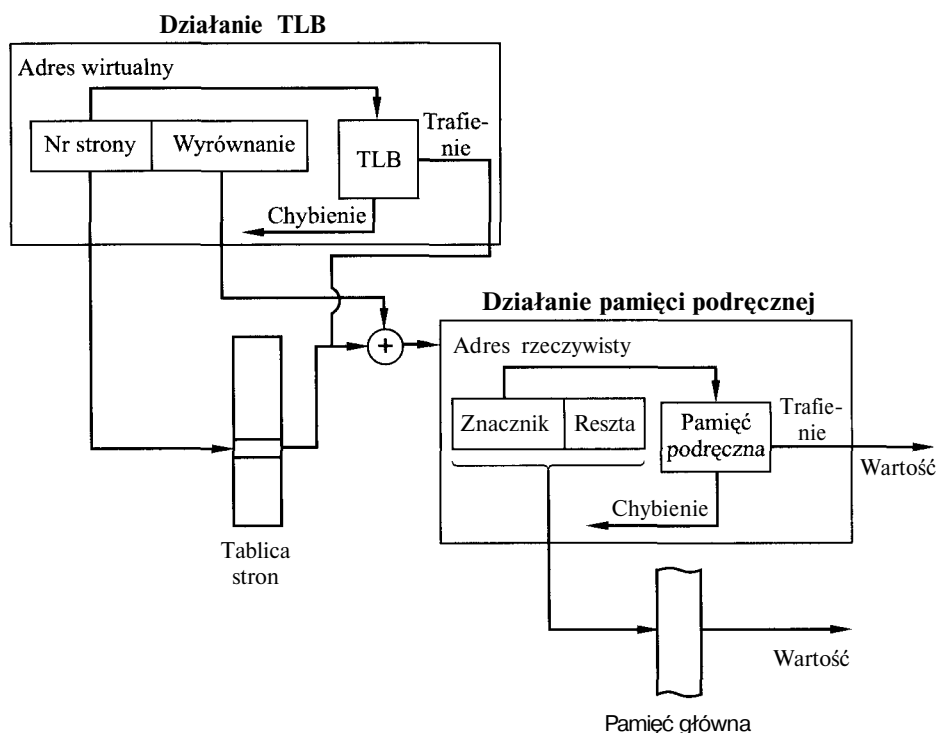


RYСУNEK 7.22. Operacja stronicowania i bufor translacji adresów tablic stron (TLB) [FURH87]

w ostatnio używanych stronach. Większość odniesień będzie więc dotyczyła zapisów tablic stron zawartych w pamięci podręcznej. Badania nad stosowaniem TLB w systemie VAX wykazały, że to rozwiązanie może znacznie poprawić wydajność [CLAR85, SATY81].

Zauważmy, że mechanizm pamięci wirtualnej musi współpracować z systemem pamięci podręcznej (nie z buforem TLB, ale z pamięcią podręczną pamięci głównej); widać to na rys. 7.23. Adres wirtualny ma na ogół postać numeru strony i adresu względnego. Po pierwsze, system pamięci sprawdza w TLB, czy jest w nim obecny odpowiedni zapis tablicy stron. Jeśli jest, to rzeczywisty (fizyczny) adres jest generowany przez połączenie numeru ramki z adresem względnym. Jeśli nie, to sięga się do zapisu w tablicy stron. Gdy jest już wygenerowany adres rzeczywisty, mający postać cechy i pozostałości (patrz rys. 4.16), następuje sprawdzenie w pamięci podręcznej, czy jest w niej obecny blok zawierający to słowo. Jeśli tak, to jest on zwracany do procesora. Jeśli nie, słowo jest pobierane z pamięci głównej.

Czytelnik z pewnością doceni złożoność sprzętową procesora wymaganą przy prostym odniesieniu do pamięci. Adres wirtualny jest tłumaczony na rzeczywisty. Obejmuje to odniesienie do tablicy stron, która może się znajdować w TLB, w pamięci głównej lub na dysku. Poszukiwane słowo może być w pa-



RYSUNEK 7.23. Działanie bufora translacji adresów tablic stron i pamięci podręcznej

mięci podręcznej, w pamięci głównej lub na dysku. W tym ostatnim przypadku strona zawierająca słowo musi być załadowana do pamięci głównej, a jej blok musi trafić do pamięci podręcznej. Ponadto zapis tablicy stron odnoszący się do tej strony musi być zaktualizowany.

Segmentacja

Istnieje jeszcze inny sposób dzielenia adresowalnej pamięci, znany jako segmentacja. Podczas gdy stronicowanie jest niewidzialne dla programisty i dostarcza mu większej przestrzeni adresowej, segmentacja jest zwykle widzialna dla programisty i jest stosowana w celu wygodniejszego organizowania programów i danych oraz jako środek kojarzenia atrybutów przywileju i ochrony z rozkazami i danymi.

Segmentacja pozwala programiście widzieć pamięć jako składającą się z wielu przestrzeni adresowych lub segmentów. Segmenty mają zmienny, dynamiczny rozmiar. Zwykle programista lub system operacyjny przypisują programy i dane do różnych segmentów. Może być wiele segmentów programów przeznaczonych dla różnych rodzajów programów i również wiele segmentów danych. Każdy segment może mieć przypisane prawa dostępu i użytkowania. Odniesienia do pamięci obejmują adres w formie numeru segmentu i adresu względnego.

Organizacja ta, w porównaniu z niesegmentowaną przestrzenią adresową, ma wiele zalet z punktu widzenia programisty:

1. Upraszcza ona operowanie rosnącymi strukturami danych. Jeśli programista nie wie z góry, jak duża stanie się określona struktura danych, nie musi tego zgadywać. Struktura danych może mieć przypisany własny segment, a system operacyjny będzie rozszerzał lub zmniejszał segment zależnie od potrzeby.
2. Umożliwia ona niezależne zmienianie i ponowne kompilowanie programów, bez wymagania, żeby cały zestaw programów był ponownie konsolidowany i ładowany. Jest to osiągnięte przez użycie wielu segmentów.
3. Możliwe jest wspólne wykorzystywanie danych przez różne procesy. Programista może umieścić program usługowy lub użyteczną tablicę danych w segmencie, który może być adresowany przez inne procesy.
4. Możliwa jest ochrona. Ponieważ segment może być utworzony jako zawierający ściśle określony zestaw programów lub danych, programista lub administrator systemu może wygodnie przypisać mu prawa dostępu.

Zalet tych nie ma pamięć stronicowana gdyż stronicowanie jest niewidzialne dla programisty. Z drugiej strony widzieliśmy, że stronicowanie jest efektywną formą zarządzania pamięcią. W celu połączenia zalet obu rozwiązań niektóre systemy są wyposażone w sprzęt i oprogramowanie systemowe, które umożliwiają stosowanie obu rozwiązań.

Zarządzanie pamięcią Pentium

Od czasu wprowadzenia architektury 32-bitowej zostały opracowane wyrafinowane schematy zarządzania pamięcią w systemach mikroprocesorowych, przy czym wykorzystano doświadczenia nagromadzone w odniesieniu do średnich i dużych systemów. W wielu przypadkach wersje mikroprocesorowe są lepsze od wielkosystemowych poprzedników. Ponieważ schematy te były opracowywane przez producentów sprzętu mikroprocesorowego i mogą współpracować z wieloma systemami operacyjnymi, są raczej schematami o ogólnym przeznaczeniu. Reprezentatywnym przykładem jest schemat użyty w Pentium. Sprzętowa strona zarządzania pamięcią Pentium jest zasadniczo taka sama, jak w przypadku procesorów 80386 i 80486, chociaż występują pewne ulepszenia.

Przestrzenie adresowe

Pentium zawiera rozwiązania sprzętowe zarówno wymagane do segmentowania, jak i do stronicowania. Oba mechanizmy są blokowane, co pozwala użytkownikowi wybrać jeden z czterech rodzajów organizacji pamięci:

- Q *Niesegmentowana pamięć niestronicowana.* W tym przypadku adresy wirtualne są takie same, jak fizyczne. Jest to użyteczne w przypadku zastosowań o małej złożoności i wysokiej wydajności, np. w sterownikach.
- *Niesegmentowana pamięć stronicowana.* Pamięć jest tu widziana jako stronicowana, liniowa przestrzeń adresowa. Ochrona i zarządzanie pamięcią odbywa się poprzez stronicowanie. Takie rozwiązanie jest faworyzowane przez niektóre systemy operacyjne, np. UNIX Berkeley.
- Q *Segmentowana pamięć niestronicowana.* W tym przypadku pamięć jest postrzegana jako zbiór przestrzeni adresów logicznych. Zaletą tego rozwiązania w porównaniu ze stronicowaniem jest to, że umożliwia ono ochronę aż do poziomu pojedynczego bajta, jeśli jest to konieczne. Ponadto, w przeciwieństwie do stronicowania, gwarantuje ono, że wymagana tablica translacji (tablica segmentu) znajduje się w mikroukładzie, gdy segment znajduje się w pamięci. Dzięki temu pamięć segmentowana niestronicowana ma przewidywalne czasy dostępu.
- Q *Segmentowana pamięć stronicowana.* Segmentowanie jest używane do definiowania logicznych partycji pamięci podlegających kontroli dostępu, a stronicowanie jest używane do zarządzania przydzielaniem pamięci wewnątrz partycji. Rozwiązanie to faworyzuje takie systemy operacyjne, jak UNIX System V.

Segmentacja

Gdy stosowana jest segmentacja, każdy adres wirtualny (nazywany w dokumentacji Pentium adresem logicznym) składa się z 16-bitowego odniesienia do segmentu i 32-bitowego adresu względnego. Dwa bity odniesienia do segmentu

służą mechanizmowi ochrony, pozostaje więc 14 bitów na określenie segmentu. Wobec tego w przypadku pamięci niesegmentowanej całkowita wirtualna przestrzeń adresowa widziana przez użytkownika wynosi $2^{32} = 4$ GB. W przypadku pamięci segmentowanej wynosi ona $2^{46} = 64$ TB (terabajtów). W przestrzeni adresów fizycznych są stosowane adresy 32-bitowe, co daje 4 GB.

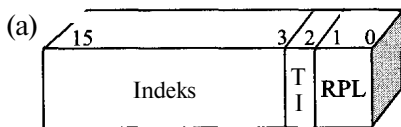
W rzeczywistości pojemność pamięci wirtualnej może być większa niż 64 TB. Jest tak, ponieważ interpretacja adresu wirtualnego przez procesor zależy od tego, który proces jest aktualnie czynny. Przestrzeń adresu wirtualnego jest dzielona na dwie części. Połowa przestrzeni adresu wirtualnego (8 K segmentów x 4 GB) jest globalna, dostępna dla wszystkich procesów; pozostała część ma charakter lokalny i jest odrębna dla każdego procesu.

Z każdym segmentem są związane dwie formy ochrony: poziom uprzywilejowania i atrybut dostępu. Występują cztery poziomy uprzywilejowania, od najbardziej chronionego (poziom 0) do najmniej chronionego (poziom 3). Poziom uprzywilejowania z segmentem danych jest jego „poziomem poufności” (ang. *classification*); poziom uprzywilejowania z segmentem programu jest jego „poziomem dostępności” (ang. *clearance*). Realizowany program może mieć dostęp tylko do takich segmentów danych, w stosunku do których jego poziom dostępności jest niższy (tzn. bardziej uprzywilejowany) lub równy (tzn. tak samo uprzywilejowany) poziomowi uprzywilejowania segmentu danych.

Sprzęt nie dyktuje sposobu używania tych poziomów uprzywilejowania; zależy to od projektu i implementacji systemu operacyjnego. Intencją projektantów było, aby poziom uprzywilejowania 1 był używany przez większą część kodu systemu operacyjnego, zaś poziom 0 - przez niewielką część systemu operacyjnego przeznaczoną do zarządzania, ochrony i kontroli dostępu do pamięci. Dwa poziomy pozostają aplikacjom. W wielu systemach programy użytkowe pozostają na poziomie 3, przy czym poziom 2 jest nieużywany. Wspecjalizowane podsystemy programów użytkowych, które muszą być chronione, ponieważ stosują własne mechanizmy zabezpieczeń, mogą kandydować do poziomu 2. Przykładami są systemy baz danych, systemy automatyzacji urzędów i środowiska inżynierii oprogramowania.

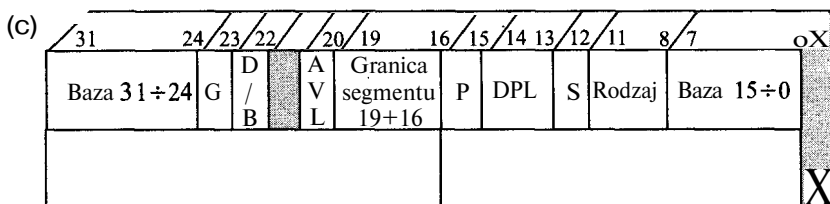
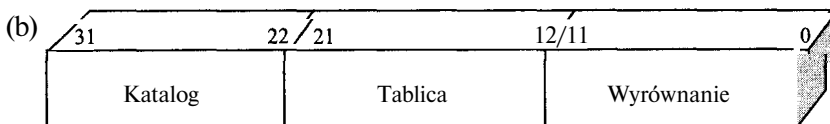
Oprócz kontrolowania dostępu do segmentów danych mechanizm uprzywilejowania ogranicza stosowanie pewnych rozkazów. Niektóre rozkazy, jak np. rozkazy operujące rejestrami zarządzania pamięcią, mogą być wykonywane wyłącznie na poziomie 0. Rozkazy wejścia-wyjścia mogą być wykonywane wyłącznie do pewnego poziomu określanego przez system operacyjny; zwykle jest to poziom 1.

Atrybut dostępu segmentu danych określa, czy dozwolone są operacje odczytu i zapisu, czy tylko odczytu. W przypadku segmentów programu atrybut dostępu określa, czy dozwolone jest odczytywanie i wykonywanie, czy tylko odczytywanie.



TI - wskaźnik tablicy

RPL - poziom uprzywilejowania



AVL - dostępne dla oprogramowania systemowego

Baza - adres segmentu bazowego

D/B - domyślny rozmiar operacji

DPL - rozmiar deskryptora uprzywilejowania

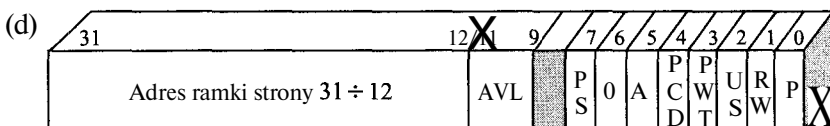
■ - zarezerwowany

G - stopień granulacji

P - obecność segmentu

Rodzaj - rodzaj segmentu

S - rodzaj deskryptora



AVL - dostępne dla programisty systemów

PS - rozmiar strony

A - potwierdzenie dostępu do strony

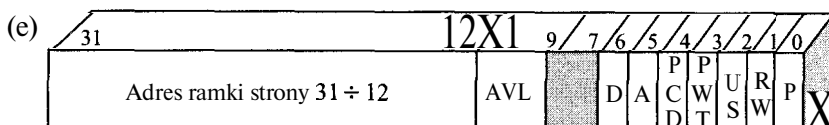
PCD - blokada pamięci podręcznej

PWT - **jednoczesny zapis**

US - **użytkownik/nadzorca**

RW - **odczyt/zapis**

P - obecna



D - potwierdzenie zapisu

RYSUNEK 7.24. Formaty zarządzania pamięcią procesora Pentium: (a) selektor segmentu; (b) adres liniowy; (c) deskryptor segmentu (zapis tablicy segmentów); (d) zapis katalogu stron; (e) zapis tablicy stron

W przypadku segmentowania mechanizm translacji adresu obejmuje odwzorowywanie adresu wirtualnego na tzw. adres liniowy (rys. 7.24b). Adres wirtualny składa się z 32-bitowego adresu względnego i z 16-bitowego selektora segmentu (rys. 7.24a). Selektor segmentu składa się z trzech pól. Są to:

- a Wskaźnik tablicy** (ang. *table indicator* - TI). Wskazuje, czy do translacji powinna być używana tablica segmentu ogólnego, czy tablica segmentu lokalnego.
- a Numer segmentu.** Określa numer segmentu służący jako indeks w tablicy segmentu.
- a Wymagany poziom uprzywilejowania** (ang. *requested privilege level* - RPL). Poziom uprzywilejowania wymagany dla tego dostępu.

Każdy zapis w tablicy segmentu obejmuje 64 bity, co widać na rys. 7.24c. Pola są zdefiniowane w tabeli 7.6.

Stronicowanie

Segmentowanie jest własnością opcjonalną i może być zablokowane. Gdy segmentowanie jest stosowane, adresy używane w programach są adresami wirtualnymi i są przekształcane na adresy liniowe w sposób uprzednio opisany. Gdy segmentowanie nie jest stosowane, w programach są używane adresy liniowe. W każdym przypadku potrzebne jest następujące działanie w celu przekształcenia adresu liniowego na rzeczywisty adres 32-bitowy.

Aby zrozumieć strukturę adresu liniowego, trzeba wiedzieć, że mechanizm stronicowania procesora Pentium jest w istocie dwupoziomową operacją przekształcania tablicowego. Pierwszy poziom stanowi katalog stron, zawierający do 1024 zapisów. Dzieli to 4-gigabajtową, liniową przestrzeń pamięci na 1024 grupy stron, z których każda ma własną tablicę stron i każda ma długość 4 MB. Każda tablica stron zawiera do 1024 zapisów; każdy zapis odpowiada pojedynczej stronie 4-kilobajtowej. Zarządzanie pamięcią dysponuje opcjami używania jednego katalogu stron dla wszystkich procesów, jednego katalogu stron dla każdego procesu lub pewnej kombinacji tych rozwiązań. Katalog stron odnoszący się do zadań bieżących znajduje się zawsze w pamięci głównej. Tablice stron mogą pozostawać w pamięci wirtualnej.

Na rysunku 7.24 są pokazane formaty zapisów w katalogach stron i w tablicach stron, natomiast pola są zdefiniowane w tabeli 7.6. Zauważmy, że mechanizmy kontroli dostępu mogą być wprowadzane przy użyciu stron lub grup stron.

W Pentium jest również wykorzystywany bufor translacji adresów tablic stron. Bufor może zawierać 32 zapisy tablic stron. Za każdym razem, gdy zmieniany jest katalog stron, bufor jest kasowany.

Na rysunku 7.25 jest pokazana kombinacja mechanizmów segmentowania i stronicowania. Dla jasności pominięto bufor translacji adresów tablic stron oraz pamięć podręczną.

TABELA 7.6. Parametry zarządzania pamięcią procesora Pentium

*Deskryptor segmentu (zapis tablicy segmentu)***Baza**

Określa adres początkowy segmentu wewnątrz 4-gigabajtowej liniowej przestrzeni adresowej.

Bit D/B

W segmencie programu jest to bit D, który wskazuje, czy argumenty i tryby adresowania są 16-, czy 32-bitowe.

Poziom uprzywilejowania deskryptora (DPL)

Określa poziom uprzywilejowania segmentu, do którego odnosi się ten deskryptor.

Bit granulacji (G)

Wskazuje, czy pole Granica ma być interpretowane w jednostkach 1 bajta, czy też 4 KB.

Granica

Definiuje rozmiar segmentu. Procesor interpretuje pole Granica na jeden z dwóch sposobów, zależnie od bitu granulacji: w jednostkach 1 bajta, aż do granicy rozmiaru segmentu równej 1 MB, lub w jednostkach 4 KB, aż do granicy 4 GB.

Bit S

Określa, czy dany segment jest segmentem systemowym, programów czy danych.

Bit obecności segmentu (P)

Używany w przypadku systemów niestronicowanych. Wskazuje, czy segment jest obecny w pamięci głównej. W przypadku systemów stronicowanych bit ten jest zawsze ustawiany na 1.

Rodzaj

Umożliwia rozróżnienie różnych rodzajów segmentów i wskazanie atrybutów dostępu.

*Zapisy katalogu stron i tablicy stron***Bit dostępu (A)**

Bit ten jest ustawiany na wartość 1 przez procesor na obu poziomach tablicy strony, gdy ma miejsce operacja odczytu lub zapisu w odpowiedniej stronie.

Bit zapisu (D)

Bit ten jest ustawiany na 1 przez procesor, gdy ma miejsce zapis na odpowiedniej stronie.

Adres ramki strony

Określa fizyczny adres strony w pamięci, jeśli jest ustawiony bit obecności. Ponieważ ramki stron są wyrównane z granicami 4 K, dolnymi 12 bitami są 0, a tylko górne 20 bitów włączono do zapisu. W katalogu stron jest to adres tablicy stron.

Bit blokady strony (PCD)

Wskazuje, czy dane zawarte na stronie mogą być umieszczane w pamięci podręcznej.

Bit rozmiaru strony (PS)

Wskazuje, czy rozmiar strony jest równy 4 KB, czy 4 MB.

Bit zapisu jednoczesnego strony (PWT)

Wskazuje, czy przy zapisywaniu danych na odpowiedniej stronie będzie stosowany zapis jednoczesny, czy też zapis opóźniony.

Bit obecności (P)

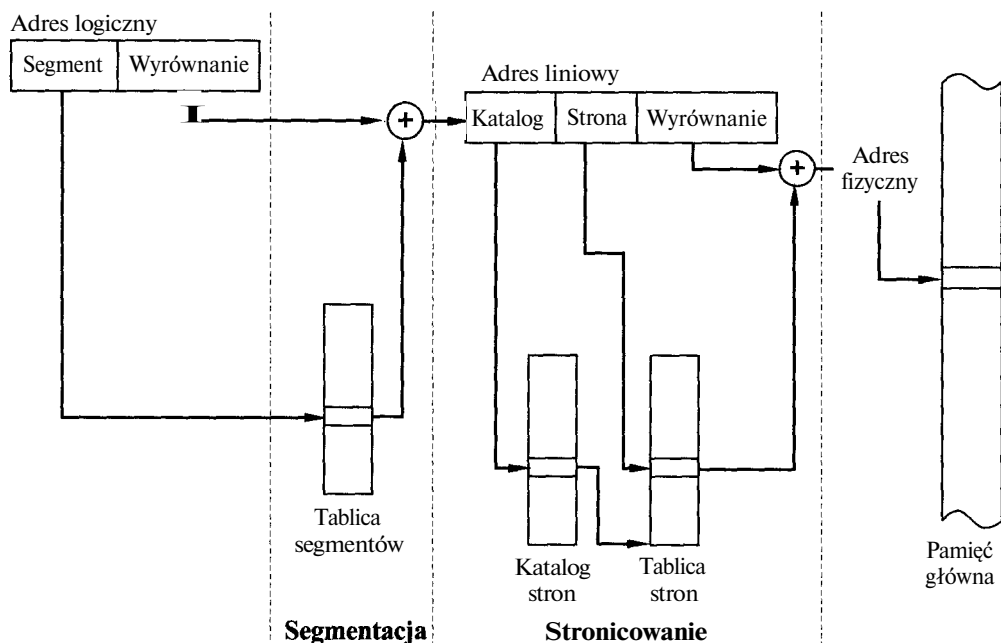
Wskazuje, czy tablica strony lub strona znajduje się w pamięci głównej.

Bit zapisu-odczytu (RW)

W przypadku stron z poziomu użytkownika wskazuje, czy strona może być tylko odczytywana, czy też również zapisywana przy wykonywaniu programów z poziomu użytkownika.

Bit użytkownik-nadzorca (US)

Wskazuje, czy strona jest dostępna tylko dla systemu operacyjnego (poziom nadzorcy), czy też jest również dostępna dla aplikacji (poziom użytkownika).



RYСУNEK 7.25. Mechanizmy translacji adresu w pamięci procesora Pentium

Pentium zawiera wreszcie nową możliwość, nie występującą w 80386 i 80486 - możliwość dwóch rozmiarów stron. Jeśli bit PSE (ang. *page size extension* - rozszerzenie rozmiaru strony) w rejestrze sterowania 4 jest ustawiony na 1, to jednostka stronicowania pozwala programiście systemu operacyjnego na określenie rozmiaru strony albo jako 4 KB, albo 4 MB.

Gdy stosowane są strony 4-megabajtowe, występuje tylko jeden poziom przekształcania tablicowego. Jeśli ma miejsce dostęp sprzętu do katalogu stron, to bit PS w katalogu stron (rys. 7.24d) jest ustawiany na 1. W tym przypadku bity od 9 do 21 są ignorowane, a bity od 22 do 31 określają adres bazowy 4-megabajtowej strony w pamięci. Występuje więc tylko jedna tablica stron.

Stosowanie stron 4-megabajtowych zmniejsza wymagania odnoszące się do zawartości związanej z zarządzaniem pamięcią w dużych pamięciach głównych. W przypadku stron 4-kilobajtowych, pełna pamięć główna o pojemności 4 GB wymaga około 4 MB na same tylko tablice stron. W przypadku stron 4-megabajtowych do *zarządzania* pamięcią wystarcza jedna tablica o długości 4 KB.

Zarządzanie pamięcią PowerPC

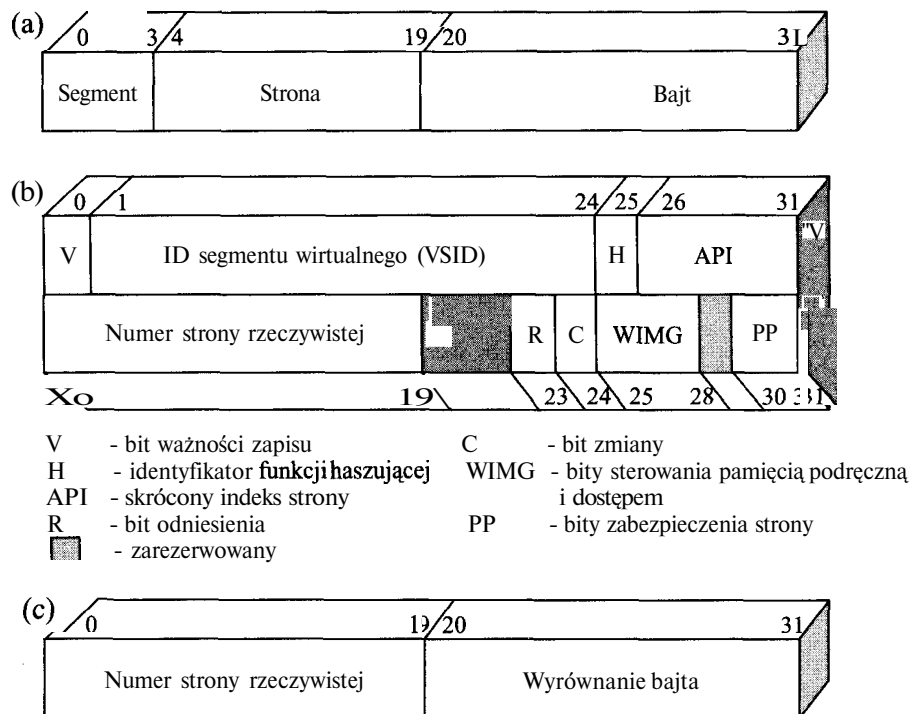
PowerPC umożliwia wykorzystanie obszernego zestawu mechanizmów adresowania. W przypadku 32-bitowej implementacji tej architektury stosowany jest mechanizm stronicowania z prostym segmentowaniem. W przypadku implementacji 64-bitowej używane jest stronicowanie oraz potężniejszy mechanizm segmentowania. Ponadto, zarówno w przypadku maszyn 32-bitowych, jak

i 64-bitowych, występuje alternatywny mechanizm sprzętowy znany jako **translacja adresu bloku**. W skrócie, schemat adresowania bloku zaprojektowano w celu wyeliminowania pewnej wady mechanizmów stronicowania. W przypadku stronicowania program może często odnosić się do dużej liczby stron. Na przykład, właściwość tę mogą wykazywać programy używające tablic OS (systemu operacyjnego) lub graficznych buforów ramki. W rezultacie często używane strony są stale pobierane i wyrzucane. Adresowanie blokowe umożliwia procesorowi rozplanowanie czterech dużych bloków pamięci rozkazów i czterech dużych bloków pamięci danych w sposób omijający mechanizm stronicowania.

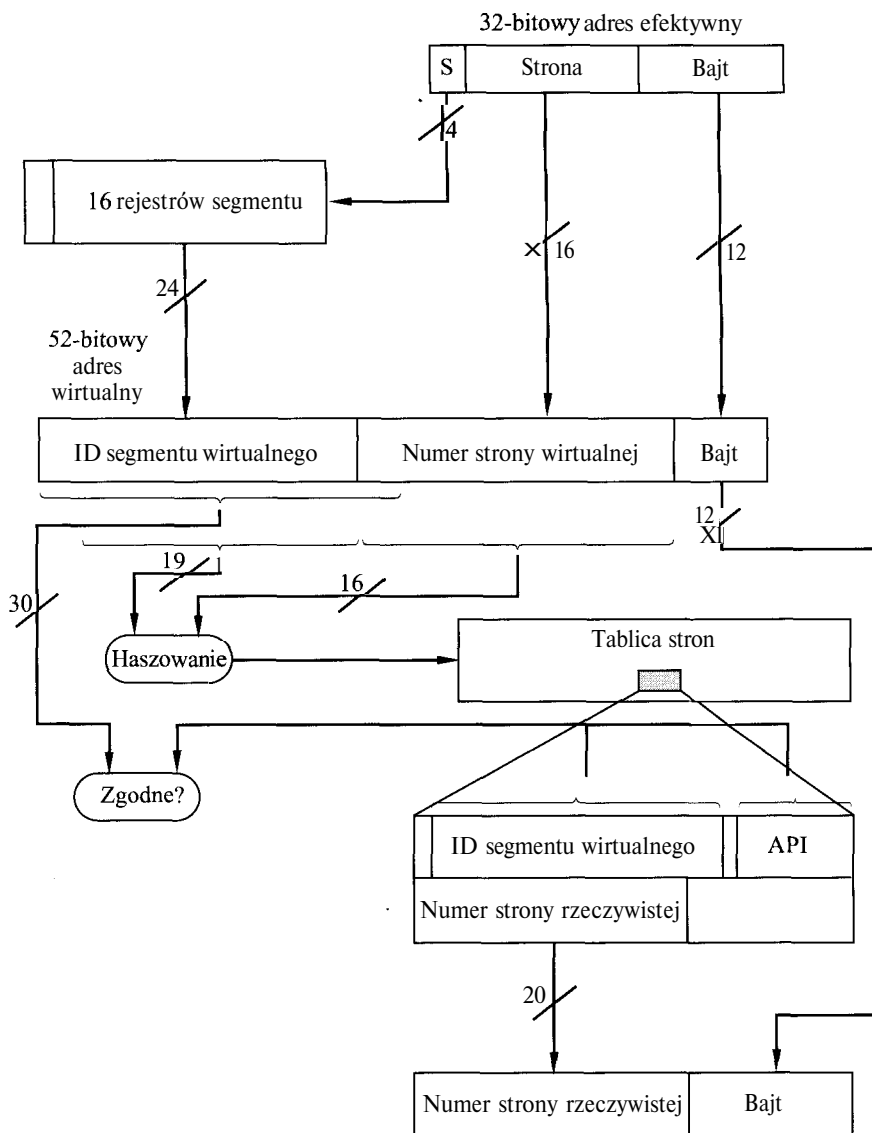
Dyskusja adresowania blokowego wykracza poza ramy tego rozdziału. W tej części podrzdziału skoncentrujemy się na mechanizmach stronicowania i segmentowania PowerPC.

Zarządzanie pamięcią 32-bitowego PowerPC

W 32-bitowym PowerPC jest używany 32-bitowy adres efektywny (rys. 7.26a). Adres ten zawiera 16-bitowy identyfikator strony i 12-bitowy selektor bajtu. Wobec tego możliwe jest używanie $2^{12} = 4$ KB stron. Dopuszcza się do $2^{16} = 64$ K stron na segment. Cztery bity adresu są używane do wyznaczania jednego z 16 rejestrów segmentu. Zawartość tych rejestrów jest kontrolowana przez



RYСУNEK 7.26. 32-bitowe formaty zarządzania pamięcią procesora PowerPC: (a) adres efektywny; (b) zapis tablicy stron; (c) adres rzeczywisty



RYSUNEK 7.27. Translacja 32-bitowego adresu procesora PowerPC

system operacyjny. Każdy rejestr segmentu zawiera bity kontroli dostępu oraz 24-bitowy identyfikator, dzięki czemu 32-bitowy adres efektywny jest odwzorowywany na 52-bitowy adres wirtualny (rys. 7.27).

W PowerPC zastosowano pojedynczą odwróconą tablicę stron. Adres wirtualny jest używany do ustalania indeksu w tablicy stron w następujący sposób. Po pierwsze, obliczany jest kod haszowania (ang. *hash code*):

$$H(0 \div 19) = \text{SID}(5 \div 23) \oplus \text{VPN}(0 \div 18)$$

Wirtualny numer strony w wirtualnym adresie jest wypełniany po stronie lewej (najbardziej znaczącej) trzema zerami binarnymi w celu utworzenia liczby 19-bitowej. Następnie, bit po bicie obliczane jest LUB wykluczające tej liczby oraz 19 najmniej znaczących bitów identyfikatora wirtualnego segmentu w celu utworzenia 19-bitowego kodu haszowania. Tablica jest organizowana jako n grup 8 zapisów. Od 10 do 19 bitów kodu haszowania (zależnie od rozmiaru tablicy stron) jest używanych do wybierania jednej z grup tablicy. Następnie sprzęt zarządzania pamięcią przegląda 8 zapisów grupy w celu sprawdzenia zgodności z adresem wirtualnym.

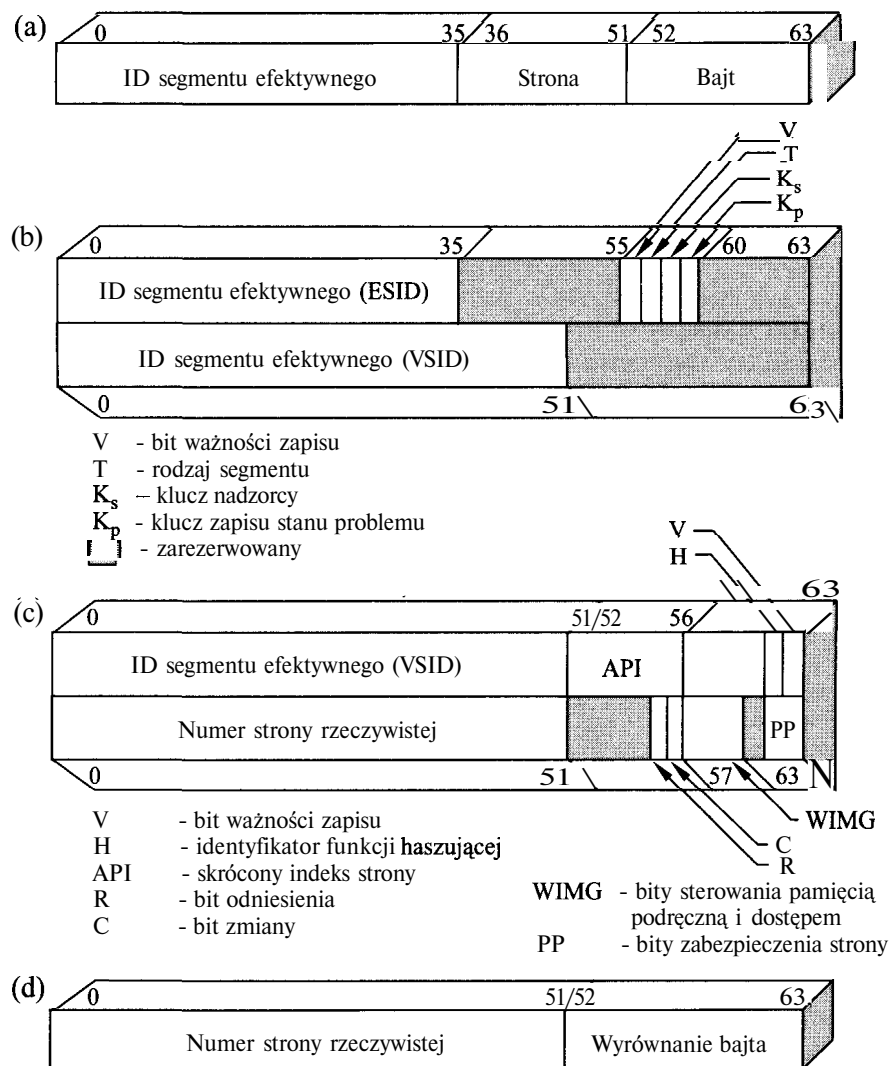
Aby umożliwić sprawdzanie zgodności, każdy zapis tablicy stron zawiera identyfikator segmentu wirtualnego i 6 najbardziej znaczących bitów wirtualnego numeru strony nazywanych skróconym indeksem strony (ponieważ przy najmniej 10 bitów z 16-bitowego wirtualnego numeru strony zawsze uczestniczy

TABELA 7.7. Parametry zarządzania pamięcią PowerPC

<i>Zapis tablicy segmentów</i>	
Identyfikator (ID) segmentu efektywnego	Wskazuje 1 z 64 G segmentów efektywnych; używany do określania zapisu w tablicy segmentu.
Bit ważności zapisu (V)	Wskazuje, czy w tym zapisie są ważne dane.
Bit rodzaju segmentu (T)	Wskazuje, czy jest to segment pamięci, czy wejścia-wyjścia.
Klucz nadzorcy (K_s)	Używany wraz z numerem wirtualnej strony w celu określenia zapisu w tablicy strony.
<i>Zapis tablicy strony</i>	
Bit ważności zapisu (V)	Wskazuje, czy w tym zapisie są ważne dane.
Identyfikator funkcji haszowania (H)	Wskazuje, czy jest to zapis pierwotny, czy wtórny z haszowaniem.
Skrócony wskaźnik strony (API)	Używany do jednoznacznego dopasowania adresu wirtualnego.
Bit odniesienia (R)	Ustawiany na wartość 1 przez procesor, gdy na odpowiedniej stronie ma miejsce operacja odczytu lub zapisu.
Bit zmiany (C)	Ustawiany na wartość 1 przez procesor, gdy na odpowiedniej stronie dokonywany jest zapis.
Bity WIMG	W = 0 - używany jest zapis opóźniony; W = 1 - używany jest zapis jednoczesny; I = 0 - przenoszenie do pamięci podręcznej nie jest zabronione; I = 1 - przenoszenie do pamięci podręcznej zabronione; M = 0 - pamięć nie jest wspólna; M = 1 - pamięć wspólna; G = 0 - pamięć nie chroniona; G = 1 - pamięć chroniona.
Bity zabezpieczenia strony (PP)	Bity kontroli dostępu używane wraz z bitami K zapisu tablicy segmentu lub rejestru segmentu w celu określenia praw dostępu.

w haszowaniu w celu wybrania grupy zapisów tablicy stron, tylko skrócona forma wirtualnego numeru strony musi być przeniesiona do zapisu tablicy strony, aby zapewnić jednoznaczność z adresem wirtualnym). Jeśli występuje zgodność, to 20-bitowy rzeczywisty numer strony z adresu jest wiązany z niższymi 12 bitami adresu efektywnego w celu utworzenia 32-bitowego adresu fizycznego, do którego wystąpiło odwołanie.

Jeśli zgodność nie występuje, to jest uzupełniany kod haszowania w celu utworzenia nowego indeksu tablicy stron, znajdującego się w takiej samej pozycji względnej na przeciwnym końcu tabeli. Grupa ta jest następnie przeglądana



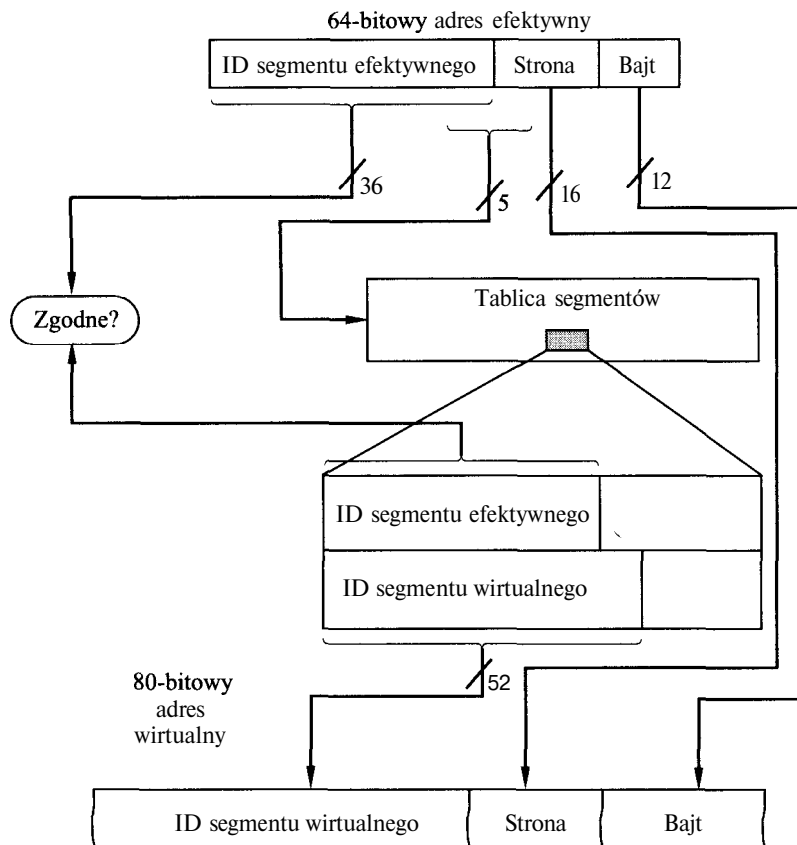
RYSUNEK 7.28. 64-bitowe formaty zarządzania pamięcią procesora PowerPC: (a) adres efektywny; (b) wpis w tablicy segmentów; (c) wpis w tablicy stron; (d) adres rzeczywisty

w celu zbadania zgodności. Jeśli zgodność nie zostanie stwierdzona, następuje przerwanie z powodu błędu strony.

Na rysunku 7.27 widać schemat logiczny mechanizmu translacji adresu, a na rys. 7.28 są pokazane formaty efektywnego adresu, zapisu tablicy stron i adresu rzeczywistego. W tabeli 7.7 są zdefiniowane parametry zapisu tablicy stron.

Zarządzanie pamięcią 64-bitowego PowerPC

Schemat zarządzania pamięcią odnoszący się do 64-bitowego PowerPC został tak zaprojektowany, aby był kompatybilny (jednokierunkowo) z implementacją 32-bitową. Wszystkie adresy efektywne, rejestry ogólne i rejestry adresu skoku zostały rozszerzone w lewo do 64 bitów. W celu dostosowania się do wymagania odwzorowywania pamięci proste rejestry segmentów zostały zastąpione przez strukturę tablicy haszowania segmentów (ang. *hashed segment table*), podobną do uprzednio opisanej struktury tablicy stron. W tej tablicy grupy zapisów zawierają po 8 zapisów segmentów tablic; każdy zapis zawiera 36-bitowy efek-



RYSUNEK 7.29. Translacja 64-bitowych adresów efektywnych na wirtualne w procesorze PowerPC

tywny identyfikator segmentu, 52-bitowy wirtualny identyfikator segmentu oraz takie same bity ochrony dostępu i selektora przestrzeni wejścia-wyjścia, jakie były zawarte w rejestrach segmentów procesora 32-bitowego (rys. 7.28).

Na rysunku 7.29 jest zilustrowane odwzorowywanie 64-bitowego adresu efektywnego na 80-bitowy adres wirtualny. Podobnie jak w przypadku rozwiązania 32-bitowego, adres efektywny zawiera 12-bitowy selektor bajta i 16-bitowy numer strony. Pozostałych 36 bitów tworzy efektywny identyfikator segmentu. Ponieważ nie jest wygodne posiadanie 2^{36} rejestrów segmentu, używana jest tablica haszowania segmentów.

Tablica segmentów składa się z 256 zapisów zorganizowanych w postaci 32 grup wpisów. Sprzęt *zarządzania* pamięcią wykorzystuje 5 najmniej znaczących bitów efektywnego identyfikatora segmentu w celu stworzenia indeksu, który wybiera w tablicy jedną z 32 grup. Jeśli przez porównanie identyfikatorów segmentów zostanie stwierdzona zgodność w grupie, to następnie jest tworzony 80-bitowy adres wirtualny. Składa się on z 52-bitowego wirtualnego identyfikatora segmentu, pochodzącego z zapisu tablicy segmentów, powiązanego z polami strony, i bajta adresu efektywnego. Jeżeli natomiast nie zostanie stwierdzona zgodność, powtórnie tworzy się indeks przez dopełnienie 5 bitów indeksu. Jeżeli podczas drugiego przeszukiwania nie stwierdza się zgodności, to następuje przerwanie związane z błędem strony.

Pozostała część procesu translacji adresu z wirtualnego na rzeczywisty przebiega według takiego schematu, jak w przypadku procesora 32-bitowego. W tym przypadku 16-bitowy wirtualny numer strony jest haszowany z niższymi 39 bitami wirtualnego numeru segmentu w celu wybrania grupy zapisów tablicy stron. Format zapisu tablicy stron (64-bitowy) ma 52-bitowy wirtualny identyfikator segmentu, 5-bitowy skrócony indeks strony oraz 52-bitowy rzeczywisty numer strony. Wynikiem odwzorowywania jest 64-bitowy adres rzeczywisty.

7.4

POLECANA LITERATURA

[STAL95] zawiera szczegółowe omówienie problemów będących przedmiotem tego rozdziału. Ponadto, podobnie jak w przypadku architektury komputerów, istnieje wiele książek na temat systemów operacyjnych. Prace [SILB94], [TAN92], [MILE92] i [DEIT90] zawierają opisy podstawowych zasad: do ich przedstawienia wykorzystano przykłady wielu ważnych systemów operacyjnych. W [SING94] i [NUTT92] potraktowano temat systemów operacyjnych na poziomie bardziej szczegółowym.

W [INTE94b] opisano architekturę zarządzania pamięcią Pentium. Dobre opisy architektury zarządzania pamięcią PowerPC można znaleźć w [DIEF94a] i [MAY94]

- DIEF94a Diefendorff K., Oehler R., Hochsprung R.: „Evolution of the PowerPC Architecture”. *IEEE Micro*, April 1994.
- DEIT90 Deitel H.: *An Introduction to Operating Systems*. Reading, Addison-Wesley, 1990.
- INTE94b Intel Corp.: *Pentium Family User's Manual. Vol.3: Architecture and Programming Manual*. Santa Clara, 1994.
- MAY94 May C., Silha E., Simoson R., Warren H. (redaktorzy): *The PowerPC Architecture*. San Francisco, Morgan Kaufmann, 1994.
- MILE92 Milenkovic M.: *Operating Systems: Concepts and Design*. New York, McGraw-Hill, 1992.
- NUTT92 Nutt G.: *Centralized and Distributed Operating Systems*. Englewood Cliffs, Prentice Hall, 1992.
- SILB94* Silberschatz A., Galvin P.: *Operating System Concepts*. Reading, Addison-Wesley, 1994.
- SING94 Singhal M., Shivaratri N.: *Advanced Concepts in Operating Systems*. New York, McGraw-Hill, 1994.
- STAL95 Stallings W.: *Operating Systems*. Second edition. Englewood Cliffs, Prentice Hall, 1995.
- TANE92 Tanenbaum A.: *Modern Operating Systems*. Englewood Cliffs, Prentice Hall, 1992.

75

PROBLEMY DO ROZWIĄZANIA

- 7.1. W pewnym systemie z podziałem czasu każdy użytkownik spędza średnio 20 s na myśleniu i pisanu między aktami konwersacji. Większość tych aktów wymaga 50 ms czasu procesora, jednak 1 na 20 wymaga 2 s. Ponadto każdy akt konwersacji obejmuje czasy ładowania i wymiatania wynoszące po 100 ms każdy.

Opisz metody szeregowania i wymiany, które powinny być przyjęte w tym systemie, żeby uzyskać najkrótszy możliwy czas odpowiedzi przy maksymalnej liczbie użytkowników.

Wskaż, jakiej pojemności pamięci będzie wymagał ten system (mierzonej typowymi rozmiarami zadań). Wyprowadź wzór na czas odpowiedzi R dla krótkich aktów konwersacji w zależności od liczby użytkowników N . Jakie przyjmiesz założenia i jakie towarzyszą temu niedokładności?

- 7.2. Załóżmy, że dysponujemy wieloprogramowym komputerem, w którym każde zadanie ma identyczne własności. W jednym okresie obliczeń T połowa czasu jest zużywana przez wejście-wyjście, a połowa przez procesor. Każde zadanie jest aktywne łącznie przez N okresów. Załóżmy, że stosowany jest prosty, cykliczny schemat priorytetu. Zdefiniujmy następujące wielkości:

- a długość cyklu przetwarzania = rzeczywisty czas zakończenia zadania,

* W roku 2000 ukazało się nakładem WNT tłumaczenie piątego wydania amerykańskiego pt. *Podstawy systemów operacyjnych* (przyp. red.).

- a przepustowość = średnia liczba zadań kończonych w okresie T ,
- a wykorzystanie procesora = procent czasu, w którym procesor jest aktywny (nie czeka).

Oblicz te wielkości dla jednego, dwóch i czterech jednoczesnych zadań, zakładając, że okres T jest rozłożony następująco:

- (a) wejście-wyjście pierwsza połowa, procesor druga;
 - (b) wejście-wyjście pierwsza i czwarta ćwiartka, procesor druga i trzecia.
- 7.3. Zdefiniujmy czas oczekiwania jako czas, który zadanie spędza, czekając w kolejce krótkookresowej (rys. 7.14). Podaj równanie, które wiąże długość cyklu przetwarzania, czas aktywnego procesora i czas oczekiwania.
- 7.4. Program zorientowany na wejście-wyjście jest to program, który (jeśli byłby realizowany sam) spędziłby więcej czasu czekając na wejście-wyjście, niż używając procesora. Program zorientowany na procesor jest przeciwieństwem poprzedniego. Załóżmy, że algorytm szeregowania krótkookresowego faworyzuje te programy, które ostatnio zużywały niewiele czasu procesora. Wyjaśnij, dlaczego ten algorytm faworyzuje programy zorientowane na wejście-wyjście, a mimo to nie blokuje permanentnie programów zorientowanych na procesor.
- 7.5. Program oblicza sumy szeregu

$$C_i = \sum_{j=1}^n a_{ij}$$

dla tablicy A o wymiarze 100 na 100. Załóżmy, że komputer stosuje stronicowanie na żądanie z rozmiarem strony 1000 słów i że pojemność pamięci głównej związana z danymi wynosi 5 ramek. Czy wystąpiłaby różnica częstości błędów stron, jeśli A byłaby przechowywana w pamięci wirtualnej w postaci wierszy i kolumn? Wyjaśnij.

- 7.6. System pamięci wirtualnej ma rozmiar strony 1024 słowa, 8 wirtualnych stron i 4 fizyczne ramki. Tablica strony jest następująca:

Wirtualny numer strony	Numer ramki
0	3
1	1
2	—
3	—
4	2
5	—
6	0
7	—

- (a) Sporządź listę wszystkich wirtualnych adresów, które mogą spowodować błędy stron.
- (b) Jakie są adresy pamięci głównej dla następujących adresów wirtualnych: 0, 3728, 1023, 1024, 1025, 7800, 4096?

- 7.7. Podaj przyczyny, dla których rozmiar strony w systemie pamięci wirtualnej nie powinien być ani bardzo mały, ani bardzo duży.
- 7.8. Następująca sekwencja wirtualnych numerów stron wystąpiła podczas pracy komputera z pamięcią wirtualną:

3 4 2 6 4 7 1 3 2 6 3 5 1 2 3

Założmy, że stosowany jest algorytm wymiany stron, do których dawno nie było odwołania. Sporządź wykres współczynnika trafień stron (udziału odniesień do stron, przy których strona została znaleziona w pamięci głównej) w funkcji pojemności pamięci głównej mierzonej w stronach n dla $1 < n < 8$. Załóż, że pamięć główna jest początkowo pusta.

- 7.9. W systemie VAX tablice stron użytkownika są lokowane pod adresami wirtualnymi w przestrzeni systemowej. Na czym polega zaleta umieszczania tablic stron użytkownika w pamięci wirtualnej zamiast w głównej?
- 7.10. Rozważmy system komputerowy dysponujący zarówno pamięcią wirtualną, jak i podręczną. W przypadku obiektu istniejącego w pamięci głównej (i być może w pamięci podręcznej) Wymagane są dwa kroki:
1. Translacja adresu wirtualnego na adres pamięci głównej.
 2. Sprawdzenie, czy dane zawarte w pamięci głównej znajdują się także w pamięci podręcznej.

W celu zaoszczędzenia czasu chcielibyśmy, aby te dwie funkcje się nakładały, to znaczy, aby przynajmniej częściowo były realizowane równolegle. W systemie Amdahl 470 translacja adresu zawsze pozostawia nienaruszonych 11 najmniej znaczących bitów adresu (względny adres bajta). Pamięć podręczna jest N -drożna sekcyjno-skojarzeniowa. Jeśli niektóre z tych 11 bitów są używane do wybierania jednej z 64 kolumn pamięci podręcznej, które spośród niżej wymienionych kroków dotyczących pamięci podręcznej mogą się nakładać z translacją bitów wysokiego rzędu adresu wirtualnego?

1. Wybór kolumny.
 2. Odczyt N wskaźników adresów pamięci podręcznej w wybranej kolumnie.
 3. Porównanie N wskaźników adresów z adresami rzeczywistymi.
 4. Wybór jednej z N linii danych.
- 7.11. Rozważmy system komputerowy, w którym stosowane są segmentowanie i stronicowanie. Gdy segment znajduje się w pamięci, niektóre słowa na ostatniej stronie są tracone. Ponadto, przy rozmiarze segmentu s i strony p występuje s/p zapisów tablicy stron. Im mniejszy jest rozmiar strony, tym mniejsza jest strata na ostatniej stronie segmentu, jednak większa jest tablica stron. Jaki rozmiar strony minimalizuje łączne straty?
- 7.12. Komputer ma pamięć podręczną, pamięć główną i dyskową wykorzystywaną do pamięci wirtualnej. Jeśli słowo znajduje się w pamięci podręcznej, to wymaganych jest A ns w celu uzyskania dostępu do niego. Jeśli jest

ono w pamięci głównej, ale nie w podręcznej, to najpierw potrzeba B ns na jego załadowania do pamięci podręcznej, po czym znów rozpoczyna się operacja odniesienia. Gdy słowo nie znajduje się w pamięci głównej, wymaganych jest C ns na pobranie go z dysku, po czym potrzeba B ns na umieszczenie go w pamięci podręcznej. Jeśli współczynnik trafień pamięci podręcznej wynosi $(n - 1)/n$, a współczynnik trafień pamięci głównej wynosi $(m - 1)/m$, to jaki jest średni czas dostępu?

- 7.13. Załóżmy, że zadanie jest dzielone na 4 równe segmenty i że system tworzy 8-zapisową tablicę deskryptorów stron dla każdego segmentu. System stanowi więc kombinację segmentowania i stronicowania. Załóżmy także, że rozmiar strony wynosi 2 KB.
- Jaki jest maksymalny rozmiar każdego segmentu?
 - Jaka jest maksymalna przestrzeń adresów logicznych dla zadania?
 - Załóżmy, że w zadaniu tym następuje dostęp do elementu zawartego w komórce fizycznej 00021 ABC. Jaki jest format adresu logicznego, który jest w tym celu generowany przez zadanie? Jaka jest maksymalna przestrzeń adresu fizycznego tego systemu?

Źródło: [ALEX93].

- 7.14. Załóżmy, że pewien mikroprocesor może uzyskiwać dostęp do 2^{32} bajtów fizycznej pamięci głównej. Wykorzystuje on jedną, segmentowaną przestrzeń adresów logicznych o maksymalnym rozmiarze 2^{31} bajtów. Każdy rozkaz zawiera pełny, dwuczęściowy adres. Stosowane są zewnętrzne jednostki zarządzania pamięcią (MMU), których schemat zarządzania przypisuje segmentom kolejne bloki pamięci fizycznej o ustalonym rozmiarze 2^{22} bajtów. Początkowy adres fizyczny segmentu jest zawsze podzielny przez 2048. Pokaż szczegółowe powiązania zewnętrznego mechanizmu odwzorowania, który dokonuje konwersji adresów logicznych na fizyczne, stosując odpowiednią liczbę jednostek MMU. Pokaż szczegółową strukturę wewnętrzną MMU (zakładając, że każda MMU zawiera 128-zapisową pamięć podręczną bezpośrednio odwzorowanych deskryptorów segmentów) oraz jak jest wybierana każda MMU.

Źródło: [ALEX93].

- 7.15. Rozważmy stronicowaną przestrzeń adresów logicznych (złożoną z 32 stron po 2 KB każda) odwzorowaną na przestrzeń 1-megabajtowej pamięci fizycznej.
- Jaki jest format adresu logicznego procesora?
 - Jaka jest długość i szerokość tablicy stron (zaniedbując bity „praw dostępu”)?
 - Jaki wpływ na tablicę stron miałyby zmniejszenie przestrzeni pamięci fizycznej o połowę?

Źródło: [ALEX93].

JEDNOSTKA CENTRALNA

Aż do tego miejsca rozpatrywaliśmy procesor w zasadzie jako „czarną skrzynkę” i rozważaliśmy jego oddziaływanie z wejściem-wyjściem oraz z pamięcią. W części III zajmiemy się strukturą wewnętrzną i działaniem procesora.

W rozdziale 8 omówimy funkcjonowanie ALU, skupiając się na reprezentacji liczb i metodach wdrażania operacji arytmetycznych. Złożony problem projektowania list rozkazów będzie poruszony w rozdziałach 9 i 10. W rozdziale 11 opiszemy użycie rejestrów jako pamięci wewnętrznej procesora, a następnie podsumujemy dotychczas przedstawiony materiał w celu dokonania przeglądu struktury i działania procesora. Resztę części III poświęcimy bardziej szczegółowemu omówieniu głównych tendencji w projektowaniu procesorów. W rozdziale 12 opiszemy rozwiązanie wykorzystujące koncepcję komputera o zredukowanej liście rozkazów (RISC), a w rozdz. 13 rozpatrzymy stosowanie metod superskalarnych, które to rozwiązanie jest używane w wielu najnowszych projektach procesorów.

Arytmetyka komputera

Rozpocniemy analizowanie procesora od jednostki arytmetyczno-logicznej (ALU). Po krótkim wprowadzeniu skupimy się na najbardziej złożonym aspekcie ALU, a mianowicie na arytmetyce komputera. Funkcje logiczne, będące częścią ALU są opisane w rozdz. 9, a wdrożenie prostych funkcji logicznych i arytmetycznych w postaci cyfrowych układów logicznych - w dodatku do książki.

Arytmetyka komputera jest zwykle realizowana za pomocą dwóch bardzo różnych rodzajów liczb: całkowitych i zmiennopozycyjnych. W obu przypadkach wybór reprezentacji jest kluczowym problemem projektowym i zajmiemy się nim na początku, po czym omówimy operacje arytmetyczne.

Przegląd systemów liczbowych znajduje się w dodatku do tego rozdziału.

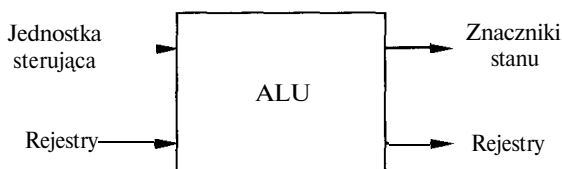
8.1

JEDNOSTKA ARYTMETYCZNO-LOGICZNA (ALU)

Jednostka arytmetyczno-logiczna (ALU) jest tą częścią komputera, która realizuje operacje arytmetyczne i logiczne na danych. Wszystkie inne elementy systemu komputerowego – jednostka sterująca, rejestry, pamięć, wejście-wyjście – istnieją głównie po to, żeby dostarczać dane do ALU w celu ich przetworzenia, a następnie odbierać wyniki. Rozpatrując ALU, sięgamy w pewnym sensie do rdzenia komputera.

Jednostka arytmetyczno-logiczna, a w rzeczywistości wszystkie podzespoły elektroniczne komputera, wykorzystują proste, cyfrowe układy logiczne, które mogą przechowywać cyfry binarne i wykonywać proste operacje logiczne Boole'a. W dodatku do tej książki znajduje się opis realizacji cyfrowych układów logicznych.

Na rysunku 8.1 jest pokazany, w bardzo ogólnym ujęciu, sposób połączenia ALU z resztą procesora. Dane są przedstawiane ALU w rejestrach i wyniki operacji są przechowywane w rejestrach. Rejestry te są miejscami tymczasowego przechowywania danych wewnątrz procesora i są połączone ścieżkami sygnało-



RYSUNEK 8.1. Wejścia i wyjścia ALU

wymi z ALU (patrz np. rys. 2.3). ALU ustawia również **znaczniki** (ang. *flags*) będące wynikiem operacji. Na przykład, znacznik przepełnienia jest ustawiany jako 1, jeżeli wynik obliczeń przekracza długość rejestru, w którym ma być przechowywany. Wartości znaczników są również przechowywane w rejestrach wewnątrz procesora. Jednostka sterująca dostarcza sygnały, które sterują działaniem ALU oraz ruchem danych do (i z) ALU.

8.2

REPREZENTACJA LICZB CAŁKOWITYCH

W systemie binarnym (patrz dodatek 8A) dowolne liczby mogą być reprezentowane po prostu za pomocą cyfr 0 i 1, znaku minus oraz przecinka. Na przykład:

$$-1101,0101_2 = -11,3125_{10}$$

Dla celów przechowywania i przetwarzania w komputerze nie dysponujemy jednakże znakami minus i przecinka. Tylko cyfry binarne (0 i 1) mogą służyć do reprezentowania liczb. Gdybyśmy używali wyłącznie nieujemnych liczb całkowitych, reprezentacja byłaby prosta. Słowo 8-bitowe mogłoby reprezentować liczby od 0 do 255. Na przykład:

$$00000000 = 0$$

$$00000001 = 1$$

$$00101001 = 41$$

$$10000000 = 128$$

$$11111111 = 255$$

Ogólnie, jeśli n -bitowy ciąg cyfr binarnych $a_{n-1} a_{n-2} \dots a_0$ jest interpretowany jako pozbawiona znaku liczba całkowita A , to jej wartość oblicza się ze wzoru:

$$A = \sum_{i=0}^{n-1} 2^i a_i$$

Reprezentacja znak-moduł

Używanie liczb całkowitych bez znaku jest niewystarczające w wielu przypadkach; na przykład wtedy, kiedy wymagane jest reprezentowanie również liczb ujemnych. Istnieje kilka konwencji, które można zastosować. Wszystkie z nich polegają na traktowaniu najbardziej znaczącego (położonego po lewej stronie)

bitu słowa jako bitu znaku: jeśli ten bit jest równy 0, to liczba jest dodatnia, a jeśli 1 – ujemna.

Najprostszą formą reprezentacji, w której jest używany bit znaku, jest reprezentacja znak-moduł. W słowie n -bitowym $n - 1$ bitów znajdujących się po prawej stronie określa moduł liczby całkowitej. Na przykład:

$$+18 = 00010010$$

$$-18 = 10010010 \quad (\text{znak-moduł})$$

Ogólny przypadek można wyrazić następująco:

$$A = \begin{cases} \sum_{i=0}^{n-2} 2^i a_i & \text{jeśli } a_{n-1} = 0 \\ -\sum_{i=0}^{n-2} 2^i a_i & \text{jeśli } a_{n-1} = 1 \end{cases}$$

Reprezentacja (notacja) znak-moduł ma kilka wad. Jedna polega na tym, że dodawanie i odejmowanie wymagają rozważania zarówno znaków liczb, jak i ich modułów w celu przeprowadzenia wymaganej operacji. Stanie się to jasne po zapoznaniu się z podrozdz. 8.3. Inną wadą jest występowanie dwóch reprezentacji liczby 0:

$$+0_{10} = 00000000$$

$$-0_{10} = 10000000 \quad (\text{znak-moduł})$$

Jest to niewygodne, ponieważ nieco trudniej jest sprawdzać 0 (operacja często przeprowadzana przez komputery), niż gdyby występowała jedna reprezentacja.

Reprezentacja uzupełnienia do dwóch¹

Reprezentacja uzupełnienia do dwóch została opracowana w celu wyeliminowania wymienionych już dwóch głównych niedogodności reprezentacji znak-moduł: nieefektywnego dodawania i odejmowania oraz dwóch reprezentacji zera.

Podobnie jak w notacji znak-moduł, w reprezentacji uzupełnienia do dwóch używa się najbardziej znaczącego bitu jako bitu znaku, co ułatwia sprawdzanie, czy liczba całkowita jest dodatnia, czy ujemna. Natomiast różni się ona od reprezentacji znak-moduł sposobem interpretowania pozostałych bitów.

Reprezentację uzupełnienia do dwóch można najlepiej zrozumieć, definiując ją za pomocą ważonej sumy bitów, jak to czyniliśmy powyżej w przypadku re-

¹ Przedstawienie reprezentacji uzupełnienia do dwóch w tym podrozdziale oraz w podrozdz. 8.3 opiera się na podejściu zasugerowanym przez G. Dattatreya z University of Texas (patrz [DATT93]).

prezentacji bezznakowej i znak-moduł. Rozważmy n -bitową liczbę całkowitą A wyrażoną za pomocą uzupełnienia do dwóch. Jeśli A jest dodatnia, to bit znaku a_{n-1} ma wartość 0. Pozostałe bity reprezentują moduł liczby w ten sam sposób, jak w przypadku notacji znak-moduł:

$$A = \sum_{i=0}^{n-2} 2^i a_i \quad \text{dla } A > 0$$

Liczba 0 jest określana jako dodatnia; wobec tego ma bit znaku równy 0 i moduł złożony z samych zer. Jak widać, zakres dodatnich liczb całkowitych, które mogą być reprezentowane, sięga od 0 (wszystkie bity modułu są 0) do $2^{n-1} - 1$ (wszystkie bity modułu są 1). Jakakolwiek większa liczba wymagałaby więcej bitów.

W przypadku ujemnej liczby A bit znaku a_{n-1} jest równy 1. Pozostałe $n - 1$ bitów może przybierać dowolną z 2^{n-1} wartości. Wobec tego zakres ujemnych liczb całkowitych, które mogą być reprezentowane, sięga od -1 do -2^{n-1} . Okazuje się, że wygodne jest takie przypisanie wartości, w którym bity $a_{n-1} a_{n-2} \dots a_0$ są równe liczbie dodatniej $2^{n-1} + A$, co wyraża się następująco:

$$2^{n-1} + A = \sum_{i=0}^{n-2} 2^i a_i \quad \text{tak więc} \quad A = -2^{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Ponieważ bit znaku jest równy 1, możemy napisać wzór na liczbę ujemną następująco:

$$A = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i \quad (8-1)$$

W przypadku dodatnich liczb całkowitych $a_{n-1} = 0$, więc $-2^{n-1} a_{n-1} = 0$. Wobec tego równanie (8-1) definiuje reprezentację uzupełnienia do dwóch zarówno liczb dodatnich, jak i ujemnych.

W tabeli 8.1 znajduje się porównanie reprezentacji znak-moduł oraz uzupełnienia do dwóch w odniesieniu do 4-bitowych liczb całkowitych. Chociaż uzupełnienie do dwóch stanowi kłopotliwą reprezentację z punktu widzenia człowieka, zobaczymy, że ułatwia ona implementację najważniejszych operacji arytmetycznych, dodawania i odejmowania. Z tego powodu jest ono niemal uniwersalnie używane do reprezentowania liczb całkowitych w procesorach.

Użyteczną ilustracją natury reprezentacji uzupełnienia do dwóch jest tablica wartości, w której wartość znajdująca się przy prawej krawędzi tablicy wynosi 1 (2^0), a każda kolejna wartość położona na lewo jest podwojona, aż do wartości położonej najbardziej na lewo, która jest zanegowana. Jak można zauważyć na rys. 8.2a, największa ujemna liczba, która może być reprezentowana w notacji uzupełnienia do dwóch, jest równa -2^{n-1} . Jeśli którykolwiek z bitów poza bitem znaku jest równy 1, dodaje on do liczby pewną dodatnią wartość. Jest również jasne, że liczba ujemna musi mieć 1 po lewej stronie, a liczba dodatnia musi

TABELA 8.1. Porównanie reprezentacji znak-moduł oraz uzupełnienia do dwóch dla 4-bitowych liczb całkowitych

Reprezentacja dziesiętna	Reprezentacja znak-moduł	Reprezentacja uzupełnienia do dwóch
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
+0	0000	0000
-0	1000	-
-1	1001	1111
-2	1010	1110
-3	1011	1101
-4	1100	1100
-5	1101	1011
-6	1110	1010
-7	1111	1001
-8	-	1000

(a)

-128	64	32	16	8	4	2	1

(b)

-128	64	32	16	8	4	2	1
1	0	0	0	0	0	1	1

-128+2 +1 = -125

(c)

-128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	0

-120 = -128+8

RYSUNEK 8.2. Zastosowanie tablicy wartości do konwersji liczb reprezentacji uzupełnienia do dwóch na reprezentację dziesiętną; (a) ośmiopozycyjna tablica wartości uzupełnienia do dwóch; (b) konwersja liczby binarnej 10000011 na dziesiętną; (c) konwersja liczby dziesiętnej -120 na binarną

mieć na tej pozycji 0. Największa liczba dodatnia jest więc reprezentowana przez ciąg zaczynający się od 0, po którym następują same 1; jest ona równa $2^{n-1} - 1$.

Pozostała część rysunku 8.2 ilustruje zastosowanie tablicy wartości do konwersji między zapisem dziesiętnym a notacją uzupełnienia do dwóch.

Konwersja między różnymi długościami bitowymi

Jest czasem pożądane przechowanie liczby n -bitowej w postaci m bitów, gdzie $m > n$. W przypadku notacji znak-moduł można tego dokonać z łatwością: po prostu trzeba przesunąć bit znaku do najdalszej lewej pozycji oraz wypełnić pozostałe wolne pozycje zerami. Na przykład:

$$\begin{aligned} +18 &= && 00010010 && (\text{znak-moduł, 8 bitów}) \\ +18 &= && 0000000000010010 && (\text{znak-moduł, 16 bitów}) \\ -18 &= && 10010010 && (\text{znak-moduł, 8 bitów}) \\ -18 &= && 1000000000010010 && (\text{znak-moduł, 16 bitów}) \end{aligned}$$

Procedura taka nie działa w przypadku całkowitych liczb ujemnych w reprezentacji uzupełnienia do dwóch. Postępując jak w powyższym przykładzie, otrzymamy:

$$\begin{aligned} +18 &= && 00010010 && (\text{uzupełnienie do dwóch, 8 bitów}) \\ +18 &= && 0000000000010010 && (\text{uzupełnienie do dwóch, 16 bitów}) \\ -18 &= && 11101110 && (\text{uzupełnienie do dwóch, 8 bitów}) \\ -32\ 658 &= && 1000000001101110 && (\text{uzupełnienie do dwóch, 16 bitów}) \end{aligned}$$

Zamiast tego, w przypadku uzupełnienia do dwóch stosuje się następującą regułę: wykonuje się przesunięcie bitu znaku do najdalszej lewej pozycji, po czym wypełnia się powstałe puste pozycje kopiami bitu znaku. W przypadku liczb dodatnich wypełnia się zerami, w przypadku ujemnych – jedynkami. Mamy więc:

$$\begin{aligned} -18 &= && 11101110 && (\text{uzupełnienie do dwóch, 8 bitów}) \\ -18 &= && 1111111111101110 && (\text{uzupełnienie do dwóch, 16 bitów}) \end{aligned}$$

Żeby zobaczyć, dlaczego ta reguła funkcjonuje, ponownie rozważmy n -bitowy ciąg cyfr binarnych $a_{n-1} a_{n-2} \dots a_1 a_0$ interpretowany jako liczba całkowita A w notacji uzupełnienia do dwóch. Jej wartość wynosi:

$$A = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Jeśli A jest liczbą dodatnią, reguła oczywiście się sprawdza. Załóżmy teraz, że A jest ujemna i że chcemy zbudować m -bitową reprezentację przy $m > n$. Wtedy

$$A = -2^{m-1} a_{m-1} + \sum_{i=0}^{m-2} 2^i a_i$$

Te dwie wartości muszą być równe:

$$-2^{m-1} + \sum_{i=0}^{n-2} 2^i a_i = -2^{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

$$-2^{m-1} + \sum_{i=n-1}^{m-2} 2^i a_i = -2^{n-1}$$

$$2^{n-1} + \sum_{i=n-1}^{m-2} 2^i a_i = 2^{m-1}$$

$$1 + \sum_{i=0}^{n-2} 2^i + \sum_{i=n-1}^{m-2} 2^i a_i = 1 + \sum_{i=0}^{m-2} 2^i$$

$$\sum_{i=n-1}^{m-2} 2^i a_i = \sum_{i=n-1}^{m-2} 2^i$$

Przechodząc od pierwszego do drugiego równania, wymagamy, żeby $n - 1$ najmniej znaczących bitów nie zmieniało się przy przejściu od jednej reprezentacji do drugiej. Następnie przechodzimy do ostatniego równania, które jest prawdziwe tylko wtedy, gdy wszystkie bity na pozycjach od $n - 1$ do $m - 2$ są równe 1. Reguła więc działa.

Reprezentacja stałopozycyjna

Zwróćmy uwagę na to, że reprezentacje analizowane w tym punkcie są czasem określane jako stałopozycyjne. Jest tak, ponieważ położenie przecinka pozycyjnego (przecinka binarnego) jest ustalone i zakładamy, że znajduje się on na prawo od cytry położonej najdalej na prawo. Programista może używać tej samej reprezentacji dla ułamków binarnych, posługując się skalowaniem liczb, tak że przecinek binarny przez implikację znajduje się w pewnym innym położeniu.

8.3

ARYTMETYKA LICZB CAŁKOWITYCH

W tym podrozdziale zajmiemy się zwykłymi funkcjami arytmetycznymi na liczbach w reprezentacji uzupełnienia do dwóch.

Negowanie

W przypadku reprezentacji znak-moduł reguła negowania liczby całkowitej jest prosta: odwrócić bit znaku. W notacji uzupełnienia do dwóch negowanie liczby całkowitej może być dokonywane za pomocą następujących reguł:

1. Weź uzupełnienie Boole'a każdego bitu liczby całkowitej (łącznie z bitem znaku).
2. Traktując wynik jako liczbę całkowitą pozbawioną znaku, dodaj 1.

Oto przykład:

$$\begin{array}{rcl}
 & 18 = 00010010 & \text{(uzupełnienie do dwóch)} \\
 \text{uzupełnienie bitowe} & = 11101101 & \\
 & + \underline{\hspace{1cm}} 1 & \\
 & 11101110 & = -18
 \end{array}$$

Jak można było się spodziewać, negacją negacji liczby jest ona sama:

$$\begin{array}{rcl}
 & -18 = 11101110 & \text{(uzupełnienie do dwóch)} \\
 \text{uzupełnienie bitowe} & = 00010001 & \\
 & + \underline{\hspace{1cm}} 1 & \\
 & 00010010 & = 18
 \end{array}$$

Możemy zademonstrować słuszność opisanej powyżej metody, używając definicji reprezentacji uzupełnienia do dwóch w postaci równania (8-1). Ponownie zinterpretujemy n -bitowy ciąg cyfr binarnych $a_{n-1} a_{n-2} \dots a_1 a_0$ jako liczbę całkowitą A w reprezentacji uzupełnienia do dwóch. Jej wartość wynosi więc

$$A = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Utwórzmy teraz uzupełnienie na poziomie bitowym $\overline{a_{n-1}} \overline{a_{n-2}} \dots \overline{a_0}$ i traktując je jako pozbawioną znaku liczbę całkowitą, dodajmy 1. Na zakończenie zinterpretujemy powstały ciąg cyfr binarnych jako liczbę całkowitą B przedstawioną w notacji uzupełnienia do dwóch. Jej wartość jest następująca:

$$B = -2^{n-1} \overline{a_{n-1}} + 1 + \sum_{i=0}^{n-2} 2^i \overline{a_i}$$

Chcemy, żeby $A - -B$, co oznacza, że $A + B = 0$. Z łatwością możemy to wykazać:

$$A + B = -(a_{n-1} + \overline{a_{n-1}})2^{n-1} + 1 + \sum_{i=0}^{n-2} 2^i (a_i + \overline{a_i}) = -2^{n-1} + 1 + \sum_{i=0}^{n-2} 2^i = -2^{n-1} + 2^{n-1} = 0$$

W powyższych obliczeniach założyliśmy, że możemy najpierw potraktować bitowe uzupełnienie A jako pozbawioną znaku liczbę całkowitą, do której dodajemy 1, po czym przyjmujemy, że wynik jest liczbą całkowitą w reprezentacji uzupełnienia do dwóch. Do rozważenia pozostają dwa przypadki szczególne. Najpierw weźmy $A = 0$. W tym przypadku przy reprezentacji 8-bitowej

$$\begin{array}{rcl}
 & 0 = 00000000 & \text{(uzupełnienie do dwóch)} \\
 \text{uzupełnienie bitowe} & = 11111111 & \\
 & + \underline{\hspace{1cm}} 1 & \\
 & 100000000 & = 0
 \end{array}$$

Występuje tu przepełnienie (nadmiar), które jest ignorowane. Wynikiem negacji zera jest zero, tak jak powinno być.

Drugi przypadek szczególny stanowi większy problem. Jeśli zanegujemy wzór bitowy ciągu składającego się z 1 z następującymi $n - 1$ zerami, to otrzymamy tę samą liczbę. Na przykład:

$$\begin{array}{r} -128 = 10000000 \quad (\text{uzupełnienie do dwóch}) \\ \text{uzupełnienie bitowe} = 01111111 \\ \quad \quad \quad + \quad \quad \quad 1 \\ \hline 10000000 = -128 \end{array}$$

Pewna anomalia tego rodzaju jest nie do uniknięcia. Liczba możliwych wzorów bitowych w słowie n -bitowym wynosi 2^n i jest to liczba parzysta. Chcemy reprezentować dodatnie i ujemne liczby całkowite oraz zero. Jeśli jednakowe ilości liczb dodatnich i ujemnych są reprezentowane przez znak i moduł, to występują dwie reprezentacje zera. Jeśli natomiast jest tylko jedna reprezentacja zera (uzupełnienie do dwóch), to musi występować nierówna liczba reprezentowanych liczb dodatnich i ujemnych. W przypadku uzupełniania do dwóch istnieje n -bitowa reprezentacja dla -2^n , ale nie dla 2^n .

Dodawanie w notacji uzupełniania do dwóch jest pokazane na rys. 8.3. Pierwsze cztery przykłady ilustrują operacje udane. Jeśli wynik operacji jest dodatni, to otrzymujemy dodatnią liczbę w zwykłej notacji binarnej. Jeśli wynik operacji jest ujemny, to otrzymujemy liczbę ujemną w reprezentacji uzupełnienia do dwóch. Zauważmy, że w pewnych przypadkach poza końcem słowa występuje bit przeniesienia. Jest on ignorowany.

Przy dodawaniu wynik może być większy, niż można zmieścić w ramach przyjętej długości słowa. Warunek ten jest nazywany **przepełnieniem**. Gdy występuje przepełnienie, ALU musi sygnalizować ten fakt, nie są więc podejmowane próby wykorzystania wyniku. W celu wykrywania przepełnienia stosowana jest następująca reguła:

(a) $\begin{array}{r} 1001 \\ \underline{0101} \\ 1110 = -2 \end{array}$	(b) $\begin{array}{r} 1100 \\ \underline{0100} \\ 0000 \\ 0000 = 0 \end{array}$
(c) $\begin{array}{r} 0011 \\ \underline{0100} \\ 0111 = 7 \end{array}$	(d) $\begin{array}{r} 1100 \\ \underline{1111} \\ \textcircled{1} 1011 \\ 1011 = -5 \end{array}$
(e) $\begin{array}{r} 0101 \\ \underline{0100} \\ 1001 = \text{Przepełnienie} \end{array}$	(f) $\begin{array}{r} 1001 \\ \underline{1010} \\ 0001 = \text{Przepełnienie} \end{array}$

RYSUNEK 8.3. Dodawanie liczb w reprezentacji uzupełnienia do dwóch: (a) $(-7) + (+5)$; (b) $(-4) + (+4)$; (c) $(+3) + (+4)$; (d) $(-4) + (-1)$; (e) $(+5) + (+4)$; (f) $(-7) + (+6)$

Reguła przepełnienia

Jeśli są dodawane dwie liczby dodatnie lub dwie ujemne, to przepełnienie następuje wtedy i tylko wtedy, gdy wynik ma znak przeciwny.

Na rysunku 8.3e i f są pokazane przykłady przepełnienia. Zauważmy, że przepełnienie może występować niezależnie od tego, czy istnieje przeniesienie. Odejmowanie jest również łatwe; wykonuje się je, wykorzystując następującą regułę:

Reguła odejmowania

W celu odjęcia od jednej liczby (odjemnej) drugiej liczby (odjemnika) weź uzupełnienie do dwóch odjemnika i dodaj do odjemnej.

Odejmowanie jest więc realizowane poprzez dodawanie, co widać na rys. 8.4. Ostatnie dwa przykłady pokazują, że reguła przepełnienia nadal ma zastosowanie.

Na rysunku 8.5 są pokazane ścieżki danych i elementy sprzętowe wymagane do dodawania i odejmowania. Centralnym elementem jest sumator binarny, do którego są doprowadzane dwie liczby przeznaczone do dodania. Na jego wyjściu pojawia się ich suma oraz wskazanie przepełnienia. Sumator binarny traktuje obie liczby jako pozbawione znaku liczby całkowite. (Implementacja sumatora jest przedstawiona w dodatku do książki). W celu dodania dwie liczby są doprowadzane do sumatora z dwóch rejestrów, oznaczonych w tym przypadku jako A i B. Wynik częściej jest przechowywany w jednym z tych rejestrów niż

$$(a) M = 2 = 0010$$

$$S = 7 = 0111$$

$$S' = 1001$$

$$0010$$

$$+1001$$

$$1011 = -5$$

$$(b) M = 5 = 0101$$

$$S = 2 = 0010$$

$$S' = 1110$$

$$0101$$

$$+1110$$

$$\textcircled{R} \quad 0011$$

$$0011 = 3$$

$$(c) M = -5 = 1011$$

$$S = 2 = 0010$$

$$S' = 1110$$

$$1011$$

$$+1110$$

$$(D) \quad 1001$$

$$1001 = -7$$

$$(d) M = 5 = 0101$$

$$S = -2 = 1110$$

$$S' = 0010$$

$$0101$$

$$+0010$$

$$0111 = 7$$

$$(e) M = 7 = 0111$$

$$S = -7 = 1001$$

$$S' = 0111$$

$$0111$$

$$+0111$$

$$1110 = \text{Przepełnienie}$$

$$(f) M = -6 = 1010$$

$$S = 4 = 0100$$

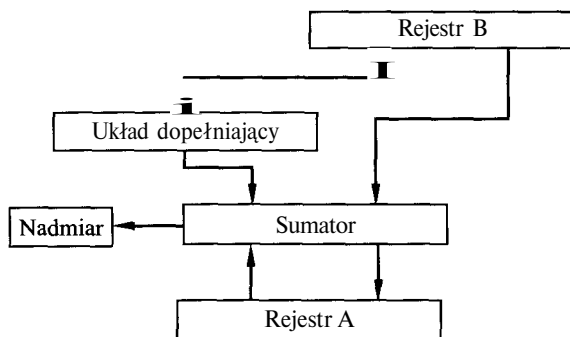
$$S' = 1100$$

$$1010$$

$$+1100$$

$$(T) \quad 0110 = \text{Przepełnienie}$$

RYСУNEK 8.4. Odejmowanie liczb w reprezentacji uzupełnienia do dwóch (M-S)



RYSUNEK 8.5. Schemat blokowy sprzętowego rozwiązania dodawania i odejmowania

w trzecim rejestrze. Wskaźnik przepełnienia jest przechowywany w 1-bitowym znaczniku przepełnienia (0 - brak przepełnienia, 1 - przepełnienie). W przypadku odejmowania odjemnik (rejestr B) przechodzi przez układ dopełniacza, dzięki czemu do sumatora jest doprowadzane uzupełnienie do dwóch.

Mnożenie

W porównaniu z dodawaniem i odejmowaniem mnożenie jest operacją złożoną niezależnie od tego, czy jest realizowane sprzętowo, czy przez oprogramowanie. W różnych komputerach wykorzystywano wiele różnych algorytmów. Tekst tego punktu ma służyć czytelnikowi jako podstawa do wyrobienia sobie poglądu na temat zwykle stosowanego podejścia.

Rozpocniemy od prostego problemu mnożenia dwóch bezznakowych (nieujemnych) liczb całkowitych, po czym zapoznamy się z najbardziej powszechnymi metodami mnożenia liczb w reprezentacji uzupełnienia do dwóch.

Bezznakowe liczby całkowite

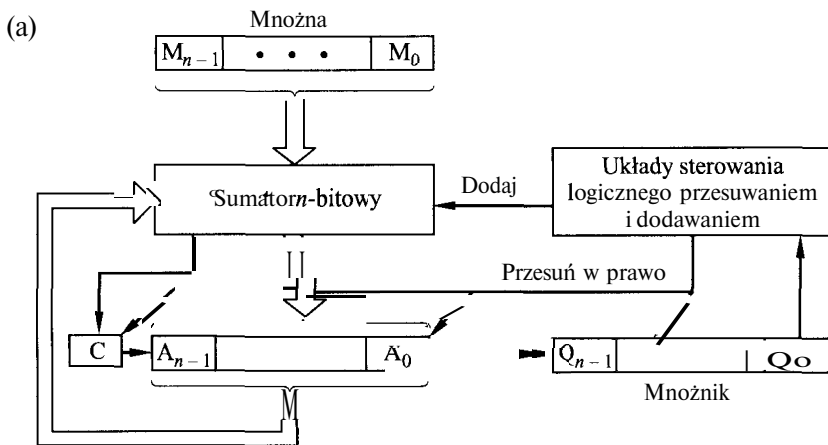
Na rysunku 8.6 jest pokazane mnożenie bezznakowych, binarnych liczb całkowitych wykonane tak, jak zwykle robi się to za pomocą ołówka i kartki papieru. Można tu poczynić kilka ważnych obserwacji:

1 0 1 1	Mnożna (11)
<u>x 1 1 0 1</u>	Mnożnik (13)
1 0 1 1	} Iloczyny cząstkowe
0 0 0 0	
1 0 1 1	
<u>1 0 1 1</u>	
1 0 0 0 1 1 1 1	Iloczyn (143)

RYSUNEK 8.6. Mnożenie binarnych liczb całkowitych bez znaku

1. Mnożenie obejmuje tworzenie iloczynów cząstkowych, po jednym dla każdej cyfry mnożnika. Iloczyny cząstkowe są następnie sumowane w celu otrzymania iloczynu końcowego.
2. Iloczyny cząstkowe są łatwe do określenia. Gdy bit mnożnika jest równy 0, iloczyn cząstkowy jest równy 0. Gdy bit mnożnika jest równy 1, iloczyn cząstkowy jest równy mnożnej.
3. Iloczyn końcowy jest otrzymywany przez sumowanie iloczynów cząstkowych. W przypadku tej operacji każdy kolejny iloczyn cząstkowy jest przesuwany o jedną pozycję w lewo względem poprzedniego iloczynu cząstkowego.
4. Wynikiem mnożenia dwóch n -bitowych binarnych liczb całkowitych jest liczba o długości do $2n$ bitów.

W porównaniu z rozwiązaniem stosowanym przy mnożeniu za pomocą ołówka i papieru można wprowadzić kilka zmian zwiększających sprawność operacji. Po pierwsze, możemy wykonywać bieżące dodawanie iloczynów



(b)

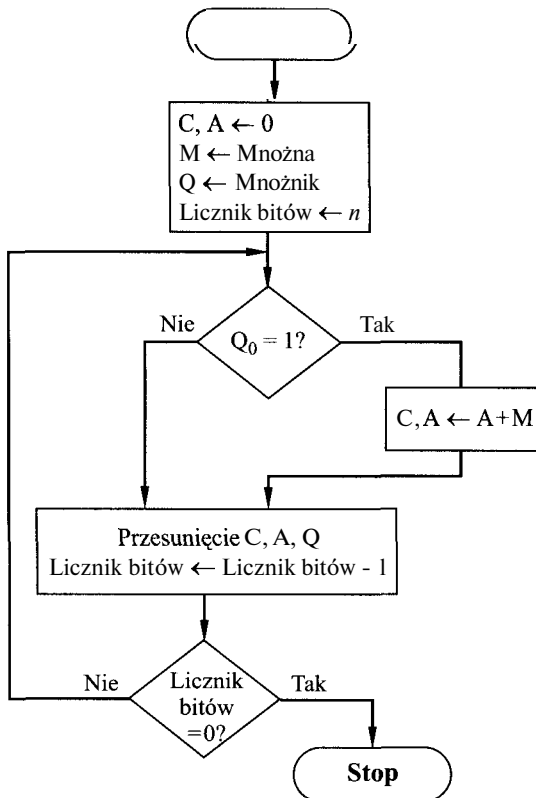
C	A	Q	Wartości początkowe		
0	0000	1101			
0	1011	1101	Dodawanie	1	Pierwszy cykl
0	0101	1110	Przesuwanie	1	
0	0010	1111	Przesuwanie	1	Drugi cykl
0	1101	1111	Dodawanie	1	
0	0110	1111	Przesuwanie	1	Trzeci cykl
1	0001	1111	Dodawanie	1	
0	1000	1111	Przesuwanie	1	Czwarty cykl (iloczyn znajduje się w A, Q)

RYSUNEK 8.7. Sprzętowe wdrożenie mnożenia liczb binarnych bez znaku (M zawiera 1011):
(a) schemat blokowy; (b) przykład (z rys. 8.6)

częstkowych, nie czekając na zakończenie. Eliminuje to potrzebę przechowywania wszystkich iloczynów częściowych; dzięki temu potrzeba mniej rejestrów. Po drugie, możemy oszczędzić nieco czasu na generowaniu wyników częściowych. Dla każdej 1 mnożnika są wymagane operacje sumowania i przesunięcia; jednak dla każdego 0 potrzebne jest tylko przesunięcie.

Na rysunku 8.7a jest pokazane rozwiązanie, w którym zastosowano te zmiany. Mnożnik i mnożna są ładowane do dwóch rejestrów (Q i M). Potrzebny jest także trzeci rejestr (A), ustawiany początkowo na 0. Występuje też 1-bitowy rejestr C, również początkowo ustawiany na 0, przechowujący potencjalny bit przeniesienia wynikający z dodawania.

Działanie układu mnożącego jest następujące. Układy logiczne sterowania odczytują mnożnik bit po bicie. Jeśli Q_0 jest równe 1, to mnożna jest dodawana do zawartości rejestru A i wynik pozostaje w rejestrze A. Następnie wszystkie bity rejestrów C, A i Q są przesuwane na prawo o jeden bit, przez co bit C przechodzi do A_{n-1} , A_0 przechodzi do Q_{n-1} , a bit Q_0 jest tracony. Jeśli Q_0 jest równy 0, to nie jest wykonywane żadne dodawanie, a jedynie przesunięcie. Proces ten jest powtarzany dla każdego bitu oryginalnego mnożnika. Wynikający



Iloczyn znajduje się w A, Q

RYSunEK 8.8. Sieć działań mnożenia liczb binarnych bez znaku

stąd $2n$ -bitowy iloczyn jest zawarty w rejestrach A i Q. Sieć działań dla tej operacji widać na rys. 8.8, zaś przykład na rys. 8.7b. Zauważmy, że w drugim cyklu, gdy bit mnożnika jest 0, nie ma operacji dodawania.

Mnożenie w notacji uzupełnienia do dwóch

Widzieliśmy, że dodawanie i odejmowanie liczb wyrażonych w notacji uzupełnienia do dwóch może być wykonywane tak, jak gdyby były one bezznakowymi liczbami całkowitymi. Rozważmy następujący przykład:

$$\begin{array}{r} 1001 \\ + 0011 \\ \hline 1100 \end{array}$$

Jeśli te liczby są traktowane jako bezznakowe liczby całkowite, to dodajemy 9 (1001) do 3 (0011) otrzymując 12 (1100). Jeśli są to liczby całkowite w notacji uzupełnienia do dwóch, to dodajemy -7 (1001) do 3 (0011) otrzymując -4 (1100).

Niestety, ten prosty schemat nie funkcjonuje w przypadku mnożenia. Żeby to stwierdzić, ponownie przeanalizujemy rys. 8.6. Pomnożyliśmy 11 (1011) przez 13 (1101) otrzymując 143 (10001111). Jeśli zinterpretujemy te liczby jako zapisane w notacji uzupełnienia do dwóch, to mamy -5 (1011) razy -3 (1101) równe -13 (10001111). Jak widać, ten prosty algorytm mnożenia nie funkcjonuje, jeśli mnożna i mnożnik są ujemne. W rzeczywistości nie działa to też poprawnie, gdy którekolwiek z nich jest ujemne. Aby to wyjaśnić, musimy wrócić do rys. 8.6 i zobaczyć, co dzieje się podczas tej operacji z potęgami 2. Przypomnijmy, że dowolna bezznakowa liczba binarna może być wyrażona jako suma potęg 2. Wobec tego

$$1101 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 2^3 + 2^2 + 2^0$$

Ponadto, mnożenie liczby binarnej przez 2^n jest dokonywane przez przesunięcie tej liczby w lewo o n bitów. Mając to na uwadze, na rys. 8.9 jest pokazane przekształcone działanie z rys. 8.6, demonstrując tworzenie iloczynów cząstkowych przez mnożenie. Jedyną różnicą na rys. 8.9 jest to, że iloczyny cząstkowe są traktowane jako liczby $2n$ -bitowe tworzone z n -bitowej mnożnej.

$$\begin{array}{r} 1011 \\ \times 1101 \\ \hline 00001011 \\ 00000000 \\ 00101100 \\ 01011000 \\ \hline 10001111 \end{array} \quad \begin{array}{l} 1011 \times 1 \times 2^0 \\ 1011 \times 0 \times 2^1 \\ 1011 \times 1 \times 2^2 \\ 1011 \times 1 \times 2^3 \end{array}$$

RYSUNEK 8.9. Mnożenie dwóch 4-bitowych liczb całkowitych bez znaku prowadzące do 8-bitowego wyniku

Wobec tego 4-bitowa mnożna 1011, jako bezznakowa liczba całkowita, jest przechowywana w postaci 8-bitowego słowa 00001011. Każdy iloczyn cząstkowy (inny niż odnoszący się do 2^0) składa się z tej liczby przesuniętej w lewo, przy czym wolne miejsca po prawej są wypełniane zerami (np. przesunięcie w lewo o dwa miejsca daje 00101100).

Możemy teraz zademonstrować, że prosty algorytm mnożenia nie będzie funkcjonował, jeśli mnożna jest ujemna. Problemem jest to, że każdy wkład ujemnej mnożnej w postaci iloczynu cząstkowego musi być liczbą ujemną na polu $2n$ -bitowym; bity znaków iloczynów cząstkowych muszą pozostawać w szeregu. Można to zobaczyć na rys. 8.10, na którym jest pokazane mnożenie 1001 przez 0011. Jeśli liczby te są traktowane jako bezznakowe liczby całkowite, to mnożenie $9 \times 3 = 27$ przebiega prosto. Jeżeli jednak 1001 jest interpretowana jako -7 w notacji uzupełnienia do dwóch, to każdy iloczyn cząstkowy musi być ujemną liczbą $2n$ -bitową w notacji uzupełnienia do dwóch, co widać na rys. 8.10b. Zauważmy, że można to uzyskać przez przesunięcie każdego iloczynu cząstkowego w lewo za pomocą binarnych 1.

Powinno być także jasne, że jeśli mnożnik jest ujemny, to prosty algorytm mnożenia również nie funkcjonuje. Przyczyną jest to, że bity mnożnika nie odpowiadają już przesunięciom (lub mnożeniom), które muszą mieć miejsce. Na przykład:

$$-3 = 1101 = -(0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) = -2^1 - 2^0$$

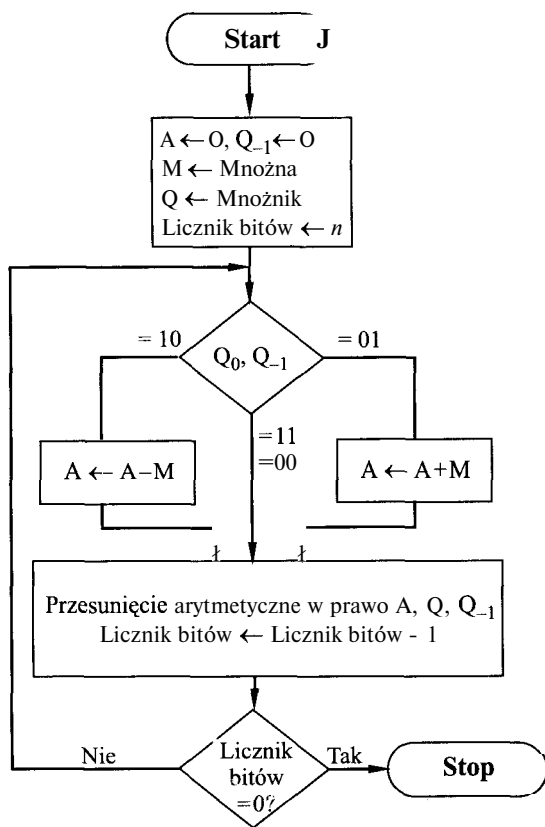
Mnożnik nie może więc być używany bezpośrednio w sposób opisany wyżej.

Istnieje wiele sposobów rozwiązania tego problemu. Jeden z nich polega na przekształceniu mnożnej i mnożnika na liczby dodatnie, przeprowadzeniu mnożenia i utworzeniu reprezentacji wyniku w postaci uzupełnienia do dwóch wtedy i tylko wtedy, gdy znaki dwóch liczb oryginalnych różniły się. Projektanci wolą jednak stosować metody, które nie wymagają tej ostatniej transformacji. Jedną z najpowszechniejszych jest algorytm Bootha [BOOTH51]. Algorytm ten ma także tę zaletę, że przyspiesza proces mnożenia w porównaniu z prostszymi rozwiązaniami.

Algorytm Bootha jest przedstawiony na rys. 8.11. Można go opisać następująco. Jak poprzednio, mnożnik i mnożna są umieszczane odpowiednio w rejestrach Q i M. Istnieje także rejestr 1-bitowy, umieszczony logicznie na prawo od

(a) <pre> 1 0 0 1 (9) ____x00 1 1 (3) 0 0 0 0 1 0 0 1 (1001) × 2⁰ 0 0 0 1 0 0 1 0 (1001) × 2¹ 0 0 0 1 1 0 1 1 (27) </pre>	(b) <pre> 1 0 0 1 (-7) ____x0 0 1 1 (3) 1 1 1 1 1 0 0 1 (-7) × 2⁰ = (-7) 1 1 1 1 0 0 1 0 (-7) × 2¹ = (-14) 1 1 1 0 1 0 1 1 (-21) </pre>

RYSUNEK 8.10. Porównanie mnożenia liczb całkowitych bez znaku i liczb całkowitych w notacji uzupełnienia do dwóch: (a) liczby całkowite bez znaku; (b) liczby całkowite w notacji uzupełnienia do dwóch



RYСУNEK 8.11. Algorytm Bootha mnożenia liczb w notacji uzupełnienia do dwóch

najmniej znaczącego bitu (Q_0) rejestru Q i oznaczony Q_{-1} ; jego zastosowanie wyjaśnimy wkrótce. Wyniki mnożenia będą się pojawiały w rejestrach A i Q . Rejestry A i Q_{-1} są początkowo ustawiane na 0. Jak poprzednio, układy logiczne sterowania przeglądają mnożnik bit po bicie. Podczas badania każdego bitu badany jest również bit znajdujący się z jego prawej strony. Jeśli oba bity są takie same (1-1 lub 0-0), to wszystkie bity rejestrów A , Q i Q_{-1} są przesuwane w prawo o jeden bit. Jeśli te dwa bity różnią się, to mnożna jest dodawana lub odejmowana od zawartości rejestru A , zależnie od tego, czy bity te są 0-1, czy 1-0. Po dodawaniu lub odejmowaniu następuje przesunięcie w prawo. W każdym przypadku przesunięcie w prawo jest takie, że najdalej na lewo położony bit A , tzn. A_{n-1} , nie tylko jest przesuwany do A_{n-2} , ale także pozostaje w A_{n-1} . Jest to wymagane dla zachowania znaku liczby w A i Q . Jest to określane jako **przesunięcie arytmetyczne**, ponieważ jest zachowywany bit znaku.

Na rysunku 8.12 widać sekwencję zdarzeń w algorytmie Bootha w przypadku mnożenia 7 przez 3. W sposób bardziej skondensowany ta sama operacja została przedstawiona na rys. 8.13a. W pozostałej części rys. 8.13 są podane inne przykłady stosowania tego algorytmu. Jak można zauważyć, działa on popraw-

A	Q	Q ₋₁		
0000	001 1	0	Stan początkowy	
1001	001 1	0	$A \leftarrow A - M$	} Pierwszy cykl
1000	1001	1	Przesunięcie	
1 1 1 0	0 1 0 0	1	Przesunięcie	) Drugi cykl
0101	0 1 0 0	1	$A \leftarrow A + M$	
0010	1010	0	Przesunięcie	) Trzeci cykl
0001	0101	0	Przesunięcie	
				} Czwarty cykl (iloczyn znajduje się w A, Q)

RYSUNEK 8.12. Przykład algorytmu Bootha (M zawiera 0111)

(a)		(b)	
$ \begin{array}{r} 0111 \\ \times 0011 (0) \\ \hline 11111001 \quad 1-0 \\ 0000000 \quad 1-1 \\ 000111 \quad 0-1 \\ \hline 00010101 \quad (21) \end{array} $		$ \begin{array}{r} 0111 \\ \times 1101 (0) \\ \hline 11111001 \quad 1-0 \\ 0000111 \quad 0-1 \\ 111001 \quad 1-0 \\ \hline 11101011 \quad (-21) \end{array} $	
(c)		(d)	
$ \begin{array}{r} 1001 \\ \times 0011 (0) \\ \hline 00000111 \quad 1-0 \\ 0000000 \quad 1-1 \\ 111001 \quad 0-1 \\ \hline 11101011 \quad (-21) \end{array} $		$ \begin{array}{r} 1001 \\ \times 1101 (0) \\ \hline 00000111 \quad 1-0 \\ 1111001 \quad 0-1 \\ 000111 \quad 1-0 \\ \hline 00010101 \quad (21) \end{array} $	

RYSUNEK 8.13. Przykłady zastosowania algorytmu Bootha: (a) $(7) \times (3) = (21)$; (b) $= (7) \times (-3) = (-21)$; (c) $(-7) \times (3) = (-21)$; (d) $(-7) \times (-3) = (21)$

nie z dowolną kombinacją liczb dodatnich i ujemnych. Zwróćmy też uwagę na efektywność tego algorytmu. Bloki jedynek lub zer są przeskakiwane, przy czym na blok przypada średnio tylko jedno dodawanie lub odejmowanie.

Dlaczego algorytm Bootha funkcjonuje? Rozważmy najpierw przypadek dodatniego mnożnika. W szczególności rozważmy dodatni mnożnik składający się z jednego bloku jedynek otoczonego przez zera, np. 00011110. Jak wiemy, mnożenie można wykonać przez sumowanie odpowiednio przesuniętych kopii mnożnej:

$$M \times (00011110) = M \times (2^4 + 2^3 + 2^2 + 2^1) = M \times (16 + 8 + 4 + 2) = M \times 30$$

Liczba takich operacji może być zredukowana do dwóch, jeśli zauważymy, że

$$2^n : 2^{n-1} : \dots : 2^{n-K} = 2^{n+1} \quad 2^{n-K}$$

Wobec tego

$$M \times (00011110) = M \times (2^5 - 2^1) = M \times (32 - 2) = M \times 30$$

Tak więc iloczyn może być generowany przez jedno dodawanie i jedno odejmowanie mnożnej. Schemat ten dotyczy dowolnej liczby bloków jedynek w mnożniku, włącznie z przypadkiem pojedynczej 1 traktowanej jako blok. Wobec tego

$$M \times (00011110) = M \times (2^6 + 2^5 + 2^4 + 2^3 + 2^1) = M \times (2^7 - 2^3 + 2^2 - 2^1)$$

Algorytm Bootha działa według tego właśnie schematu; wykonywane jest odejmowanie, gdy napotkana jest pierwsza 1 bloku (1-0), oraz dodawanie, gdy napotkany jest koniec bloku (0-1).

Aby wykazać, że taki sam schemat funkcjonuje w przypadku ujemnego mnożnika, musimy zwrócić uwagę na następujące zależności. Niech X będzie liczbą ujemną w notacji uzupełnienia do dwóch:

$$\text{Reprezentacja } X = \{1x_n - 2x_{n-3} \dots x \setminus x_0\}$$

Wobec tego wartość X może być wyrażona następująco:

$$X = -2^{n-1} + x_{n-2} \times 2^{n-2} + x_{n-3} \times 2^{n-3} + \dots + x_1 \times 2^1 + x_0 \times 2^0 \quad (8-3)$$

Można to zweryfikować, stosując ten algorytm do liczb w tabeli 8.1.

Wiemy teraz, że najbardziej na lewo położonym bitem liczby X jest 1, ponieważ X jest ujemna. Założmy, że najdalej na lewo położone 0 znajduje się w pozycji k -tej. Liczba X ma więc postać:

$$\text{Reprezentacja } X = \{111 \dots 10x_{k-1}x_{k-2} \dots x_1 x_0\} \quad (8-4)$$

Wobec tego wartość X wynosi

$$X = -2^{n-1} + 2^{n-2} + \dots + 2^{k+1} + x_{k-1} \times 2^{k-1} + \dots + x_0 \times 2^0 \quad (8-5)$$

Na podstawie równania (8-2) możemy stwierdzić, że

$$2^{n-2} + 2^{n-3} + \dots + 2^{k+1} = 2^{n-1} - 2^{k+1}$$

Po przestawieniu

$$-2^{n-1} + 2^{n-2} + 2^{n-3} + \dots + 2^{k+1} = -2^{k+1} \quad (8-6)$$

Podstawiając równanie (8-6) do (8-5), otrzymujemy:

$$X = -2^{k+1} + x_{k-1} \times 2^{k-1} + \dots + x_0 \times 2^0 \quad (8-7)$$

Na zakończenie możemy wrócić do algorytmu Bootha. Jeśli weźmie się pod uwagę reprezentację X (równanie (8-4)), to staje się jasne, że wszystkie bity począwszy od X_0 aż do najdalej na lewo położonego 0 są traktowane właściwie, ponieważ umożliwiają uzyskanie wszystkich składników równania (8-7) z wyjątkiem (-2^{k+1}) i mają wobec tego właściwą postać. Gdy algorytm przegląda naj-

dalej na lewo położone 0 i napotyka następną 1 (2^{k+1}), następuje przejście 1-0 i ma miejsce odejmowanie (-2^{k+1}). Jest to brakujący człon równania (8-7).

Jako przykład rozważmy mnożenie pewnej mnożnej przez -6. W notacji uzupełnienia do dwóch i przy zastosowaniu słowa 8-bitowego -6 jest reprezentowane przez 11111010. Na podstawie równania (8-3) wiemy, że

$$-6 = -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1$$

co można z łatwością sprawdzić. Wobec tego

$$M2 \times (11111010) = M \times (-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1)$$

Po zastosowaniu równania (8-7)

$$M \times (11111010) = M \times (-2^3 + 2^1)$$

co, jak można zweryfikować, nadal oznacza $M \times (-6)$. Na zakończenie, powtarzając naszą linię rozumowania, otrzymujemy

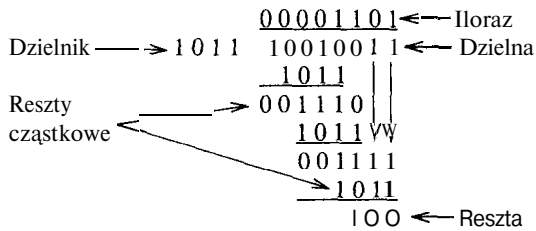
$$M \times (11111010) = M \times (-2^3 + 2^2 + 2^1)$$

Teraz możemy już stwierdzić, że algorytm Bootha działa według tego właśnie schematu. Wykonuje on odejmowanie po napotkaniu pierwszej 1 (1-0), dodawanie po napotkaniu (0-1) i wreszcie następne odejmowanie po napotkaniu pierwszej 1 z następnego bloku jedynek. Wobec tego algorytm Bootha wykonuje mniej dodawań i odejmowań niż algorytm prostszy.

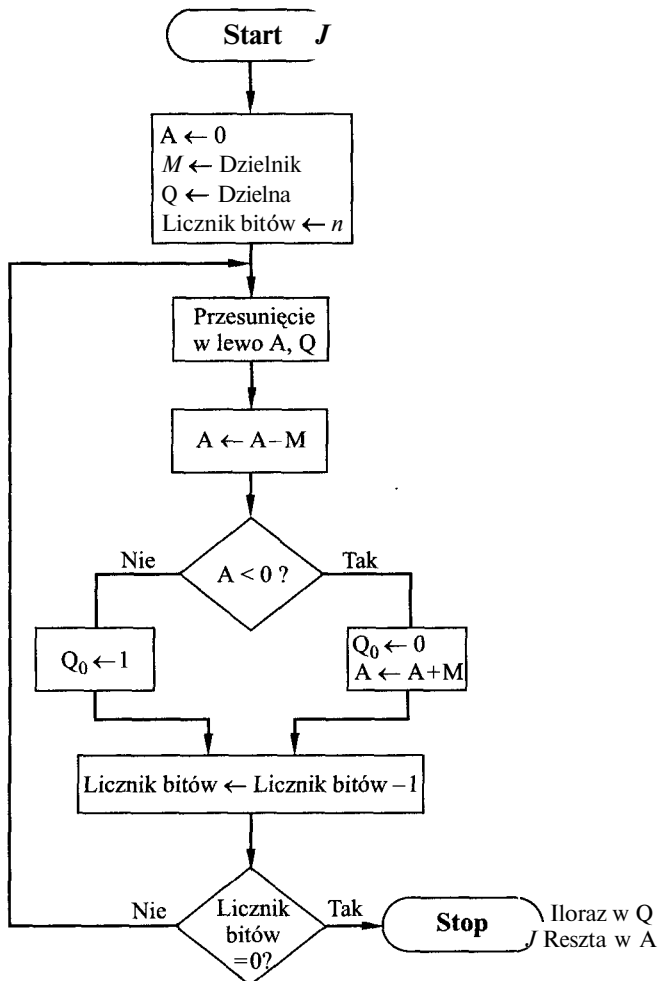
Dzielenie

Dzielenie jest nieco bardziej złożone niż mnożenie, jednak opiera się na takich samych zasadach ogólnych. Jak poprzednio, podstawą algorytmu jest rozwiązanie stosowane przy obliczeniach za pomocą ołówka i papieru, a sama operacja obejmuje powtarzające się przesuwanie oraz dodawanie lub odejmowanie.

Na rysunku 8.14 jest pokazany przykład długiego dzielenia bezznakowych binarnych liczb całkowitych. Pouczające jest szczegółowe opisanie tego procesu. Po pierwsze, bity dzielnej są badane od lewej do prawej, aż zespół zbadanych bitów będzie reprezentował liczbę większą lub równą dzielnikowi; jest to określane jako sprawdzanie, czy dzielnik może *podzielić* liczbę. Aż do wystąpienia tego zdarzenia w ilorazie są umieszczane zera od lewej do prawej. Gdy wystąpi już to zdarzenie, w ilorazie umieszczana jest 1, a dzielnik jest odejmowany od dzielnej częściowej. Wynik jest określany jako *reszta częściowa*. Potem od tego punktu, dzielenie powtarza się cyklicznie. W każdym cyklu dodatkowe bity dzielnej są dołączane do reszty częściowej, aż wynik będzie większy lub równy dzielnikowi. Jak poprzednio, dzielnik jest odejmowany od tej liczby, przez co powstaje nowa reszta częściowa. Proces jest kontynuowany, aż wszystkie bity dzielnej zostaną zużyte.



RYSUNEK 8.14. Dzielenie całkowitych liczb binarnych bez znaku



RYSUNEK 8.15. Sieć działań dzielenia liczb binarnych bez znaku

Na rysunku 8.15 jest pokazany algorytm maszynowy, który odpowiada procesowi dzielenia. Dzielnik jest umieszczany w rejestrze M, a dzielna w rejestrze Q. Na każdym etapie zawartości A i Q są razem przesuwane w lewo o jeden bit. Zawartość M jest odejmowana od A w celu określenia, czy A dzieli resztę częściową. Jeśli tak, w Q_0 jest umieszczany bit 1. W przeciwnym razie Q_0 dostaje bit 0, a zawartość M musi być dodana z powrotem do zawartości A w celu odtworzenia poprzedniej wartości. Stanowi to zakończenie cyklu, a proces jest kontynuowany przez n cykli. Na zakończenie iloraz znajduje się w rejestrze Q, a reszta w rejestrze A.

Z pewną trudnością proces ten może być rozciągnięty na liczby ujemne. Przedstawimy tu jedno z podejść odnoszące się do liczb w notacji uzupełnienia do dwóch. Kilka przykładów tego podejścia widać na rys. 8.16. Algorytm można podsumować następująco:

1. Załaduj dzielnik do rejestru M, a dzielną do rejestrów A i Q. Dzielna musi być wyrażona jako liczba w reprezentacji uzupełnienia do dwóch. Wobec tego, na przykład, 4-bitowa liczba 0111 staje się 00000111, a 1001 jest przekształcana na 111111001.
2. Przesuń A i Q w lewo o jedną pozycję bitową.
3. Jeśli M i A mają takie same znaki, przeprowadź operację $A \leftarrow A - M$; w przeciwnym razie $A \leftarrow A + M$.
4. Powyższa operacja jest udana, jeśli znak A pozostaje taki sam po zakończeniu operacji.
 - (a) Jeśli operacja jest udana lub ($A = 0$ i $Q = 0$), to ustaw $Q_0 \leftarrow 1$.
 - (b) Jeśli operacja jest nieudana i ($A \neq 0$ lub $Q \neq 0$), to ustaw $Q_0 \leftarrow 0$ i przywróć poprzednią wartość A.
5. Powtórz kroki od 2 do 4 tyle razy, ile jest pozycji bitowych w Q.
6. Reszta jest zawarta w A. Jeśli znaki dzielnika i dzielnej były takie same, to iloraz jest w Q; w przeciwnym razie poprawny iloraz jest uzupełnieniem do dwóch zawartości rejestru Q.

Na podstawie rysunku 8.16 można zauważyć, że dzieląc -7 przez 3 oraz 7 przez -3 otrzymamy różne reszty. Dzieje się tak, ponieważ reszta jest zdefiniowana następująco:

$$D = Q \times V + R$$

gdzie

D - dzielna,

Q - iloraz,

V - dzielnik,

R - reszta.

Wyniki na rysunku 8.16 są zgodne z tym wzorem.

(a)			(b)		
A	Q	M = 0011	A	Q	M = 1101
0000	0111	Wartość początkowa	0000	0111	Wartość początkowa
0000	1110	Przesunięcie	0000	1110	Przesunięcie
1101		Odejmowanie	1101		Dodawanie
0000	1110	Przywrócenie	0000	1110	Przywrócenie
0001	1100	Przesunięcie	0001	1100	Przesunięcie
1110		Odejmowanie	1110		Dodawanie
0001	1100	Przywrócenie	0001	1100	Przywrócenie
0011	1000	Przesunięcie	0011	1000	Przesunięcie
0000		Odejmowanie	0000		Dodawanie
0000	1001	Ustawienie $Q_0 = 1$	0000	1001	Ustawienie $Q_0 = 1$
0001	0010	Przesunięcie	0001	0010	Przesunięcie
1110		Odejmowanie	1110		Dodawanie
0001	0010	Przywrócenie	0001	0010	Przywrócenie
(c)			(d)		
A	Q	M = 0011	A	Q	M = 1101
1111	1001	Wartość początkowa	1111	1001	Wartość początkowa
1111	0010	Przesunięcie	1111	0010	Przesunięcie
0010		Dodawanie	0010		Odejmowanie
1111	0010	Przywrócenie	1111	0010	Przywrócenie
1110	0100	Przesunięcie	1110	0100	Przesunięcie
0001		Dodawanie	0001		Odejmowanie
1110	0100	Przywrócenie	1110	0100	Przywrócenie
1100	1000	Przesunięcie	1100	1000	Przesunięcie
1111		Dodawanie	1111		Odejmowanie
1111	1001	Ustawienie $Q_0 = 1$	1111	1001	Ustawienie $Q_0 = 1$
1111	0010	Przesunięcie	1111	0010	Przesunięcie
0010		Dodawanie	0010		Odejmowanie
1111	0010	Przywrócenie	1111	0010	Przywrócenie

RYSUNEK 8.16. Przykłady dzielenia w notacji uzupełnienia do dwóch: (a) $7 \div 3$; (b) $7 \div (-3)$; (c) $(-7) \div 3$; (d) $(-7) \div (-3)$

8.4

REPREZENTACJA ZMIENNOPOZYCYJNA

Zasady

W przypadku notacji stałopozycyjnej (np. uzupełnienia do dwóch) możliwe jest reprezentowanie zakresu dodatnich i ujemnych liczb całkowitych ze środkiem w zerze. Przy założeniu ustalonego przecinka pozycyjnego format ten umożliwia również reprezentację liczb ze składnikiem ułamkowym.

Podejście to ma ograniczenia. Nie mogą być reprezentowane ani bardzo duże liczby, ani bardzo małe ułamki. Ponadto, ułamkowe składniki ilorazu przy dzieleniu dwóch dużych liczb mogą być utracone.

W przypadku liczb dziesiętnych obchodzi się to ograniczenie stosując notację naukową. Wtedy 976 000 000 000 000 może być reprezentowana jako $9,76 \times 10^{14}$, a 0,0000000000000976 jako $9,76 \times 10^{-14}$. To, co zrobiliśmy, jest w rezultacie dynamicznym przesunięciem przecinka dziesiętnego do wygodnego położenia i użyciem potęgi 10 do określania rzeczywistego położenia tego przecinka. Umożliwia to reprezentowanie bardzo dużych i bardzo małych liczb za pomocą tylko kilku cyfr.

To samo podejście może być zastosowane do liczb binarnych. Możemy reprezentować liczbę w postaci

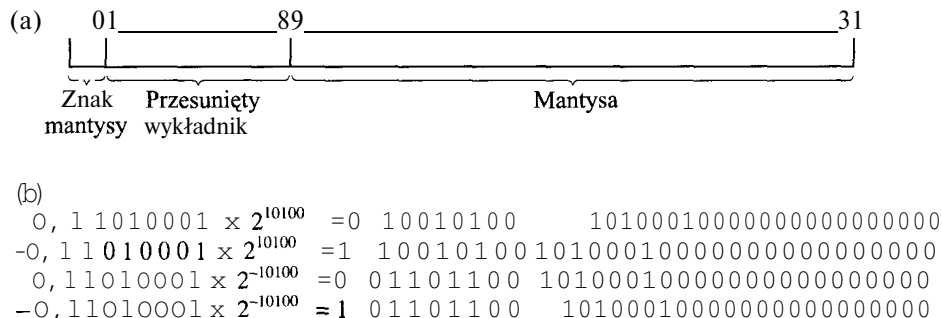
$$\pm S \times B^{\pm E}$$

Liczba ta może być przechowywana w postaci słowa binarnego o trzech polach zawierających

- a znak: plus lub minus,
- mantysę S,
- a wykładnik E.

Podstawa B jest ustalona i nie musi być przechowywana, ponieważ jest taka sama dla wszystkich liczb.

Zasady stosowane przy reprezentowaniu zmiennopozycyjnych liczb binarnych najlepiej wyjaśnić na przykładzie. Na rysunku 8.17a jest pokazany typowy, 32-bitowy format zmiennopozycyjny. Lewy bit reprezentuje znak liczby (0 - dodatnia, 1 - ujemna). Wartość wykładnika jest przechowywana w bitach od 1 do 8. Zastosowana tu reprezentacja jest znana jako **reprezentacja przesunięta**. Ustalona wartość, zwana przesunięciem, jest odejmowana od pola w celu otrzymania prawdziwej wartości wykładnika. W tym przypadku za pomocą pola 8-bitowego można reprezentować liczby od 0 do 255. Po zastosowaniu przesunięcia równego 128 prawdziwe wartości wykładnika znajdują się w zakresie od -128 do +127. W tym przykładzie założyliśmy, że podstawa jest równa 2.



RYSunEK 8.17. Typowy 32-bitowy format zmiennopozycyjny: (a) format; (b) przykłady

Ostatnią częścią słowa jest mantysa. Liczba zmiennopozycyjna może być wyrażana na wiele sposobów. Następujące z nich są równoważne:

$$\begin{aligned} 0,110 \times 2^5 \\ 110 \times 2^2 \\ 0,0110 \times 2^6 \end{aligned}$$

i tak dalej. W celu uproszczenia operacji na liczbach zmiennopozycyjnych wymagane jest zwykle, aby były one znormalizowane. W naszym przykładzie liczbą znormalizowaną jest liczba w postaci

$$\pm 0,1bbb \dots b \times 2^{\pm E}$$

gdzie b jest dowolną cyfrą binarną (0 lub 1). Implikuje to, że lewy bit mantysy jest zawsze równy 1. Nie jest więc konieczne przechowywanie tego bitu. Wobec tego pole 23-bitowe jest używane do przechowywania 24-bitowej mantysy o wartości między 0,5 a 1,0.

Na rysunku 8.17b są podane przykłady liczb przedstawionych w tym formacie. Zwróć uwagę na następujące własności:

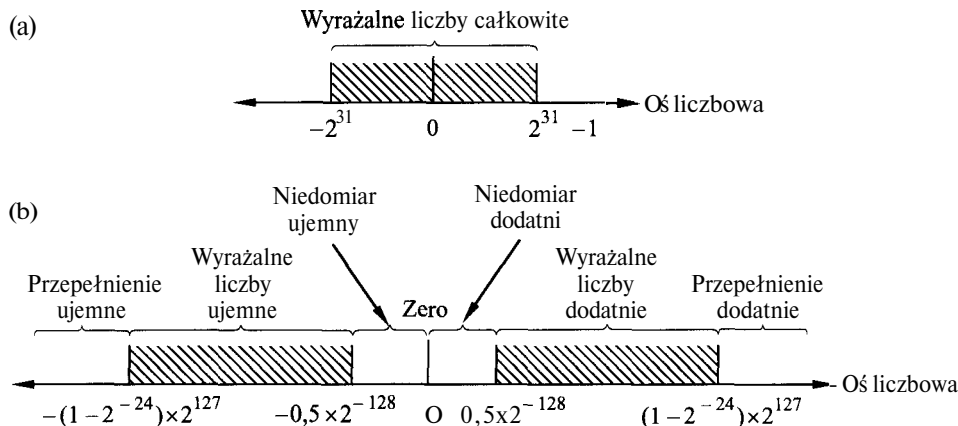
- a Znak jest zawarty w pierwszym bicie słowa.
- Pierwszy bit rzeczywistej mantysy jest zawsze 1 i nie musi być przechowywany w polu znaczącym.
- a W celu przechowywania w polu wykładnika do rzeczywistej wartości wykładnika jest dodawana wartość 128.
- a Podstawa wynosi 2.

Na rysunku 8.18 jest pokazany zakres liczb, które mogą być zapisane w słowie 32-bitowym przy założeniu tej reprezentacji. Przy zapisaniu liczb całkowitych w notacji uzupełnienia do dwóch mogą być reprezentowane wszystkie liczby całkowite od -2^{31} do $2^{31} - 1$, co oznacza 2^{32} różnych liczb. Przyjęcie przykładowego formatu zmiennopozycyjnego z rys. 8.17 umożliwia przedstawienie liczb z następujących zakresów:

- a liczby ujemne między $-(1 - 2^{-24}) \times 2^{127}$ a $-0,5 \times 2^{-128}$;
- liczby dodatnie między $0,5 \times 2^{-128}$ a $(1 - 2^{-24}) \times 2^{127}$.

Powyższe zakresy nie obejmują pięciu obszarów:

- a liczby ujemne mniejsze niż $-(1 - 2^{-24}) \times 2^{127}$, określane jako **przepełnienie ujemne**;
- liczby ujemne większe niż $-0,5 \times 2^{-128}$, określane jako **niedomiar ujemny**;
- a zero;
- a liczby dodatnie mniejsze niż $0,5 \times 2^{-128}$, określane jako **niedomiar dodatni**;
- a liczby dodatnie większe niż $(1 - 2^{-24}) \times 2^{127}$, określane jako **przepełnienie dodatnie**.

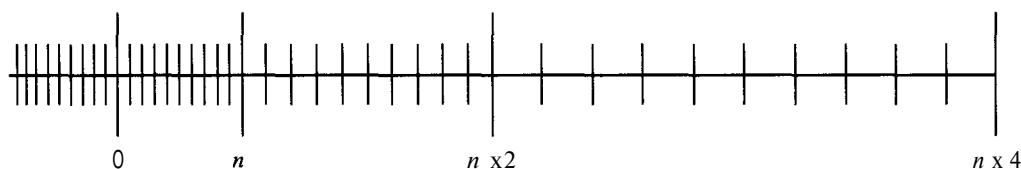


RYSUNEK 8.18. Liczby wyrażalne za pomocą typowych formatów 32-bitowych: (a) liczby całkowite w notacji uzupełnienia do dwóch; (b) liczby zmiennopozycyjne

Za pomocą tej reprezentacji nie da się przedstawić wartości 0. Jednak, jak zobaczymy, stosowane reprezentacje zmiennopozycyjne obejmują specjalny wzór bitowy do oznaczania zera. Przepięnienie następuje, gdy wynikiem operacji arytmetycznej jest liczba większa niż może być wyrażona przez potęgę 127 (np. $2^{120} \times 2^{100} = 2^{220}$). Niedomiar występuje natomiast wtedy, gdy wielkość ułamkowa jest zbyt mała (np. $2^{-120} \times 2^{-100} = 2^{-220}$). Niedomiar stanowi mniej poważny problem, ponieważ wynik może być na ogół zadowalająco aproksymowany przez 0.

Ważne jest zwrócenie uwagi na to, że za pomocą notacji zmiennopozycyjnej nie reprezentuje się większej liczby pojedynczych wartości. Maksymalna liczba różnych wartości, które mogą być przedstawione za pomocą 32 bitów wynosi nadal 2^{32} . To, czego dokonaliśmy, to tylko rozproszenie tych liczb na dwa zakresy, dodatni i ujemny.

Zauważmy również, że liczby reprezentowane w notacji zmiennopozycyjnej nie są rozmieszczone równomiernie na osi liczb, jak liczby stałopozycyjne. Możliwe wartości są rozłożone gęściej na początku osi, a rzadziej w miarę oddalania się od początku, co widać na rys. 8.19. Jest to jeden z kompromisów matematyki zmiennopozycyjnej: wiele obliczeń prowadzi do wyników, które nie są dokładne i muszą być zaokrąglane do najbliższych wartości możliwych do reprezentowania za pomocą tej notacji.



RYSUNEK 8.19. Gęstość liczb zmiennopozycyjnych

W przypadku formatu pokazanego na rys. 8.17 następuje wymiennosc między zakresem i dokladnoscia. W przykladzie tym 8 bitow przeznaczono na wykladnik, a 23 na mantyse. Ješli zwikszymy liczbe bitow wykladnika, to poszerzymy w ten sposob zakres liczb mozliwych do wyrazenia. Poniewaz jednak tylko ustalona liczba roznych wartosci moze byc wyrazona, zredukujemy w ten sposob gestosc tych liczb i tym samym dokladnosc. Jedyna droga zwikszenia zarowno zakresu, jak i dokladnosc, jest uzycie wiekszej liczby bitow. Dlatego wieksznosc komputerow oferuje liczby w formacie o **pojedynczej i podwójnej precyzji**. Na przyklad, format pojedynczej precyzji moze miec 32 bity, a podwójnej - 64 bity.

Wystepuje wiec wymiennosc między liczbą bitow wykladnika a liczbą bitow mantysy. Jednak sprawa jest jeszcze bardziej zlozona. Przyjeta domyslnie podstawa wykladnika nie musi wynosic 2. Na przyklad w architekturze IBM S/370 zastosowano podstawe 16. Format sklada sie z 7-bitowego wykladnika i 24-bitowej mantysy. Tak wiec

$$0,11010001 \times 2^{10100} = 0,11010001 \times 16^{101}$$

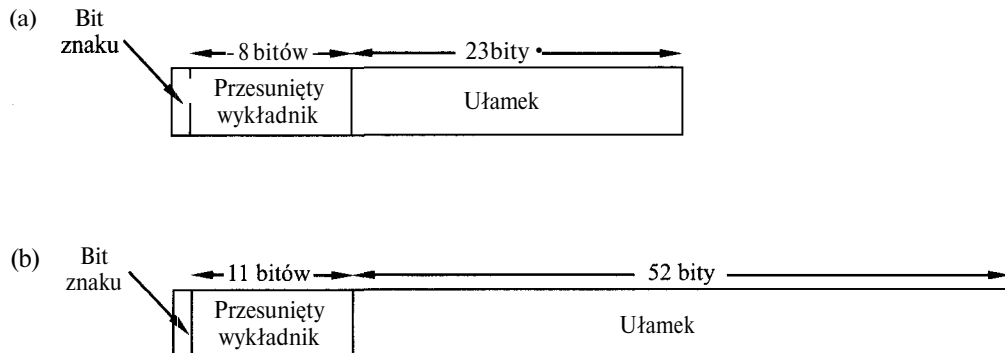
zaś wykladnik reprezentuje liczbe 5, a nie 20.

Zaletą stosowania wiekszej podstawy jest mozliwosc uzyskania wiekszego zakresu liczb przy tej samej liczbie bitow wykladnika. Pamietajmy jednak, ze nie zwikszyliśmy w ten sposob liczby roznych wartosci, ktore moga byc reprezentowane. W przypadku ustalonego formatu wieksza podstawa wykladnika daje wiekszy zakres liczb kosztem dokladnosc.

Norma IEEE dotycząca zmiennopozycyjnej arytmetyki binarnej

Najwazniejsza reprezentacja zmiennopozycyjna jest zdefiniowana w normie IEEE Standard 754 [IEEE85]. Normę tę opracowano, aby ulatwic przenoszenie programow z jednego procesora do drugiego oraz aby pobudzic rozwój zlozonych programow numerycznych. Norma ta zostala szeroko zaakceptowana i jest stosowana praktycznie we wszystkich wspolczesnych procesorach i koprocessorach arytmetycznych.

Norma IEEE określa zarowno pojedynczy format 32-bitowy, jak i podwójny 64-bitowy (rys. 8.20), z wykladnikami odpowiednio 8-bitowym i 11-bitowym. Domyślna podstawa wynosi 2. Norma definiuje ponadto dwa formaty rozszerzone, pojedynczy i podwójny, ktorych dokladne rozwiazania zaleza od implementacji. Formaty rozszerzone obejmują dodatkowe bity wykladnika (rozszerzony zakres) i mantysy (zwikszona dokladnosc). Są one przewidziane do stosowania w obliczeniach posrednich. Dzieki zwikszonej dokladnosc, stosując rozszerzone formaty, zmniejsza sie grozba, ze koncowy wynik bedzie niekształcony przez nadmierny blad zaokraglania; dzieki zaś wiekszemu zakresowi, maleje grozba posredniego przepełnienia powodujacego odrzucenie obliczeń,



RYSUNEK 8.20. Formaty IEEE 754: (a) format pojedynczy; (b) format podwójny

których wynik byłby możliwy do przedstawienia w formacie podstawowym. Dodatkowym powodem stosowania formatu pojedynczego są oferowane przezeń pewne korzyści formatu podwójnego bez zwiększania czasochłonności, towarzyszącej zwykle większej dokładności. W tabelicy 8.2 są przedstawione parametry tych czterech formatów.

TABELA 8.2. Parametry formatu IEEE 754

Parametr	Format			
	pojedynczy	pojedynczy rozszerzony	podwójny	podwójny rozszerzony
Długość słowa (bitów)	32	≥ 43	64	≥ 79
Długość wykładnika (bitów)	8	≥ 11	11	≥ 15
Przesunięcie wykładnika	127	nieokreślone	1023	nieokreślone
Maksymalny wykładnik	127	≥ 1023	1023	$> 16\,383$
Minimalny wykładnik	-126	≤ -1022	-1022	$\leq -16\,382$
Zakres liczb (podstawa 10)	$10^{-38}, 10^{+38}$	nieokreślony	$10^{-308}, 10^{+308}$	nieokreślony
Długość podstawy (bitów)	23	≥ 31	52	≥ 63
Liczba wykładników	254	nieokreślona	2046	nieokreślona
Liczba ułamków	2^{23}	nieokreślona	2^{52}	nieokreślona
Liczba wartości	$1,98 \times 2^{31}$	nieokreślona	$1,99 \times 2^{63}$	nieokreślona

Nie wszystkie wzory bitowe formatów IEEE są interpretowane w zwykły sposób; niektóre z nich są używane do reprezentowania wartości specjalnych. W tabeli 8.3 są podane wartości przypisane różnym wzorom bitowym. Ekstremalne wartości wykładnika w postaci samych zer (0) i samych jedynek (255 w przypadku formatu pojedynczego, 2047 w przypadku podwójnego) określają wartości specjalne. Reprezentowane są następujące klasy liczb:

- a Przy wartościach wykładnika w zakresie od 1 do 254 w przypadku formatu pojedynczego oraz od 1 do 2046 w przypadku podwójnego reprezentowane są znormalizowane, niezerowe liczby zmiennopozycyjne. Wykładnik jest przesunięty, wobec tego jego zakres zawiera się od -126 do +127 w przy-

TABELA 8.3. Interpretacja liczb zmiennopozycyjnych IEEE 754

	Pojedyncza precyzja (32 bity)				Podwójna precyzja (64 bity)			
	znak	przesunięty wykładnik	ułamek	wartość	znak	przesunięty wykładnik	ułamek	wartość
Zero								
dodatnie	0	0	0	0	0	0	0	0
Zero								
ujemne	1	0	0	-0	1	0	0	-0
Plus nieskończoność	0	255 (same 1)	0	∞	0	2047 (same 1)	0	∞
Minus nieskończoność	0	255 (same 1)	0	$-\infty$	0	2047 (same 1)	0	$-\infty$
Ciche NaN	0 lub 1	255 (same 1)	$\neq 0$	NaN	0 lub 1	2047 (same 1)	$\neq 0$	NaN
Sygnalizacyjne NaN	0 lub 1	255 (same 1)	$\neq 0$	NaN	0 lub 1	2047 (same 1)	$\neq 0$	NaN
Dodatnia znormalizowana niezerowa	0	$0 < e < 255$	f	$2^{e-127} (1,f)$	0	$0 < e < 2047$	f	$2^{e-1023} (1,f)$
Ujemna znormalizowana niezerowa	1	$0 < e < 255$	f	$-2^{e-127} (1,f)$	1	$0 < e < 2047$	f	$-2^{e-1023} (1,f)$
Dodatnia nieznormalizowana	0	0	$f \neq 0$	$2^{e-126} (0,f)$	0	0	$f \neq 0$	$2^{e-1022} (0,f)$
Ujemna nieznormalizowana	1	0	$f \neq 0$	$-2^{e-126} (0,f)$	1	0	$f \neq 0$	$-2^{e-1022} (0,f)$

padku formatu pojedynczego i od -1022 do +1023 w przypadku podwójnego. Znormalizowana liczba wymaga 1 bitu na lewo od przecinka binarnego; bit ten jest domyślny, co daje efektywnie 24-bitową lub 53-bitową mantysę (określaną w normie jako *frakcja*).

- Wykładnik równy zeru w połączeniu z frakcją (mantysą) równą zeru reprezentuje dodatnie lub ujemne zero, zależnie od bitu znaku. Jak wspomnieliśmy wcześniej, użyteczne jest dokładne reprezentowanie wartości 0.
- Wykładnik złożony z samych jedynek w połączeniu z frakcją równą zeru reprezentuje dodatnią lub ujemną nieskończoność, zależnie od bitu znaku. Możliwość istnienia reprezentacji nieskończoności (∞) jest również użyteczna. Pozostawia to użytkownikowi decyzję, czy traktować przepełnienie jako warunek błędu, czy też przenieść wartość ∞ i przetwarzać ją zgodnie z programem.
- Wykładnik równy zeru w połączeniu z niezerową frakcją reprezentuje liczbę zdenormalizowaną. W tym przypadku bit na lewo od przecinka binarne-

go jest zerem, a rzeczywisty wykładnik jest równy -126 lub -1022. Liczba jest dodatnia lub ujemna zależnie od bitu znaku.

- a Wykładnik złożony z samych jedynek w połączeniu z niezerową frakcją ma nadaną wartość NaN, która **nie oznacza liczby** (ang. *not a number*) i jest używana do sygnalizowania różnych warunków wyjątkowych.

Znaczeniem liczb zdenormalizowanych oraz NaN zajmiemy się w podrzdz. 8.5.

8.5

ARYTMETYKA ZMIENNOPOZYCYJNA

W tabeli 8.4 znajduje się podsumowanie podstawowych operacji arytmetyki zmiennopozycyjnej. W przypadku dodawania i odejmowania konieczne jest zapewnienie, żeby oba argumenty miały taki sam wykładnik. Może to wymagać przesunięcia przecinka pozycyjnego w jednym z argumentów. Mnożenie i dzielenie są pod tym względem prostsze.

TABELA 8.4. Liczby zmiennopozycyjne i operacje arytmetyczne

Liczby zmiennopozycyjne

$$x = x_s B^{x_E}$$

$$y = y_s B^{y_E}$$

Operacje arytmetyczne

$$x + y = (x_s B^{x_E - y_E} + y_s) \times B^{y_E} \quad j \quad x_E \leq y_E$$

$$x - y = (x_s B^{x_E - y_E} - y_s) \times B^{y_E} \quad j$$

$$x \times y = (x_s \times y_s) \times B^{x_E + y_E}$$

$$x \div y = (x_s \div y_s) \times B^{x_E - y_E}$$

Problemy mogą się pojawić jako wynik tych operacji. Należą do nich:

- a **Przepełnienie wykładnika.** Dodatni wykładnik przekracza maksymalną dopuszczalną wartość. W pewnych systemach może to być oznaczane jako $+\infty$ lub $-\infty$.
- a **Niedomiąg wykładnika.** Ujemny wykładnik przekracza maksymalną dopuszczalną wartość. Oznacza to, że liczba jest zbyt mała, aby mogła być reprezentowana. Może być traktowana jako 0.
- a **Niedomiąg mantysy.** W procesie wyrównywania mantys cyfry mogą „wypłynąć” poza prawy koniec mantysy. Wymagana jest pewna forma zaokrąglania, co jeszcze przedyskutujemy.
- a **Przepełnienie mantysy.** Wynikiem dodawania mantys o takim samym znaku może być wyprowadzenie najbardziej znaczącego bitu. Można to naprawić przez powtórne wyrównywanie, co jeszcze wyjaśnimy.

Dodawanie i odejmowanie

W arytmetyce zmiennopozycyjnej dodawanie i odejmowanie są bardziej złożone niż mnożenie i dzielenie. Jest tak z powodu konieczności wyrównywania. Są cztery podstawowe etapy algorytmu dodawania i odejmowania:

1. Sprawdzenie zer.
2. Wyrównanie mantys.
3. Dodanie lub odjęcie mantys.
4. Normalizowanie wyniku.

Typową sieć działań widać na rys. 8.21. Omawiając ją krok po kroku, wyjaśnimy główne działania wymagane do dodawania i odejmowania zmiennopozycyjnego. Zakładamy format podobny do tego z rys. 8.20. W przypadku operacji dodawania i odejmowania oba argumenty muszą być przeniesione do rejestrów, które będą używane przez ALU. Jeśli format zmiennopozycyjny przewiduje domyślny bit mantysy, to dla celów tych operacji musi on być ujawniony. Zwykle wykładniki i mantysy są przechowywane w odrębnych rejestrach i są łączone ponownie po uzyskaniu wyniku.

Ponieważ dodawanie i odejmowanie przebiegają identycznie z wyjątkiem zmiany znaku, proces rozpoczyna się od zmiany znaku odjemnika, jeśli jest to operacja odejmowania. Następnie, jeśli którykolwiek z argumentów jest zerem, pozostały jest przedstawiany jako wynik.

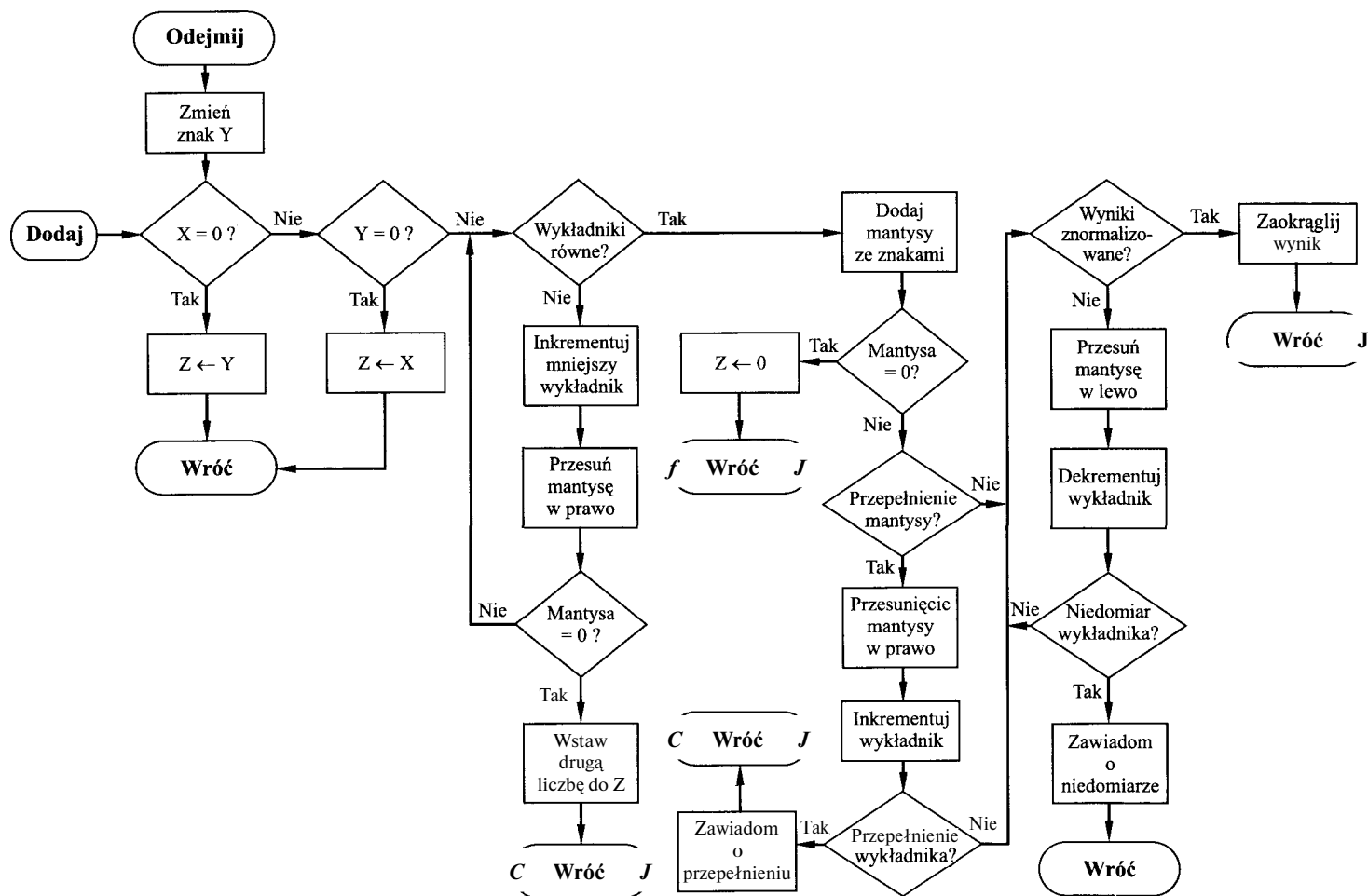
Następną fazą jest takie przekształcenie liczb, aby oba wykładniki były równe. Żeby dostrzec potrzebę tego przekształcenia, rozważmy następujące dodawanie dziesiętne:

$$123 \times 10^0 + 456 \times 10^{-2}$$

Jest oczywiste, że nie możemy po prostu dodać mantys. Cyfry muszą najpierw być ustawione w pozycjach równoważnych, to znaczy 4 z drugiej liczby musi być wyrównana z 3 z pierwszej. Wtedy oba wykładniki będą równe, co jest matematycznym warunkiem dodawania dwóch liczb w tej postaci. Wobec tego:

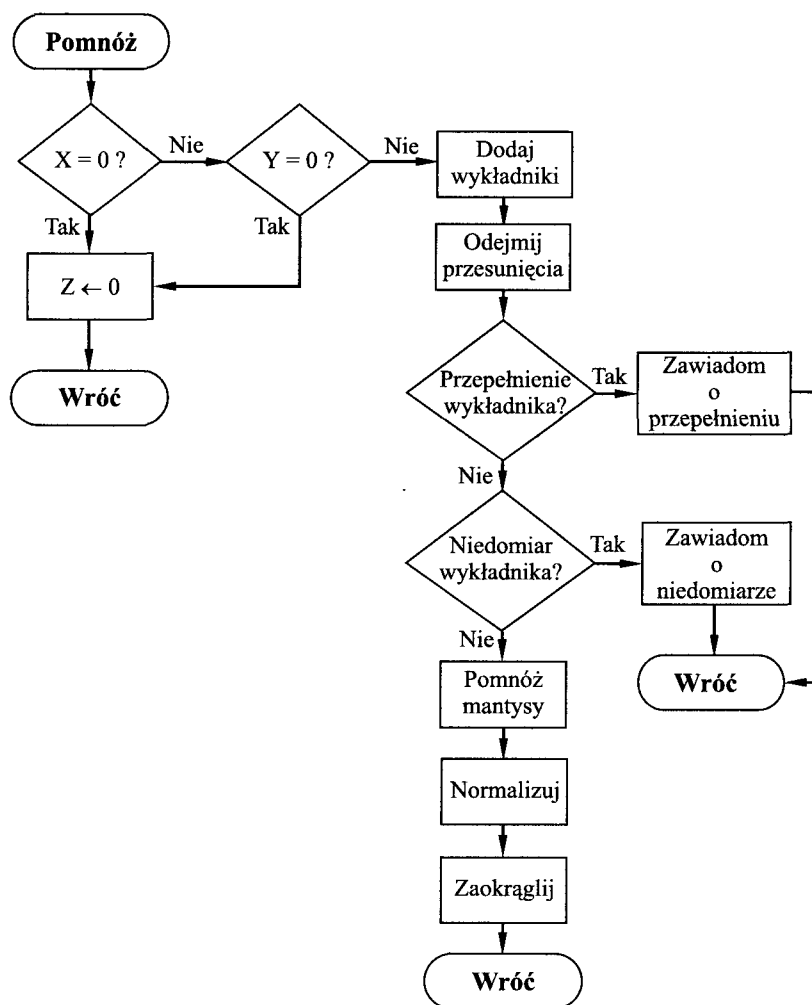
$$123 \times 10^0 + 456 \times 10^{-2} = 123 \times 10^0 + 4,56 \times 10^0 = 127,56 \times 10^0$$

Wyrównanie jest osiągane przez przesuwanie albo mniejszej liczby w prawo (zwiększanie jej wykładnika), albo większej liczby w lewo. Ponieważ wynikiem którejkolwiek z tych operacji może być utrata cyfr, przesuwa się raczej mniejszą liczbę; ewentualne stracone cyfry mają stosunkowo małe znaczenie. Wyrównanie jest osiągane przez powtarzające się przesuwanie mantysy o jedną cyfrę w prawo i odpowiednie zwiększanie wykładnika, aż do zrównania się wykładników. (Zauważmy, że jeśli podstawa jest równa 16, to przesunięcie o jedną cyfrę jest przesunięciem o 4 bity). Jeśli wynikiem procesu jest zerowa wartość mantysy, to pozostała liczba jest traktowana jako wynik. Jeśli więc obie liczby mają znacznie różniące się wykładniki, mniejsza liczba jest tracona.

RYSUNEK 8.21. Dodawanie i odejmowanie zmiennopozycyjne ($z \leftarrow x \pm y$)

Następnie obie mantysy są sumowane z uwzględnieniem znaków. Ponieważ znaki mogą się różnić, wynikiem może być 0. Istnieje również możliwość przepełnienia mantysy o jedną cyfrę. Jeśli tak się dzieje, mantysa wyniku jest przesuwana w prawo, a jego wykładnik - korygowany. Wynikiem może być również przepełnienie wykładnika; sytuacja taka jest zgłaszana, a operacja ulega *zatrzymaniu*.

Następną fazą jest normalizowanie wyniku. Normalizacja polega na przesuwaniu cyfr mantysy w lewo, aż najbardziej znacząca cyfra (bit lub 4 bity przy podstawie 16) nie będzie zerem. Każde przesunięcie powoduje zmniejszenie wykładnika i dlatego może być przyczyną niedomiaru wykładnika. Na zakończenie wynik musi być zaokrąglony i zgłoszony. Dyskusję zaokrąglania przeprowadzimy po omówieniu mnożenia i dzielenia.

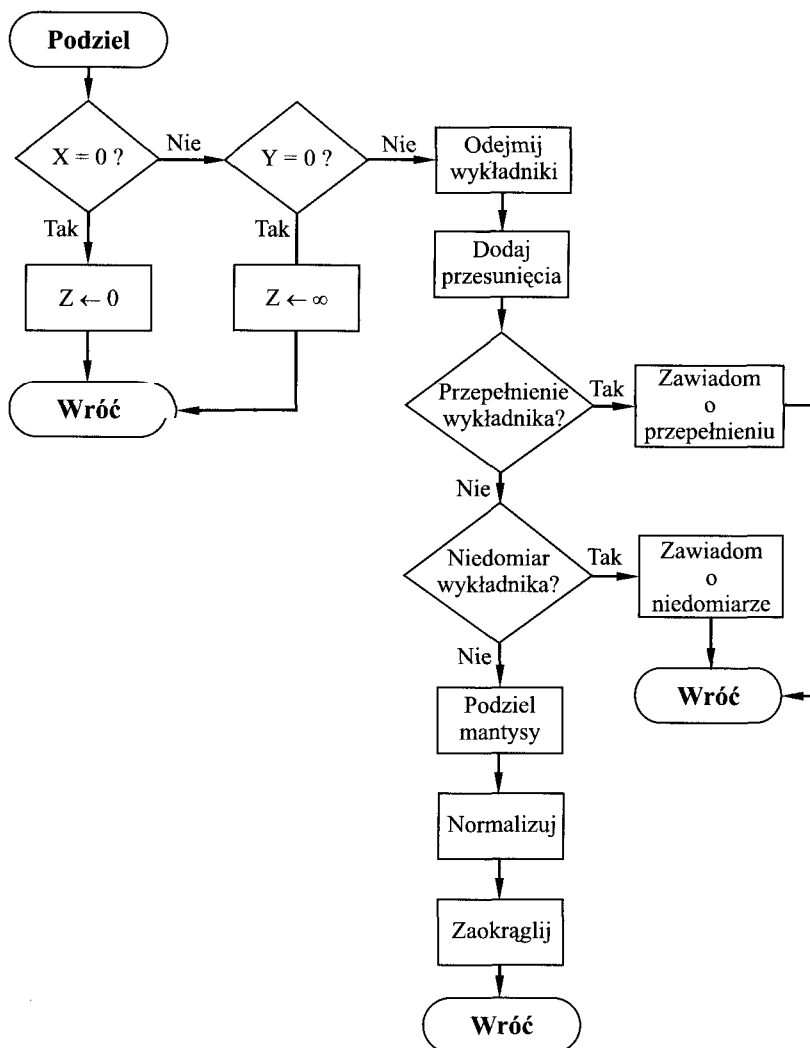


RYSUNEK 8.22. Mnożenie zmiennopozycyjne ($z \leftarrow x \times y$)

Mnożenie i dzielenie

Mnożenie i dzielenie zmiennopozycyjne są procesami o wiele prostszymi niż dodawanie i odejmowanie, co wyniknie z dalszej dyskusji.

Rozważmy mnożenie zilustrowane na rys. 8.22. Najpierw, jeśli którykolwiek z argumentów jest równy 0, to 0 jest zgłaszane jako wynik. Następnym krokiem jest dodanie wykładników. Jeśli wykładniki są przechowywane w postaci przesuniętej, to suma wykładników spowodowałaby podwojenie przesunięcia. Wobec tego wartość przesunięcia musi być odjęta od sumy. Wynikiem może być przepełnienie lub niedomiar wykładnika, co powinno być zgłoszone i co stanowi zakończenie algorytmu.



RYSUNEK 8.23. Dzielenie zmiennopozycyjne ($z \leftarrow x/y$)

Jeśli wykładnik iloczynu jest we właściwym zakresie, to następnym krokiem jest mnożenie mantys z uwzględnieniem ich znaków. Mnożenie jest realizowane w ten sam sposób, jak w przypadku liczb całkowitych. Mamy tu do czynienia z reprezentacją znak-moduł, jednak szczegóły są podobne do mnożenia liczb wyrażonych w notacji uzupełnienia do dwóch. Iloczyn ma długość dwukrotnie większą od długości mnożnej i mnożnika. Dodatkowe bity będą utracone podczas zaokrąglania.

Po obliczeniu iloczynu wynik jest normalizowany i zaokrąglany, podobnie jak przy dodawaniu i odejmowaniu. Zauważmy, że normalizacja może spowodować niedobór wykładnika.

Na zakończenie rozważmy sieć działań dla dzielenia pokazaną na rys. 8.23. Znów pierwszym krokiem jest sprawdzanie zer. Jeśli dzielnik jest równy 0, to jest zgłaszany błąd lub wynik jest ustalany jako nieskończoność, zależnie od implementacji. Dzielną równa 0 daje wynik 0. Następnie wykładnik dzielnika jest odejmowany od wykładnika dzielnej. Powoduje to usunięcie przesunięcia, które wobec tego musi być dodane. Wykonywane są testy w celu wykrycia przepełnienia lub niedomiaru wykładnika.

Następnym krokiem jest dzielenie mantys. Po tym następuje zwykła normalizacja i zaokrąglanie.

Analiza dokładności

Bity zabezpieczenia

Wspomnieliśmy, że przed operacją zmiennopozycyjną wykładnik i mantysa każdego argumentu są ładowane do rejestrów ALU. W przypadku mantysy długość rejestru jest prawie zawsze większa niż długość mantysy wraz z bitem domyślnym, jeśli taki występuje. Rejestr zawiera dodatkowe bity, zwane **bitami zabezpieczenia**, które są używane do wypełniania prawej strony mantysy zerami.

Przyczyna użycia bitów zabezpieczenia jest zilustrowana na rys. 8.24. Rozważmy liczbę w formacie IEEE, mającą 24-bitową mantysę zawierającą bit domyślny 1 na lewo od przecinka binarnego. Dwoma liczbami, których wartości są bardzo bliskie, są $X = 1,00 \dots 0 \times 2^1$ oraz $Y = 1,11 \dots 11 \times 2^0$. Jeśli mniejsza liczba ma być odjęta od większej, to musi ona być przesunięta o 1 bit w prawo w celu wyrównania wykładników. Widać to na rys. 8.24a. W tym procesie Y traci jeden bit znaczący; wynikiem jest 2^{-23} . Ta sama operacja jest powtórzona w części b rysunku, jednak w tym przypadku użyto bitów zabezpieczenia. Teraz najmniej znaczący bit nie jest tracony w wyniku wyrównywania, wynikiem zaś jest 2^{-24} , co oznacza 2-krotne zmniejszenie wartości w porównaniu z poprzednią odpowiedzią. Gdy podstawą jest 16, strata dokładności może być większa. Jak widać na rys. 8.24c i d, różnicą może być czynnik 16.

- (a) $x = 1.000 \dots 00 \times 2^1$
 $-y = 0.111 \dots 11 \times 2^1$
 $z = 0.000 \dots 01 \times 2^1$
 $= 1.000 \dots 00 \times 2^{-22}$
- (b) $x = 1.000 \dots 00 \ 0000 \times 2^1$
 $-y = 0.111 \dots 11 \ 1000 \times 2^1$
 $z = 0.000 \dots 00 \ 1000 \times 2^1$
 $= 1.000 \dots 00 \ 0000 \times 2^{-23}$
- (c) $x = .100000 \times 16^1$
 $-y = .0FFFFFF \times 16^1$
 $z = .000001 \times 16^1$
 $= .100000 \times 16^{-4}$
- (d) $x = .100000 \ 00 \times 16^1$
 $-y = .0FFFFFF \ 00 \times 16^1$
 $z = .000000 \ 10 \times 16^1$
 $= .100000 \ 00 \times 16^{-5}$

RYSUNEK 8.24. Zastosowanie bitów zabezpieczenia: (a) przykład binarny, bez bitów zabezpieczenia; (b) przykład binarny, z bitami zabezpieczenia; (c) przykład szesnastkowy, bez bitów zabezpieczenia; (d) przykład szesnastkowy, z bitami zabezpieczenia

Zaokrąglanie

Innym czynnikiem wpływającym na dokładność wyniku jest metoda zaokrąglania. Wynik jakiegokolwiek operacji na mantysach jest na ogół przechowywany w dłuższym rejestrze. Podczas sprowadzania wyniku do formatu zmiennopozycyjnego dodatkowe bity muszą być usunięte.

Zbadano wiele metod zaokrąglania. Norma IEEE wymienia cztery alternatywne podejścia:

- ☐ **Zaokrąglanie do najbliższej.** Wynik jest zaokrąglany do najbliższej reprezentowalnej liczby.
- ☐ **Zaokrąglanie w kierunku $+\infty$.** Wynik jest zaokrąglany w górę, w kierunku plus nieskończoności.
- ☐ **Zaokrąglanie w kierunku $-\infty$.** Wynik jest zaokrąglany w dół, w kierunku minus nieskończoności.
- a **Zaokrąglanie w kierunku 0.** Wynik jest zaokrąglany w kierunku zera.

Rozważmy kolejno każdą z tych metod. **Zaokrąglanie do najbliższej** jest wymienionym w normie trybem zaokrąglania opartym na zaniedbywaniu i jest definiowane następująco: jako zaokrąglenie powinna być przyjęta wartość reprezentowalna najbliższa nieskończonej dokładnemu wynikowi; jeśli dwie najbliższe wartości są równoodległe, powinna być przyjęta ta, której najmniej znaczący bit jest zerem.

Jeśli na przykład dodatkowe bity, poza 23 bitami możliwymi do przechowania, są równe 10010, to dodatkowe bity dają w wyniku więcej niż połowę ostatniej reprezentowalnej pozycji bitowej. W tym przypadku prawidłową odpowiedzią jest dodanie binarnej 1 do ostatniego reprezentowalnego bitu, co stanowi zaokrąglenie do następnej reprezentowalnej liczby. Przyjmijmy teraz, że dodatkowymi bitami są 01111. W tym przypadku dodatkowe bity stanowią mniej niż połowę ostatniej reprezentowalnej pozycji bitowej. Prawidłową odpowiedzią jest po prostu opuszczenie dodatkowych bitów (obcięcie), w wyniku czego otrzymuje się zaokrąglenie w dół do następnej reprezentowalnej liczby.

Norma odnosi się także do specjalnego przypadku dodatkowych bitów w postaci 10000 ... W tym przypadku wynik znajduje się dokładnie pośrodku między dwiema możliwymi reprezentowalnymi wartościami. Jedną z możliwych metod byłoby zawsze obcinanie i byłaby to operacja najprostsza. Jednak trudność z tym prostym podejściem polega na tym, że wprowadza ono niewielkie, lecz kumulujące się przesunięcie do sekwencji obliczeń. Co jest potrzebne, to metoda zaokrąglania pozbawiona przesunięcia. Jednym z możliwych podejść byłoby zaokrąglenie w górę lub w dół na zasadzie przypadku, w wyniku czego przeciętnie wynik nie byłby przesunięty. Argumentem przeciwko temu rozwiązaniu jest to, że nie prowadzi ono do przewidywalnych, deterministycznych wyników. Podejściem przyjętym w normie IEEE jest wymuszenie parzystego wyniku: jeśli wynik obliczeń leży dokładnie pośrodku między dwiema reprezentowalnymi liczbami, to wartość jest zaokrąglana w górę, gdy ostatnim reprezentowalnym bitem jest 1, lub jest pozostawiana bez zmiany, jeśli jest nim 0.

Dwie następne możliwości, **zaokrąglenie do plus i minus nieskończoności**, są użyteczne przy wdrażaniu metody znanej jako **arytmetyka przedziałów** (ang. *interval arithmetic*). Pomysł będący podstawą arytmetyki przedziałów jest następujący. Po zakończeniu sekwencji operacji zmiennopozycyjnych nie możemy znać dokładnej odpowiedzi ze względu na ograniczenia sprzętowe wprowadzające zaokrąglenie. Jeśli przeprowadzimy każde obliczenie w sekwencji dwukrotnie, za jednym razem zaokrąglając w górę, a za drugim w dół, w rezultacie otrzymamy górną i dolną granicę dokładnej odpowiedzi. Jeśli odstęp między górną a dolną granicą jest odpowiednio mały, to uzyskamy odpowiednio dokładny wynik. Jeśli nie, to przynajmniej o tym wiemy i możemy przeprowadzić dodatkową analizę.

Ostatnią metodą wymienioną w normie jest **zaokrąglenie w kierunku zera**. Jest to w rzeczywistości po prostu obcinanie: dodatkowe bity są ignorowane. Jest to z pewnością metoda najprostsza. Jednak jej rezultatem jest to, że wielkość obciętej liczby jest zawsze mniejsza lub równa dokładniejszej liczbie oryginalnej, co wprowadza ciągłe przesunięcie w dół. Jest to większe przesunięcie niż omawiane wyżej, ponieważ dotyczy ono każdej operacji mającej niezerowe bity dodatkowe.

Norma IEEE dotycząca binarnej arytmetyki zmiennopozycyjnej

Norma IEEE 754 wykracza poza prostą definicję formatu, określając specyficzne metody i procedury służące do tego, aby arytmetyka zmiennopozycyjna dawała jednolite i przewidywalne wyniki, niezależne od platformy sprzętowej. Jeden z aspektów już omówiliśmy, mianowicie zaokrąglanie. Przedmiotem tego punktu są trzy inne zagadnienia: nieskończoność, symbole NaN i liczby zde-normalizowane.

Nieskończoność

Arytmetyka nieskończoności jest traktowana jako przypadek graniczny *rzeczywistej* arytmetyki, przy czym wartościom nieskończoności jest nadana następująca interpretacja:

$$-\infty < (\text{każda skończona liczba}) < +\infty$$

Z wyjątkiem szczególnych przypadków przedyskutowanych poniżej, każda operacja arytmetyczna obejmująca nieskończoność daje oczywiste wyniki. Na przykład:

$$5 + (+\infty) = +\infty$$

$$5 - (+\infty) = -\infty$$

$$5 + (-\infty) = -\infty$$

$$5 - (-\infty) = +\infty$$

$$(+\infty) + (+\infty) = +\infty$$

$$(-\infty) + (-\infty) = -\infty$$

$$(-\infty) - (+\infty) = -\infty$$

$$(+\infty) - (-\infty) = +\infty$$

Ciche i sygnalizacyjne jednostki NaN

NaN jest symboliczną jednostką zakodowaną w formacie zmiennopozycyjnym, która może występować w dwóch rodzajach: cichym i sygnalizacyjnym. **Sygnalizacyjna** NaN służy do sygnalizowania wyjątków nieważnej operacji, gdy pojawia się ona jako jeden z argumentów. **Sygnalizacyjna** NaN nadaje wartość niezainicjowanym zmiennym oraz rozszerzeniom quasiarytmetycznym, które nie są przedmiotem normy. **Cicha** NaN propaguje się przez prawie wszystkie operacje arytmetyczne, nie sygnalizując wyjątku. W tabeli 8.5 są pokazane operacje powodujące powstanie cichych NaN.

Zauważmy, że oba rodzaje NaN mają taki sam format ogólny: wykładnik złożony z samych jedynek i niezerowa frakcja (mantysa). Wzór bitowy frakcji

TABELA 8.5. Operacje prowadzące do cichego NaN

Operacja	Ciche NaN wytworzone przez
Dowolna	dowolną operację na sygnalizacyjnym NaN
Dodawanie lub odejmowanie	operacje na nieskończonościach: $(+00) + (-\infty)$ $(-00) + (+\infty)$ $(+00) - (+\infty)$ $(-\infty) - (-\infty)$
Mnożenie	$0 \times \infty$
Dzielenie	$\frac{0}{-}$ lub $\frac{\infty}{0}$
Reszta	x reszta 0 lub ∞ reszta y
Pierwiastek kwadratowy	$\sqrt{x}$ gdzie $x < 0$

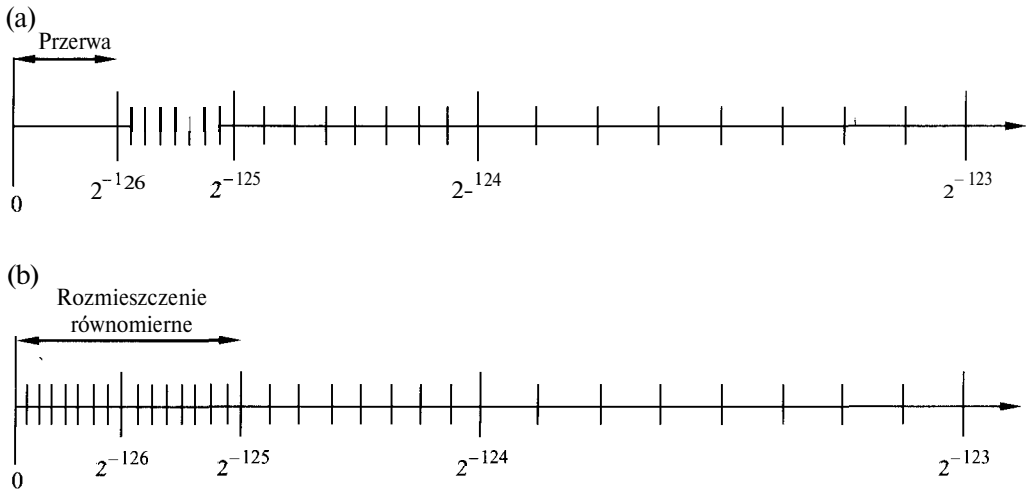
niezerowej jest zależny od implementacji; wartości frakcji mogą być używane do odróżniania cichych NaN od sygnalizacyjnych oraz do określania szczególnych warunków wyjątkowych.

Liczby zdenormalizowane

Liczby zdenormalizowane są włączone do normy IEEE 754 w celu wykorzystania w razie niedomiaru wykładnika. Gdy wykładnik wyniku jest zbyt mały (w przypadku ujemnego wykładnika ma zbyt dużą wielkość), wynik jest denormalizowany przez przesuwanie frakcji w prawo i zwiększanie wykładnika za każdym przesunięciem, aż wykładnik znajdzie się w zakresie reprezentowalnym.

Na rysunku 8.25 jest pokazany efekt wykorzystania liczb zdenormalizowanych. Liczby reprezentowalne mogą być pogrupowane w przedziały o postaci $[2^n, 2^{n+1}]$. Wewnątrz każdego takiego przedziału wykładniki pozostają stałe, a frakcje zmieniają się, tworząc jednolite rozmieszczenie reprezentowalnych liczb wewnątrz przedziału. Gdy zbliżamy się do zera, każdy kolejny przedział ma szerokość równą połowie szerokości poprzedniego przedziału, jednak zawiera taką samą ilość reprezentowalnych liczb. Wobec tego gęstość reprezentowalnych liczb wzrasta w miarę zbliżania się do zera. Jeżeli jednak używamy wyłącznie liczb znormalizowanych, istnieje przerwa między najmniejszą liczbą znormalizowaną a zerem. W przypadku 32-bitowego formatu IEEE 754 w każdym przedziale istnieje 2^{32} reprezentowalnych liczb, a najmniejszą reprezentowalną liczbą dodatnią jest 2^{-126} . Dzięki dodaniu liczb zdenormalizowanych w przedziale między 0 a 2^{-126} uzyskuje się dodatkowe, równomiernie rozmieszczone 2^{23} liczb.

Stosowanie liczb zdenormalizowanych jest określane jako **stopniowy niedomiar** (ang. *gradual underflow*)[COON81]. Bez liczb zdenormalizowanych



RYSUNEK 8.25. Wpływ liczb zdenormalizowanych IEEE 754: (a) format 32-bitowy bez liczb zdenormalizowanych; (b) format 32-bitowy z liczbami zdenormalizowanymi

przerwa między najmniejszą reprezentowalną liczbą niezerową a zerem jest o wiele szersza niż przerwa między najmniejszą reprezentowalną liczbą niezerową a następną większą liczbą. Stopniowy niedomiar powoduje wypełnienie tej przerwy i zredukowanie wpływu niedomiaru wykładnika do poziomu porównywalnego z zaokrągleniem liczb znormalizowanych.

8.6

POLECANA LITERATURA

Studenci poważnie interesujący się arytmetyką komputerową bezwzględnie powinni zapoznać się z dwoma tomami pracy [SWAR90]. Tom I został po raz pierwszy opublikowany w roku 1980 i zawiera kluczowe artykuły (niektóre z nich trudne do osiągnięcia inną drogą) dotyczące podstaw arytmetyki komputerowej. Tom II zawiera nowsze artykuły, obejmujące aspekty teoretyczne, projektowe i wdrożeniowe arytmetyki komputerowej. Dwa inne wartościowe zbiory artykułów stanowią wydania *IEEE Transactions on Computers* z sierpnia 1990 i z sierpnia 1994 roku. Do dobrych książek na temat arytmetyki komputerowej należą [OMON94], [KORE93] i [MORG92].

Na temat arytmetyki zmiennopozycyjnej istnieje właściwie zatytułowana pozycja [GOLD91]: *Co każdy naukowiec komputerowy powinien wiedzieć o arytmetyce zmiennopozycyjnej*. Bardziej złożony opis można znaleźć w [WALL90]. Inne doskonałe ujęcie tego tematu zawiera [KNUT81].

GOLD91 Goldberg D.: „What Every Computer Scientist Should Know About Floating-Point Arithmetic”. *ACM Computing Surveys*, March 1991.

- KNUT81 Knuth D.: *The Art of Computer Programming. Vol. 2: Seminumerical Algorithms*. Reading, AddisonWesley, 1981.
- KORE93 Koren I.: *Computer Arithmetic Algorithms*. Englewood Cliffs, Prentice Hall, 1993.
- MORG92 Morgan D.: *Numerical Methods*. San Mateo, M&T Books, 1992.
- OMON94 Omondi A.: *Computer Arithmetic Systems: Algorithms, Architecture, and Implementations*. Englewood Cliffs, Prentice Hall, 1994.
- SWAR90 Swartzlander E., red.: *Computer Arithmetic. Vol. I i II*. Los Alamitos, IEEE Computer Society Press, 1990.
- WALL90 Wallis P.: *Improving Floating-Point Programming*. New York, Wiley, 1992.

87

PROBLEMY DO ROZWIĄZANIA

- 8.1. Oblicz następujące różnice, stosując uzupełnienie do dwóch:
- | | | | |
|----------------|----------------|----------------------|------------------|
| (a) 111000 | (b) 11001100 | (c) 111100001111 | (d) 11000011 |
| <u>-110011</u> | <u>-101110</u> | <u>-110011110011</u> | <u>-11101000</u> |
- 8.2. Czy właściwa jest następująca alternatywna definicja przepełnienia w arytmetyce uzupełnienia do dwóch?
- Jeśli LUB wykluczające bitów przeniesienia: wchodzącego do najbardziej na lewo położonej kolumny i opuszczającego ją, jest równe 1, to występuje warunek przepełnienia. W przeciwnym razie nie występuje.
- 8.3. Porównaj rysunek 8.8 i 8.11. Dlaczego na tym ostatnim nie jest używany bit C?
- 8.4. Nakreśl schemat blokowy ilustrujący proces dzielenia podobny do rys. 8.7.
- 8.5. Algorytm dzielenia pokazany na rys. 8.15 jest znany jako metoda przywracająca, ponieważ wartość rejestru A musi być przywrócona po nieudanym odejmowaniu. Nieco bardziej złożone podejście, znane jako nieprzywracające, zapobiega niekoniecznemu odejmowaniu i dodawaniu. Zaproponuj algorytm dla tego drugiego podejścia.
- 8.6. W arytmetyce liczb całkowitych iloraz J/K dwóch liczb całkowitych J i K jest mniejszy lub równy zwykłemu ilorazowi. Czy twierdzenie to jest prawdziwe?
- 8.7. Udowodnij, że mnożenie dwóch n -cyfrowych liczb o podstawie B daje iloczyn o liczbie cyfr nie przekraczającej $2n$.
- 8.8. Porównaj silne i słabe strony reprezentacji: przesuniętej, znak-moduł i uzupełnienia do dwóch.
- 8.9. Podziel -145 przez 13 w notacji binarnej uzupełnienia do dwóch, posługując się słowem 12-bitowym. Zastosuj algorytm opisany w p. 8.3.
- 8.10. Załóż, że wykładnik e jest ograniczony do zakresu $0 < e < X$, przy przesunięciu q , podstawie b i formacie o długości p cyfr.
- (a) Jakie są największe i najmniejsze wartości dodatnie, które mogą być zapisane?

- (b) Jakie są największe i najmniejsze wartości dodatnie, które mogą być zapisane jako znormalizowane liczby zmiennopozycyjne?
- 8.11. Wyraż następujące liczby w 32-bitowym formacie zmiennopozycyjnym IEEE:
- (a) -5 (d) 384
 (b) -6 (e) 1/16
 (c) -1,5 (f) -1/32
- 8.12. Wyraż następujące liczby w 32-bitowym formacie zmiennopozycyjnym IBM, w którym jest stosowany 7-bitowy wykładnik i domyślna podstawa 16:
- (a) 1,0 (e) -15,0
 (b) 0,5 (f) $5,4 \times 10^{-79}$
 (c) 1/64 (g) $7,2 \times 10^{75}$
 (d) 0,0
- 8.13. Jaka powinna być wartość przesunięcia w przypadku:
- (a) wykładnika podstawy 2 ($B = 2$) w polu 6-bitowym?
 (b) wykładnika podstawy 8 ($B = 8$) w polu 7-bitowym?
- 8.14. Wykreśl oś liczbową podobną do tej z rys. 8.18b, dotyczącą formatów zmiennopozycyjnych z rys. 8.20.
- 8.15. Gdy ludzie mówią o niedokładności w arytmetyce zmiennopozycyjnej, opisują często błędy wynikające z kasowania, które ma miejsce podczas odejmowania prawie równych wielkości. Jednak gdy X i Y są w przybliżeniu równe, różnica $X - Y$ jest osiągana dokładnie, bez błędu. Co ci ludzie mają w rzeczywistości na myśli?
- 8.16. Jakakolwiek notacja zmiennopozycyjna stosowana w komputerze może reprezentować dokładnie tylko pewne liczby rzeczywiste; wszystkie inne muszą być aproksymowane. Jeśli A' jest przechowywaną wartością aproksymującą rzeczywistą wartość A , to błąd względny r jest wyrażany jako

$$r = \frac{A - A'}{A}$$

Przedstaw wielkość dziesiętną +0,4 w następującym formacie zmiennopozycyjnym:

- a podstawa: 2,
 □ wykładnik: przesunięty, 4-bitowy,
 a mantysa: 7-bitowa.

Ile wynosi błąd względny?

- 8.17. Wartości numeryczne A i B są przechowywane w komputerze jako przybliżenia A' i B' . Zaniedbując jakiekolwiek inne błędy wynikające z obcięcia lub zaokrąglania, wykaż, że względny błąd iloczynu jest w przybliżeniu sumą błędów względnych obu czynników.

- 8.18. Jeśli $A = 1,427$, oblicz błąd względny przy A obciętym do 1,42 oraz przy A zaokrąglonym do 1,43.
- 8.19. Jeden z najpoważniejszych błędów w obliczeniach komputerowych występuje, gdy są odejmowane dwie prawie równe liczby. Przyjmij $A = 0,22288$ i $B = 0,22211$. Komputer obcina wszystkie wartości do czterech cyfr dziesiętnych. Wobec tego $A' = 0,2228$ oraz $B' = 0,2221$.
- (a) Jakie są błędy względne A' i B' ?
- (b) Jaki jest błąd względny $C' = A' - B'$?

DODATEK 8A

Systemy liczbowe

8A.1. System dziesiętny

Możemy sobie z tego nie zdawać sprawy, jednak w codziennym życiu stosujemy w celu reprezentowania liczb system oparty na cyfrach. Używamy cyfr dziesiętnych (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), a system określamy jako **dziesiętny**.

Zastanówmy się, co oznacza liczba 83. Zapis ten należy rozumieć jako 8 dziesiątek plus 3:

$$83 = 8 \times 10 + 3$$

Liczba 4728 oznacza 4 tysiące, 7 setek, 2 dziesiątki plus 8:

$$4728 = 4 \times 1000 + 7 \times 100 + 2 \times 10 + 8$$

Mówi się, że system dziesiętny ma podstawę 10. Oznacza to, że każda cyfra w liczbie jest mnożona przez 10 do potęgi odpowiadającej pozycji tej cyfry. Wobec tego

$$83 = 8 \times 10^1 + 3 \times 10^0$$

$$4728 = 4 \times 10^3 + 7 \times 10^2 + 2 \times 10^1 + 8 \times 10^0$$

Wartości ułamkowe są reprezentowane w ten sam sposób. Wobec tego

$$472,83 = 4 \times 10^2 + 7 \times 10^1 + 2 \times 10^0 + 8 \times 10^{-1} + 3 \times 10^{-2}$$

Ogólnie, dziesiętna reprezentacja $X = \{ \dots x_2 x_1 x_0, x_{-1} x_{-2} x_{-3} \dots \}$, a wartość X można zapisać wzorem: $X = \sum_i x_i 10^i$.

8A.2. System binarny

W systemie dziesiętnym używa się dziesięciu różnych cyfr do reprezentowania liczb o podstawie 10. W systemie binarnym dysponujemy tylko dwiema cyframi, 1 i 0. Wobec tego liczby w systemie binarnym są reprezentowane przy podstawie 2.

Aby zapobiec nieporozumieniom, stosujemy czasem przy liczbach indeksy wskazujące ich podstawę. Na przykład, 83_{10} i 4728_{10} są liczbami reprezentowanymi w notacji dziesiętnej lub po prostu dziesiętnymi. W notacji binarnej 1 i 0 mają to samo znaczenie, co w notacji dziesiętnej:

$$0_2 = 0_{10}$$

$$1_2 = 1_{10}$$

Jak reprezentujemy większe liczby? Podobnie jak w przypadku notacji dziesiętnej, każda cyfra w liczbie binarnej ma wartość zależną od jej pozycji:

$$10_2 = 1 \times 2^1 + 0 \times 2^0 = 2_{10}$$

$$11_2 = 1 \times 2^1 + 1 \times 2^0 = 3_{10}$$

$$100_2 = 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 4_{10}$$

i tak dalej. W tabeli 8.6 pokazano pierwsze 64 8-bitowe liczby binarne wraz z ich równoważnikami dziesiętnymi (w tabeli są również podane liczby w nota-

TABELA 8.6. Liczby binarne oraz ich równoważniki dziesiętne i szesnastkowe

Binarna	Dziesiętna	Szesnastkowa	Binarna	Dziesiętna	Szesnastkowa
00000000	0	0	00100000	32	20
00000001	1	1	00100001	33	21
00000010	2	2	00100010	34	22
00000011	3	3	00100011	35	23
00000100	4	4	00100100	36	24
00000101	5	5	00100101	37	25
00000110	6	6	00100110	38	26
00000111	7	7	00100111	39	27
00001000	8	8	00101000	40	28
00001001	9	9	00101001	41	29
00001010	10	A	00101010	42	2A
00001011	11	B	00101011	43	2B
00001100	12	C	00101100	44	2C
00001101	13	D	00101101	45	2D
00001110	14	E	00101110	46	2E
00001111	15	F	00101111	47	2F
00010000	16	10	00110000	48	30
00010001	17	11	00110001	49	31
00010010	18	12	00110010	50	32
00010011	19	13	00110011	51	33
00010100	20	14	00110100	52	34
00010101	21	15	00110101	53	35
00010110	22	16	00110110	54	36
00010111	23	17	00110111	55	37
00011000	24	18	00111000	56	38
00011001	25	19	00111001	57	39
00011010	26	1A	00111010	58	3A
00011011	27	1B	00111011	59	3B
00011100	28	1C	00111100	60	3C
00011101	29	1D	00111101	61	3D
00011110	30	1E	00111110	62	3E
00011111	31	1F	00111111	63	3F

cji szesnastkowej, która jest objaśniona dalej). Jak poprzednio, wartości ułamkowe są reprezentowane za pomocą ujemnych potęg podstawy:

$$1001,101 \approx 2^3 + 2^0 + 2^{-1} + 2^{-3}.$$

8A.3. Konwersja liczb binarnych na dziesiętne i na odwrot

Konwersja liczby z notacji binarnej na dziesiętną jest prosta. W poprzednim punkcie pokazaliśmy kilka przykładów takiej konwersji. Wszystko co jest potrzebne, to pomnożenie każdej cyfry binarnej przez odpowiednią potęgę 2 i dodanie wyników.

W przypadku konwersji z notacji dziesiętnej na binarną część całkowita i część ułamkowa są traktowane oddzielnie. Załóżmy, że jest wymagana konwersja dziesiętnej liczby całkowitej N na postać binarną. Jeśli podzielimy N przez 2 (w systemie dziesiętnym) i otrzymamy iloraz N_1 i resztę r_1 , to możemy zapisać, że

$$N = 2 \times N_1 + r_1 \quad r_1 = 0 \text{ lub } 1$$

Następnie dzielimy iloraz N_1 przez 2. Nowy iloraz niech wynosi N_2 , a nowa reszta - r_2 . Wobec tego

$$N_1 = 2 \times N_2 + r_2 \quad r_2 = 0 \text{ lub } 1$$

stąd

$$N = 2(2N_2 + r_2) + r_1 = 2^2 N_2 + r_2 \times 2^1 + r_1 \times 2^0$$

Jeśli następnie

$$N_2 = 2N_3 + r_3$$

to mamy

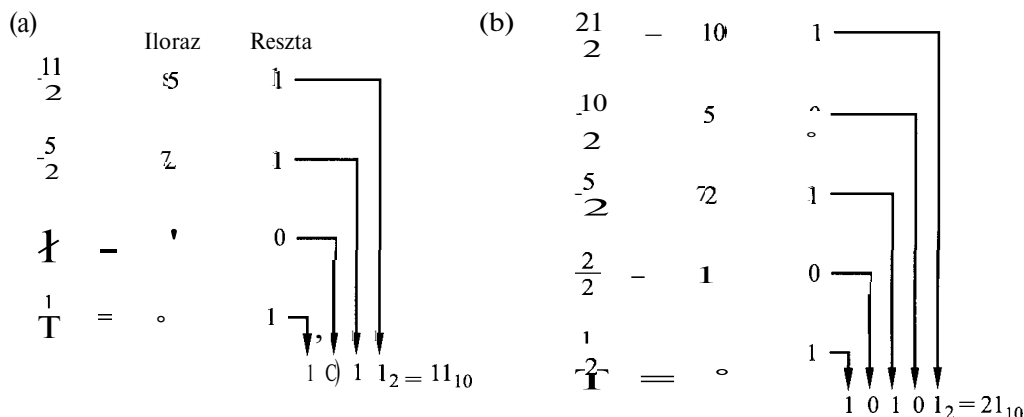
$$N = 2^3 N_3 + r_3 \times 2^2 + r_2 \times 2^1 + r_1 \times 2^0$$

Kontynuując to, ze względu na $N > N_1 > N_2 \dots$, możemy ewentualnie otrzymać iloraz $N_k = 1$ oraz resztę r_k , która jest równa 0 lub 1. Wobec tego

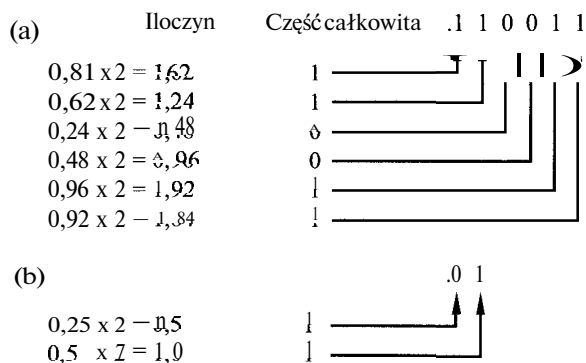
$$N = 1 \times 2^k + r_k \times 2^{k-1} + \dots + r_3 \times 2^2 + r_2 \times 2^1 + r_1 \times 2^0$$

Oznacza to, że konwersji z podstawy 10 do podstawy 2 dokonuje się, powtarzając dzielenie przez 2. Kolejne reszty oraz ostatni iloraz (N_1) dają nam cyfry binarne N w kolejności rosnącego znaczenia. Na rysunku 8.26 są pokazane dwa przykłady.

Konwersja części ułamkowej polega na powtarzaniu mnożenia przez 2, co widać na rys. 8.27. W kolejnych krokach ułamkowa część liczby dziesiętnej jest mnożona przez 2. Cyfra znajdująca się na lewo od przecinka w iloczynie jest równa 0 lub 1 i wchodzi w skład reprezentacji binarnej, począwszy od najbardziej znaczącego bitu. Część ułamkowa iloczynu jest używana jako mnożna w następnym kroku. Aby zobaczyć, jak to działa, weźmy dodatni ułamek dziesiętny $F < 1$. Możemy wyrazić F jako



RYСУNEK 8.26. Przykłady konwersji liczb całkowitych z notacji dziesiętnej na binarną: (a) 11_{10} ; (b) 21_{10}



RYСУNEK 8.27. Przykłady konwersji liczb ułamkowych z notacji dziesiętnej na binarną: (a) $0,81_{10} \approx 0,110011_2$ (w przybliżeniu); (b) $0,25_{10} = 0,01_2$ (dokładnie)

$$F = a_{-1} \times 2^{-1} + a_{-2} \times 2^{-2} + a_{-3} \times 2^{-3} + \dots$$

gdzie każde a_i jest równe 0 lub 1. Jeśli pomnożymy to przez 2, otrzymamy

$$2F = a_{-1} + a_{-2} \times 2^{-1} + a_{-3} \times 2^{-2} + \dots$$

Części całkowite obu stron muszą być równe. Wobec tego całkowita część $2F$, która musi być albo 0, albo 1 (ponieważ $0 < F < 1$), jest równa po prostu a_{-1} . Stąd $2F = a_{-1} + F_1$, gdzie $0 < F_1 < 1$, oraz

$$F = a_{-2} \times 2^{-1} + a_{-3} \times 2^{-2} + \dots$$

W celu znalezienia a_{-2} powtarzamy proces. Proces ten niekoniecznie jest dokładny; chodzi o to, że ułamek dziesiętny o skończonej liczbie cyfr może wymagać ułamka binarnego o nieskończonej liczbie cyfr. W takich przypadkach algorytm konwersji jest zwykle zatrzymywany po wstępnie ustalonej liczbie kroków, zależnie od wymaganej dokładności.

8A.4. Notacja szesnastkowa

Ze względu na nieodłączną, binarną naturę zespołów komputera cyfrowego, wszystkie formy danych wewnątrz komputera są reprezentowane za pomocą różnych kodów binarnych. Widzieliśmy przykłady kodów binarnych dla tekstu i notacji binarnej dla liczb. Później zobaczymy przykłady użycia kodów binarnych dla innych rodzajów danych. Niezależnie jednak od tego, jak wygodny jest system binarny z punktu widzenia komputerów, jest on zbyt nieporęczny z punktu widzenia ludzi. W związku z tym wielu specjalistów, którzy muszą spędzać czas, pracując z danymi pierwotnymi w komputerach, woli bardziej zwartą notację.

Jakiej użyć notacji? Jedną z możliwości jest notacja dziesiętna. Widzieliśmy (tabela 8.6), że każda liczba binarna ma swój dziesiętny równoważnik. Notacja ta jest bardziej zwarta niż binarna, jednak jest kłopotliwa z powodu żmudnego procesu konwersji między podstawą 2 a podstawą 10.

Zamiast tego przyjęto notację **szesnastkowa**. Cyfry binarne są grupowane w zespoły po 4. Każda możliwa kombinacja 4 cyfr binarnych otrzymuje następujący symbol:

0000 = 0	1000 = 8
0001 = 1	1001 = 9
0010 = 2	1010 = A
0011 = 3	1011 = B
0100 = 4	1100 = C
0101 = 5	1101 = D
0110 = 6	1110 = E
0111 = 7	1111 = F

Ponieważ użyto 16 symboli, notacja została nazwana szesnastkową, a 16 symboli - cyframi szesnastkowymi.

Ciąg cyfr szesnastkowych może być traktowany jako reprezentacja liczby całkowitej o podstawie 16. Wobec tego

$$1A_{16} = 1_{16} \times 16^1 + A_{16} \times 16^0 = 1_{10} \times 16^1 + 10_{10} \times 16^0 = 26$$

Jednak notacja szesnastkowa nie jest używana do reprezentowania liczb całkowitych. Jest stosowana jako zwięzła notacja do reprezentowania dowolnego ciągu cyfr binarnych, niezależnie od tego, czy reprezentują one tekst, liczby czy inny rodzaj danych. Przyczyny stosowania notacji szesnastkowej są następujące:

1. Jest bardziej zwarta niż notacja binarna.
2. W większości komputerów dane binarne zajmują pewne wielokrotności 4 bitów, co oznacza wielokrotność pojedynczych cyfr szesnastkowych.
3. Konwersja między notacją binarną a szesnastkową jest bardzo łatwa.

Jako przykład dotyczący ostatniego punktu rozważmy ciąg binarny 110111100001. Jest on równoważny następującemu zapisowi szesnastkowemu:

$$\begin{array}{ccc} 1101 & 1110 & 0001 = DE1_{16} \\ D & E & 1 \end{array}$$

Proces ten jest przeprowadzany tak naturalnie, że doświadczony programista może w myśli przetwarzać reprezentacje binarne na szesnastkowe bez konieczności pisania. Jest całkiem możliwe, że czytelnik nigdy nie będzie potrzebował tej szczególnej umiejętności. Jednak, ponieważ może on spotkać się z notacją szesnastkową, włączono ją do tej książki.

Listy rozkazów: własności i funkcje

Wielu obiektów opisanych w tej książce nie widzi użytkownik lub programista komputera. Jeśli programista używa języka wysokiego poziomu, takiego jak Pascal lub Ada, niewiele dostrzega on z architektury wykorzystywanej maszyny.

Jednym z obszarów, w których projektant komputera i jego programista mają wspólny obraz maszyny, jest lista rozkazów maszynowych. Z punktu widzenia projektanta, lista rozkazów maszynowych określa wymagania funkcjonalne w stosunku do procesora: implementacja procesora jest zadaniem, które w znacznej mierze polega na wdrożeniu listy rozkazów maszynowych. Z punktu widzenia użytkownika, tylko ten użytkownik, który wybiera programowanie w języku maszyny (w istocie w języku asemblerowym; patrz podrozdz. 9.4.), uświadamia sobie strukturę rejestrów i pamięci, rodzaje danych bezpośrednio obsługiwanych przez maszynę oraz funkcjonowanie ALU.

Opis listy rozkazów maszynowych komputera oznacza duży krok naprzód w wyjaśnianiu działania procesora. Dlatego w tym i w następnym rozdziale skupiamy się na rozkazach maszynowych, po czym przechodzimy do analizowania struktury i działania procesora.

9.1

WŁASNOŚCI ROZKAZÓW MASZYNOWYCH

Działanie procesora jest określone przez rozkazy, które on wykonuje. Rozkazy te są nazywane **rozkazami maszynowymi lub rozkazami komputerowymi**. Procesor może realizować wiele funkcji, co znajduje odbicie w różnorodności definiowanych dla niego rozkazów. Zbiór różnych rozkazów, które może wykonywać procesor, jest określany jako **lista rozkazów**.

Elementy rozkazu maszynowego

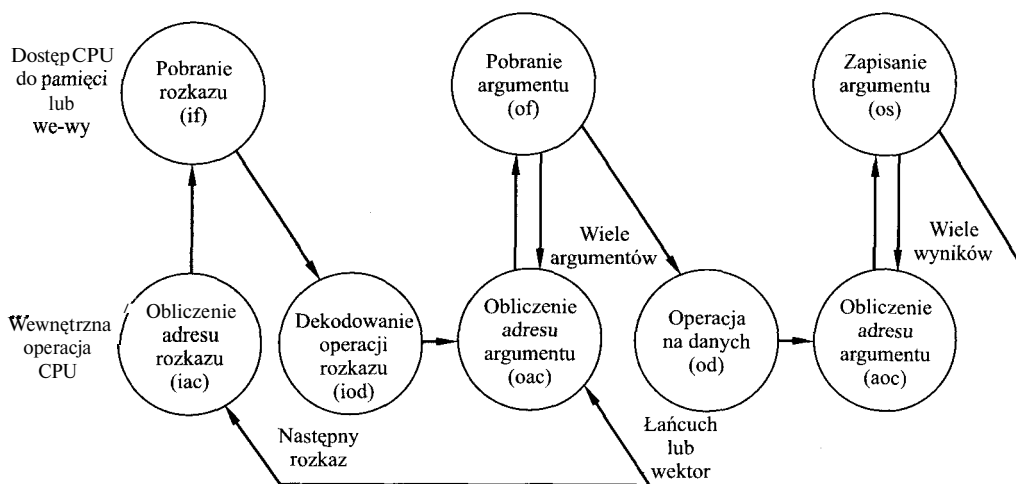
Każdy rozkaz musi zawierać informacje wymagane przez procesor do jego wykonania. Na rysunku 9.1, będącym powtórzeniem rys. 3.6, są pokazane etapy wykonywania rozkazów. Przez implikację, etapy te definiują elementy rozkazu maszynowego. Elementami tymi są:

- Q Kod operacji.** Określa operację, jaka ma być przeprowadzona (np. ADD, I/O). Operacja jest precyzowana za pomocą kodu binarnego - kodu operacji.
- a Odniesienie do argumentów źródłowych.** Operacja może obejmować jeden lub wiele argumentów źródłowych; są one danymi wejściowymi operacji.
- Q Odniesienie do wyniku.** Operacja może prowadzić do powstania wyniku.
- a Odniesienie do następnego rozkazu.** Mówi ono procesorowi, skąd ma on pobrać następny rozkaz po zakończeniu wykonywania bieżącego rozkazu.

Następny rozkaz przewidziany do pobrania jest umieszczony w pamięci głównej lub (w przypadku systemu pamięci wirtualnej) albo w pamięci głównej, albo w pomocniczej (dyskowej). W większości przypadków następny rozkaz przewidziany do pobrania następuje bezpośrednio po bieżącym rozkazie. Nie występuje wówczas jawne odniesienie do następnego rozkazu. Gdy odniesienie to jest potrzebne, musi być dostarczony adres w pamięci głównej lub wirtualnej. Formę, w której jest dostarczany adres, opiszemy w rozdz. 10.

Argumenty źródłowe i wyniki mogą się znajdować w jednym z trzech obszarów. Te obszary to:

- *Pamięć główna lub wirtualna.* Podobnie jak w przypadku odniesienia do następnego rozkazu, musi być dostarczony adres pamięci.



RYSUNEK 9.1. Graf stanów cyklu rozkazu

- a *Rejestr procesora.* Z rzadkimi wyjątkami procesor zawiera jeden lub wiele rejestrów, do których mogą się odnosić rozkazy maszynowe. Jeśli istnieje tylko jeden rejestr, odniesienie to może być domyślne. Jeśli istnieje więcej rejestrów niż jeden, to każdy rejestr ma przypisany unikatowy numer i rozkaz musi zawierać numer pożądanego rejestru.
- u *Urządzenia wejścia-wyjścia.* Rozkaz musi określać moduł wejścia-wyjścia oraz urządzenie wejścia-wyjścia używane w operacji. Jeśli używane jest wejście-wyjście odwzorowane w pamięci, jest to jeszcze jeden adres pamięci głównej lub wirtualnej.

Reprezentacja rozkazu

Wewnątrz komputera każdy rozkaz jest reprezentowany za pomocą ciągu bitów. Rozkaz jest dzielony na pola odpowiadające elementom składowym rozkazu. Ten obraz rozkazu nazywa się **formatem rozkazu**. Prosty przykład jest pokazany na rys. 9.2. Innym przykładem jest format rozkazu IAS pokazany na rys. 2.2. W większości list rozkazów używa się więcej niż jednego formatu. Rozkaz, podczas wykonywania, jest wczytywany do rejestru rozkazów (IR) w procesorze. Procesor musi być zdolny do pobrania danych z różnych pól rozkazu w celu wykonania wymaganej operacji.

Zarówno programiście, jak i czytelnikowi książek trudno jest posługiwać się binarną reprezentacją rozkazów maszynowych. Dlatego powszechną praktyką stało się używanie **symbolicznej reprezentacji** rozkazów maszynowych. Przykład takiej reprezentacji był użyty w stosunku do listy rozkazów IAS w tabeli 2.1.

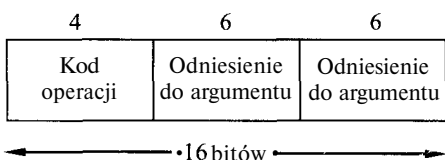
Kody operacji są zapisywane za pomocą skrótów zwanych **mnemonikami**. Do powszechnie znanych przykładów należą:

ADD	Dodaj (<i>add</i>)
SUB	Odejmij (<i>subtract</i>)
MPY	Pomnóż (<i>multiply</i>)
LOAD	Ładuj dane z pamięci (<i>load data from memory</i>)
STOR	Zapisz dane w pamięci (<i>store data to memory</i>).

Argumenty są również reprezentowane symbolicznie. Na przykład rozkaz

ADD R, Y

może oznaczać: dodaj wartość zawartą w pozycji danych Y do zawartości rejestru R. W tym przykładzie Y odnosi się do adresu komórki pamięci, R zaś ozna-



RYSUNEK 9.2. Prosty format rozkazu

cza określony rejestr. Zauważmy, że operacja jest przeprowadzana na zawartościach tych lokacji, a nie na adresach.

Jest więc możliwe napisanie programu w języku maszynowym w postaci symbolicznej. Każdy symboliczny kod operacji ma ustaloną reprezentację binarną, a programista określa lokalizację każdego symbolicznego argumentu. Programista może na przykład rozpocząć od listy definicji:

$$X = 513$$
$$Y = 514$$

i tak dalej. Prosty program akceptuje te symboliczne dane wejściowe, przekształca kody operacji i odniesienia do argumentów na postać binarną, po czym buduje binarne rozkazy maszynowe.

Osoby programujące w języku maszynowym są rzadkością. Większość dzisiejszych programów pisze się w języku wysokiego poziomu lub w języku assemblerowym, który będzie omówiony na końcu tego rozdziału. Jednak symboliczny język maszynowy pozostaje narzędziem użytecznym do opisanie rozkazów maszynowych i do tego właśnie celu będziemy go używać.

Rodzaje rozkazów

Rozważmy rozkaz w języku wysokiego poziomu, który może być wyrażony w takim języku, jak BASIC czy FORTRAN. Na przykład

$$X = X + Y$$

Instrukcja ta zaleca komputerowi dodanie wartości przechowywanej w Y do wartości przechowywanej w X oraz umieszczenia wyniku w X. Jak może to być wykonane za pomocą rozkazów maszynowych? Załóżmy, że zmienne X i Y odpowiadają lokacjom 513 i 514. Jeśli przyjmiemy prostą listę rozkazów maszynowych, to operacja ta może być wykonana za pomocą trzech rozkazów:

1. Załaduj do rejestru zawartość komórki pamięci 513.
2. Dodaj zawartość komórki pamięci 514 do rejestru.
3. Przechowaj zawartość rejestru w komórce pamięci 513.

Jak można zauważyć, pojedyncza instrukcja w języku BASIC może wymagać trzech rozkazów maszynowych. Jest to typowa zależność między językiem wysokiego poziomu a językiem maszynowym. Język wysokiego poziomu wyraża operacje w zwartej formie algebraicznej, używając zmiennych. Język maszynowy wyraża operacje w postaci podstawowej, obejmując ruch danych do (lub z) rejestrów.

Posługując się tym prostym przykładem jako przewodnikiem, rozważmy rodzaje rozkazów, które muszą być przewidziane w rzeczywistym komputerze. Komputer powinien dysponować listą rozkazów, które umożliwiają użytkownikowi formułowanie dowolnych zadań dotyczących przetwarzania danych. In-

nym sposobem widzenia tego problemu jest rozważenie możliwości języka wysokiego poziomu. Dowolny program napisany w języku wysokiego poziomu musi być przetłumaczony na język maszynowy, aby mógł być wykonany. Wobec tego lista rozkazów maszynowych musi być wystarczająca do wyrażenia dowolnych instrukcji języka wysokiego poziomu. Mając to na uwadze, możemy podzielić rozkazy na następujące rodzaje:

- ☐ *Przetwarzanie danych.* Rozkazy arytmetyczne i logiczne.
- ☐ *Przechowywanie danych.* Rozkazy pamięciowe.
- ☐ *Ruch danych.* Rozkazy wejścia-wyjścia.
- ☐ *Sterowanie.* Rozkazy testowania i rozgałęzienia.

Rozkazy **arytmetyczne** zapewniają zdolność obliczeniową przetwarzania danych numerycznych. Rozkazy **logiczne** (Boole'a) operują na bitach słowa raczej jako na bitach niż na liczbach; wobec tego zapewniają one zdolność przetwarzania dowolnych innych rodzajów danych, jakie użytkownik może chcieć wykorzystywać. Operacje te są wykonywane głównie na danych znajdujących się w rejestrach procesora. Wobec tego muszą istnieć rozkazy **pamięciowe** służące do przenoszenia danych między pamięcią a rejestrami. Rozkazy **wejścia-wyjścia** są potrzebne do przenoszenia danych i programów do pamięci oraz wyników obliczeń do użytkownika. **Rozkazy testowania** są używane do sprawdzania wartości słów danych lub stanu obliczeń. **Rozkazy rozgałęzienia** są używane do przeskoczenia do innego zestawu rozkazów zależnie od podjętych decyzji.

Przeanalizujemy różne rodzaje rozkazów bardziej szczegółowo w dalszym ciągu rozdziału.

Liczba adresów

Jeden z tradycyjnych sposobów opisywania architektury procesora opiera się na liczbie adresów zawartych w każdym rozkazie. Wymiar ten stał się mniej znaczący wraz ze wzrostem złożoności CPU. Jednak przeanalizowanie tej własności jest użyteczne.

Jaka jest maksymalna liczba adresów, która może być potrzebna w rozkazie? Jest oczywiste, że operacje arytmetyczne i logiczne wymagają argumentów. Praktycznie wszystkie operacje arytmetyczne i logiczne są albo operacjami jednoargumentowymi, albo dwuargumentowymi. Wobec tego potrzebujemy maksymalnie dwóch adresów jako odniesienia do argumentów. Wynik operacji musi być przechowany, co sugeruje trzeci adres. Na zakończenie, po wykonaniu rozkazu, musi być pobrany następny rozkaz i potrzebny jest jego adres.

Z powyższego rozumowania wynika, że rozkaz powinien zawierać cztery odniesienia adresowe: dwa dotyczące argumentów, jedno wyniku i jedno następnego rozkazu. W praktyce czteroadresowe rozkazy występują bardzo rzadko. Większość procesorów to jednostki jedno-, dwu- lub trzyadresowe, przy czym adres następnego rozkazu jest domyślny (uzyskiwany z licznika programu).

Na rysunku 9.3 znajduje się porównanie typowych rozkazów jedno-, dwu- i trzyadresowych, które mogą być użyte do obliczenia $Y = (A - B) \div (C + D \times E)$. Za pomocą trzech adresów każdy rozkaz ustala dwie lokalizacje argumentów i lokalizację wyniku. Ponieważ nie chcielibyśmy zmieniać lokalizacji argumentów, do przechowywania wyników pośrednich jest używana tymczasowa lokalizacja T. Zauważmy, że występują tu cztery rozkazy i że oryginalne wyrażenie ma pięć argumentów.

Trzyadresowe formaty rozkazów nie są powszechne, ponieważ są stosunkowo długie w związku z koniecznością objęcia trzech odniesień adresowych. W przypadku rozkazów dwuadresowych oraz operacji binarnych jeden adres musi być używany podwójnie, zarówno jako adres argumentu, jak i wyniku. Wobec tego rozkaz SUB Y, B powoduje obliczenie $Y - B$ oraz przechowanie wyniku w Y. Format dwuadresowy umożliwia zmniejszenie wymagań objętościowych, jednak wprowadza pewne niewygody. Aby zapobiec zmianie wartości argumentu, stosowany jest rozkaz MOVE, powodujący przeniesienie jednej z wartości do lokalizacji wyniku lub lokalizacji tymczasowej przed wykonaniem operacji. Nasz próbny program rozszerza się w ten sposób do sześciu rozkazów.

(a)

Rozkaz		Komentarz
SUB	Y, A, B	$Y \leftarrow A - B$
MPY	T, D, E	$T \leftarrow D \times E$
ADD	T, T, C	$T \leftarrow T + C$
DIV	Y, Y, T	$Y \leftarrow Y \div T$

(b)

Rozkaz		Komentarz
MOVE	Y, A	$Y \leftarrow A$
SUB	Y, B	$Y \leftarrow Y - B$
MOVE	T, D	$T \leftarrow D$
MPY	T, E	$T \leftarrow T \times E$
ADD	T, C	$T \leftarrow T + C$
DIV	Y, T	$Y \leftarrow Y \div T$

(c)

Rozkaz		Komentarz
LOAD	D	$AC \leftarrow D$
MPY	E	$AC \leftarrow AC \times E$
ADD	C	$AC \leftarrow AC + C$
STOR	Y	$Y \leftarrow AC$
LOAD	A	$AC \leftarrow A$
SUB	B	$AC \leftarrow AC - B$
DIV	Y	$AC \leftarrow AC \div Y$
STOR	Y	$Y \leftarrow AC$

RYSunek 9.3. Programy do obliczenia $Y = (A - B) \div (C + D \times E)$: (a) rozkazy trójadresowe; (b) rozkazy dwuadresowe; (c) rozkazy jednoadresowe

Jeszcze prostszy jest rozkaz jednoadresowy. Aby mógł on działać, drugi adres musi być domyślny. Było to powszechne we wcześniejszych maszynach, w których adres domyślny dotyczył rejestru procesora zwanego akumulatorem (AC). Akumulator zawiera jeden z argumentów i jest używany do przechowywania wyniku. Zadanie z naszego przykładu rozrasta się do ośmiu rozkazów.

W rzeczywistości jest możliwe, aby niektóre rozkazy funkcjonowały bez adresu. Rozkazy bezadresowe mogą być stosowane w przypadku szczególnej organizacji pamięci zwanej **stosem**. Stos jest zbiorem lokacji funkcjonującym na zasadzie ostatni na wejściu - pierwszy na wyjściu. Stos znajduje się w znanym położeniu i często przynajmniej jego dwa szczytowe elementy znajdują się w rejestrach procesora. Stosy są opisane w dodatku 9.A. Używanie ich przeanalizujemy w dalszym ciągu tego rozdziału oraz w rozdz. 10.

W tabeli 9.1 znajduje się podsumowanie interpretacji rozkazów zawierających 0, 1, 2 lub 3 adresy. W każdym przypadku jest przyjęte, że adres następnego rozkazu jest domyślny oraz że jedna operacja wymaga dwóch argumentów i dostarcza jednego wyniku.

TABELA 9.1. Wykorzystanie adresów rozkazu (rozkazy nierozgałęziające)

Liczba adresów	Reprezentacja symboliczna	Interpretacja
3	OP A, B, C	$A \leftarrow B \text{ OP } C$
2	OP A, B	$A \leftarrow A \text{ OP } B$
1	OP A	$AC \leftarrow AC \text{ OP } A$
0	OP	$T \leftarrow T \text{ OP } (T-1)$

Liczba adresów w rozkazie jest podstawową decyzją projektową. Mniej adresów w rozkazie oznacza prymitywniejsze rozkazy, wymagające prostszych procesorów. Oznacza to także krótsze rozkazy. Jednak programy zawierają wówczas więcej rozkazów, co na ogół powoduje wydłużenie czasów wykonywania oraz wydłużenie i skomplikowanie samych programów. Między rozkazami jedno- i wieloadresowymi występuje pewien próg. W przypadku rozkazów jednoadresowych programista ma na ogół dostęp tylko do jednego rejestru o ogólnym przeznaczeniu, tj. do akumulatora. W przypadku rozkazów wieloadresowych zwykle występuje wiele rejestrów o ogólnym przeznaczeniu. Dzięki temu niektóre operacje mogą być wykonywane na samych tylko rejestrach. Ponieważ dostęp do rejestrów jest szybszy, niż do pamięci, przyspiesza to wykonywanie programów. Z powodu elastyczności i możliwości używania wielu rejestrów w większości współczesnych maszyn stosuje się mieszaninę rozkazów dwu- i trzeadresowych.

Kompromis projektowy związany z wyborem liczby adresów w rozkazie jest dodatkowo komplikowany przez inne czynniki. Istnieje problem, czy adres odnosi się do lokacji w pamięci, czy do rejestru. Ponieważ rejestrów jest mniej, odniesienie do rejestru wymaga mniejszej liczby bitów. Ponadto, jak zobaczymy w następnym rozdziale, maszyna może oferować różne tryby adresowania, a specyfikacja trybu zabiera jeden lub wiele bitów. W rezultacie w większości projektów procesorów przewidziano różne formaty rozkazów.

Projektowanie listy rozkazów

Jednym z najbardziej interesujących i najczęściej analizowanych aspektów projektowania komputera jest projektowanie listy rozkazów. Jest ono bardzo złożone, ponieważ wpływa na wiele aspektów systemu komputerowego. Lista rozkazów definiuje wiele funkcji realizowanych przez procesor i dlatego ma znaczący wpływ na implementację procesora. Lista rozkazów jest środkiem, za pomocą którego programista steruje procesorem. Dlatego też podczas projektowania listy rozkazów muszą być brane pod uwagę wymagania programisty.

Niespodzianką dla czytelnika może być to, że pewne najbardziej podstawowe problemy związane z projektowaniem list rozkazów nadal są przedmiotem debat. Istotnie, w ostatnich latach poziom niezgodności w odniesieniu do tych problemów wzrósł. Do najważniejszych spośród tych problemów projektowych należą:

- a *Repertuar operacji*. Ile operacji, które operacje i jak złożone operacje powinny być przewidziane.
- *Rodzaje danych*. Różne rodzaje danych, na których dokonywane są operacje.
- a *Format rozkazu*. Długość rozkazu (w bitach), liczba adresów, rozmiar różnych pól itd.
- *Rejestry*. Liczba rejestrów w procesorze, do których mogą się odnosić rozkazy, oraz ich zastosowanie.
- Q *Adresowanie*. Tryb lub tryby, w których są specyfikowane adresy argumentów.

Zagadnienia te są ze sobą ściśle powiązane i muszą być rozważane łącznie podczas projektowania listy rozkazów. Oczywiście, w tej książce muszą być one rozpatrzone po kolei, jednak spróbujemy pokazać ich wzajemne zależności.

Ze względu na wagę tego tematu, większość części III jest poświęcona projektowaniu listy rozkazów. Po tym podrozdziale mającym charakter przeglądu, dalej w tym rozdziale zajmiemy się analizą rodzaju danych i repertuarem operacji. W rozdziale 10 rozpatrzmy tryby adresowania (co obejmuje rozważenie rejestrów) oraz formaty rozkazów. W rozdziale 12 przedstawimy ekscytujące nowe rozwiązania znane jako komputery o zredukowanej liście rozkazów (RISC). Architektura RISC stawia pod znakiem zapytania wiele decyzji podjętych przy projektowaniu list rozkazów wielu współczesnych komputerów komercyjnych. Przykładem maszyny RISC jest PowerPC. W tym rozdziale i w następnym korzystamy z tego przykładu. Jednak znaczenie PowerPC w kontekście RISC jest przedyskutowane dopiero w rozdz. 12.

RODZAJE ARGUMENTÓW

Rozkazy maszynowe operują na danych. Najważniejsze, ogólne kategorie danych to:

- a adresy,
- a liczby,
- a znaki,
- a dane logiczne.

Analizując tryby adresowania w rozdz. 10 zobaczymy, że w rzeczywistości adresy są formą danych. W wielu przypadkach muszą być wykonywane pewne obliczenia na odniesieniach w rozkazach w celu wyznaczenia adresu w pamięci głównej lub wirtualnej. W tym kontekście adresy są uważane za liczby całkowite bez znaku.

Innymi powszechnymi rodzajami danych są liczby, znaki i dane logiczne. Każdy z nich zostanie krótko przeanalizowany w tym podrozdziale. Ponadto, w niektórych maszynach występują specjalizowane rodzaje danych lub struktur danych. Na przykład, operatorzy maszyn mogą działać bezpośrednio na listach lub ciągach znaków.

Liczby

Wszystkie języki maszynowe obejmują dane numeryczne. Nawet w przypadku przetwarzania danych nienumerycznych potrzebne są liczby występujące w licznikach, określające szerokości pól itd. Ważną różnicą między liczbami używanymi w zwykłej matematyce a liczbami przechowywanymi w komputerze jest to, że te ostatnie są ograniczone. Jest to prawda w podwójnym sensie. Po pierwsze, istnieje granica wielkości liczb reprezentowalnych w komputerze, a po drugie, w przypadku liczb zmiennopozycyjnych, granica ich dokładności. Programista powinien więc znać konsekwencje zaokrąglania, przepełnienia i niedomiaru.

W komputerach są powszechne trzy rodzaje danych numerycznych:

- a całkowite lub stałopozycyjne,
- zmiennopozycyjne,
- a dziesiętne.

Dwa pierwsze rodzaje przeanalizowaliśmy dość szczegółowo w rozdz. 8. Pozostaje więc powiedzieć kilka słów o liczbach dziesiętnych.

Chociaż wszystkie wewnętrzne operacje komputera są z natury binarne, użytkownicy systemu operują liczbami dziesiętnymi. Istnieje więc konieczność konwersji liczb dziesiętnych na binarne na wejściu oraz binarnych na dziesiętne na wyjściu. W przypadku zastosowań, w których występuje wiele operacji wej-

ścia-wyjścia oraz raczej mało stosunkowo prostych obliczeń, wygodniej jest przechowywać i przetwarzać liczby w postaci dziesiętnej. Najbardziej powszechną reprezentacją służącą do tego celu są upakowane liczby dziesiętne.

W przypadku upakowanych liczb dziesiętnych każda cyfra dziesiętna jest reprezentowana w kodzie 4-bitowym, tworzonym w oczywisty sposób. Wobec tego 0 = 0000, 1 = 0001, ..., 8 = 1000 i 9 = 1001. Zauważmy, że jest to kod raczej nieefektywny, ponieważ używa tylko 10 spośród możliwych 16 wartości 4-bitowych. W celu utworzenia liczb 4-bitowe kody są zestawiane szeregowo, zwykle w wielokrotnościach 8 bitów. Kodem 246 jest więc 0000001001000110. Kod ten jest oczywiście mniej zwarty niż prosta reprezentacja binarna, jednak pozwala uniknąć konwersji. Liczby ujemne mogą być reprezentowane przez dołączenie 4-bitowej cyfry znaku albo z lewej, albo z prawej strony ciągu upakowanych cyfr dziesiętnych. Na przykład kod 1111 może reprezentować znak minus.

W wielu maszynach przewidziano rozkazy arytmetyczne do wykonywania operacji bezpośrednio na upakowanych liczbach dziesiętnych. Algorytmy te są całkiem podobne do opisanych w podrozdz. 8.3, jednak muszą uwzględniać operację przeniesienia dziesiętnego.

Znaki

Powszechną formą danych jest tekst lub ciągi znaków. Chociaż dane tekstowe są najwygodniejsze dla ludzi, nie mogą one być w postaci znaków łatwo przechowywane i transmitowane przez systemy przetwarzania i komunikacji. Systemy te są zaprojektowane dla danych binarnych. W związku z tym opracowano wiele kodów, za pomocą których znaki są reprezentowane w postaci ciągów bitów. Być może najwcześniejszym, powszechnie znanym przykładem jest kod Morse'a. Dzisiaj najczęściej stosowanym kodem znaków w Stanach Zjednoczonych jest kod ASCII (tab. 6.1), ogłoszony przez American National Standards Institute (ANSI). ASCII jest używany szeroko poza Stanami Zjednoczonymi. Każdy znak w tym kodzie jest reprezentowany przez unikatowy wzór 7-bitowy; może więc być reprezentowanych 128 różnych znaków. Jest to liczba większa niż wymagana do reprezentowania znaków drukowanych, dlatego część wzorów reprezentuje znaki *sterowania*. Niektóre z tych znaków sterowania są stosowane do sterowania drukowaniem znaków na stronie. Inne wiążą się z procedurami komunikacyjnymi. Znaki zakodowane za pomocą ASCII są prawie zawsze przechowywane i transmitowane przy użyciu 8 bitów na znak. Ósmy bit może być ustalony jako 0 lub użyty jako bit parzystości do wykrywania błędów. W tym ostatnim przypadku bit jest ustalany w ten sposób, że całkowita liczba binarnych jedynek w każdym okcie jest zawsze nieparzysta (parzystość nieparzysta) lub zawsze parzysta (parzystość parzysta).

Zauważmy, że w przypadku wzoru bitowego ASCII 011XXXX, cyfry od 0 do 9 są reprezentowane przez ich binarne równoważniki, 0000 do 1001, zło-

kalizowane w prawej części wzoru. Jest to ten sam kod, co stosowany w przypadku upakowanych liczb dziesiętnych. Ułatwia to konwersję między 7-bitowym kodem ASCII a 4-bitową upakowaną reprezentacją dziesiętną.

Innym kodem używanym do kodowania znaków jest EBCDIC (*Extended Binary Coded Decimal Interchange Code*). Kod EBCDIC jest stosowany w maszynach IBM S/370. Jest to kod 8-bitowy. Podobnie jak ASCII, EBCDIC jest kompatybilny z upakowaną reprezentacją dziesiętną. W przypadku EBCDIC kody od 11110000 do 1111001 reprezentują cyfry od 0 do 9.

Dane logiczne

Zwykle każde słowo lub inna jednostka adresowalna (bajt, półsłowo itd) jest traktowane jako jednostka danych. Jest jednak czasem użyteczne rozpatrywanie jednostki n -bitowej jako składającej się z n jednobitowych pozycji, z których każda ma wartość 1 lub 0. Dane widziane w ten sposób są traktowane jako dane *logiczne*.

Ten zorientowany bitowo obraz ma dwie zalety. Po pierwsze, czasem może być potrzebne przechowywanie tablicy boolowskich lub binarnych danych, z których każda może przybierać tylko wartości 1 (prawdziwa) i 0 (fałszywa). Dane w postaci logicznej mogą być bardziej efektywnie przechowywane w pamięci. Po drugie, występują sytuacje, w których chcemy manipulować bitami jednostki danych. Jeśli na przykład w oprogramowaniu są wdrażane operacje zmiennopozycyjne, to w pewnych operacjach zachodzi potrzeba przesuwania znaczących bitów. Inny przykład: w celu konwersji kodu ASCII na upakowany kod dziesiętny musimy wyciągnąć 4 prawe bity z każdego bajtu.

Zauważmy, że w powyższych przykładach te same dane są traktowane czasem jako logiczne a czasem jako numeryczne lub tekstowe. „Rodzaj” jednostki danych jest określany przez wykonywaną na nich operację. Nie jest to normalne w przypadku języków wysokiego poziomu (np. Pascala), ma to jednak miejsce prawie zawsze w przypadku języka maszynowego.

Rodzaje danych Pentium

Pentium może operować danymi o długości 8 bitów (bajt), 16 bitów (słowo), 32 bity (podwójne słowo) i 64 bity (poczwórne słowo). Aby osiągnąć maksymalną elastyczność struktur danych oraz efektywne wykorzystanie pamięci, słowa nie muszą być wyrównywane pod adresami numerowanymi parzyście, podwójne słowa nie muszą być wyrównywane pod adresami podzielnymi przez cztery a poczwórne słowa nie muszą być wyrównywane pod adresami podzielnymi przez 8. Jeśli jednak dostęp do danych następuje przez magistralę 32-bitową, przesyłanie danych następuje w podwójnych słowach jako jednostkach, począwszy od adresów podzielnych przez 4. Procesor zamienia zapotrzebowanie odniesione do niewyrównanych wartości na sekwencję zapotrzebowań

dostosowanych do transferu magistralowego. Podobnie jak w przypadku wszystkich procesorów Intel 80x86, Pentium używa stylu określanego jako „cienkokońcowy”* (*little-endian*); oznacza to, że najmniej znaczący bajt jest przechowywany pod najniższym adresem (patrz dodatek 9B zawierający omówienie „cienko- i grubokońcowości”).

Bajt, słowo, słowo podwójne i słowo poczwórne są określane jako ogólne rodzaje danych. Ponadto Pentium przyjmuje obszerny zestaw specyficznych rodzajów danych, które są rozpoznawane i uruchamiane za pomocą szczególnych rozkazów. Są one te wymieniono w tabeli 9.2.

TABELA 9.2. Rodzaje danych procesora Pentium

Rodzaj danych	Opis
Ogólne	Lokacje o dowolnej zawartości binarnej obejmujące bajt, słowo (16 bitów), podwójne słowo (32 bity) lub poczwórne słowo (64 bity).
Całkowite	Wartości binarne ze znakiem, zawarte w bajcie, słowie lub podwójnym słowie, w reprezentacji uzupełnienia do dwóch.
Porządkowe	Liczby całkowite bez znaku zawarte w bajcie, słowie lub podwójnym słowie.
Nieupakowana binarna reprezentacja liczb dziesiętnych (BCD)	Reprezentacja cyfry BCD w zakresie 0÷9, o jednej cyfrze w każdym bajcie.
Upakowana BCD	Upakowana bajtowa reprezentacja dwóch cyfr BCD; wartości w zakresie 0÷99.
Wskaźnik bliski (<i>near pointer</i>)	32-bitowy adres efektywny, reprezentujący przesunięcie wewnątrz segmentu. Używany dla wszystkich wskaźników w pamięci niesegmentowanej oraz do odniesień wewnątrz segmentu w pamięci segmentowanej.
Pole bitowe	Sekwencja sąsiadujących bitów, w której pozycja każdego bitu jest traktowana jako jednostka niezależna. Łańcuch bitów może się rozpoczynać od dowolnej pozycji bitowej dowolnego bajtu i może zawierać do $2^{32}-1$ bitów.
Łańcuch bajtów	Sekwencja sąsiadujących bajtów, słów lub podwójnych słów zawierająca od zera do $2^{32}-1$ bajtów.
Zmiennopozycyjne	Patrz rys. 9.4.

Rodzaj zmiennopozycyjny odnosi się w rzeczywistości do zbioru rodzajów, które są używane przez jednostkę zmiennopozycyjną i którymi operują rozkazy Zmiennopozycyjne. Jak widać na rys. 9.4, rozkazy zmiennopozycyjne mogą operować liczbami całkowitymi, upakowanymi liczbami dziesiętnymi oraz liczbami zmiennopozycyjnymi. Liczby całkowite mają postać uzupełnienia do dwóch i mogą mieć długość 16, 32 lub 64 bitów. Upakowane dziesiętne liczby całkowite są przechowywane w reprezentacji znak-moduł o 18 cyfrach obejmujących zakres 0 do 9. Trzy reprezentacje zmiennopozycyjne odpowiadają normie IEEE 754.

* Zobacz przypis na s. 416 w dodatku 9b (*przyp.red.*).

Formaty danych	Zakres	Precyzja	Najbardziej znaczący bajt								Najwyższy adresowany bajt							
			7	0	7	0	7	0	7	0	7	0	7	0	7	0		
Liczba całkowita o długości słowa	10^4	16 bitów	(uzupełnienie do dwóch)															
Krótką liczba całkowita	10^9	32 bity	(uzupełnienie do dwóch)															
Długa liczba całkowita	10^{18}	64 bity	(uzupełnienie do dwóch)															
Upakowana BCD	10^{18}	18 cyfr	s X d₁₇ d₁₆ d₁₅ d₁₄ d₁₃ d₁₂ d₁₁ d₁₀ d₉ d₈ d₇ d₆ d₅ d₄ d₃ d₂ d₁ d₀															
Precyzja pojedyncza	10^{+4932}	24 bity	s Przesunięty wykładnik Mantysa															
Precyzja podwójna	10^{+38}	53 bity	s Przesunięty wykładnik Mantysa															
Precyzja rozszerzona	10^{+308}	64 bity	s Przesunięty wykładnik Mantysa															

s - bit znaku (0 = dodatnia; 1 = ujemna),

d_n - cyfra dziesiętna (dwie na bajt),

X - bity nie mające znaczenia; ignorowane przy ładowaniu; zera przy zapisywaniu,

D - położenie domyślnego przecinka binarnego,

I - bit całkowitości mantysy; przechowywany przy tymczasowej wartości rzeczywistej, domyślny przy precyzji pojedynczej i podwójnej.

RYSunek 9.4. Formaty danych numerycznych jednostki zmiennopozycyjnej Pentium

Rodzaje danych PowerPC

PowerPC może operować danymi o długości 8 bitów (bajt), 16 bitów (półsłowo), 32 bity (słowo) i 64 bity (podwójne słowo). Niektóre rozkazy wymagają, żeby argumenty w pamięci były wyrównane na granicy 32 bitów. Na ogół jednak wyrównanie nie jest wymagane. Interesującą własnością PowerPC jest to, że może być stosowany zarówno styl „cienko-”, jak i „grubokońcowy”; znaczy to,

że najmniej znaczący bajt może być przechowywany pod najniższym lub najwyższym adresem („cienko- i grubokońcowość” jest omówiona w dodatku 9B).

Bajt, półsłowo, słowo i podwójne słowo stanowią ogólne rodzaje danych. Procesor interpretuje zawartość jednostki danych zależnie od rozkazu. Procesor stałopozycyjny rozpoznaje następujące rodzaje danych:

- **Bajt bez znaku.** Może być użyty w operacjach logicznych oraz operacjach arytmetycznych na liczbach całkowitych. Przy ładowaniu z pamięci do rejestru ogólnego jest do niego dodawane zero po lewej stronie w celu dopełnienia rejestru.
- a **Półsłowo bez znaku.** Jak wyżej, dla wielkości 16-bitowych.
- **Półsłowo ze znakiem.** Używane w operacjach arytmetycznych; rozciągane w lewo w celu wypełnienia rejestru poprzez replikowanie bitu znaku we wszystkich wolnych pozycjach.
- a **Słowo bez znaku.** Używane w operacjach logicznych i jako wskaźnik adresu.
- a **Słowo ze znakiem.** Używane w operacjach arytmetycznych.
- a **Podwójne słowo bez znaku.** Używane jako wskaźnik adresu.
- Q **Łańcuch bajtów.** Od 0 do 128 bajtów długości.

Ponadto PowerPC operuje na danych zmiennopozycyjnych o pojedynczej i podwójnej dokładności, zdefiniowanych w IEEE 754.

RODZAJE OPERACJI

Liczba kodów operacji różnych procesorów różni się znacznie. Jednak we wszystkich maszynach można odnaleźć takie same ogólne rodzaje operacji. Użyteczny, typowy podział wygląda następująco:

- operacje transferu danych,
- Q operacje arytmetyczne,
- operacje logiczne,
- Q operacje konwersji,
- a operacje wejścia-wyjścia,
- a operacje sterowania systemowego,
- operacje przekazywania sterowania.

Tabela 9.3 (oparta na [HAYE88]) zawiera zestawienie powszechnie stosowanych rodzajów rozkazów należących do wszystkich wymienionych kategorii. W tym punkcie dokonamy krótkiego przeglądu różnych rodzajów operacji, omawiając jednocześnie działania podejmowane przez procesor w celu wykonania poszczególnych rodzajów operacji (działania te są podsumowane w tabeli 9.4). Ten ostatni temat jest bardziej szczegółowo przedstawiony w rozdz. 11.

TABELA 9.3. Operacje powszechnie występujące w listach rozkazów

Rodzaj	Nazwa operacji	Opis
Transfer danych	Move (transfer)	przeniesienie słowa lub bloku ze źródła do miejsca przeznaczenia
	Store	przeniesienie słowa z procesora do pamięci
	Load (fetch)	przeniesienie słowa z pamięci do procesora
	Exchange	zamiana zawartości źródła i miejsca przeznaczenia
	Clear (reset)	przeniesienie słowa złożonego z 0 do miejsca przeznaczenia
	Set	przeniesienie słowa złożonego z 1 do miejsca przeznaczenia
	Push	przeniesienie słowa ze źródła na wierzchołek stosu
	Pop	przeniesienie słowa z wierzchołka stosu do miejsca przeznaczenia
Arytmetyczne	Add	obliczenie sumy dwóch argumentów
	Subtract	obliczenie różnicy dwóch argumentów
	Multiply	obliczenie iloczynu dwóch argumentów
	Divide	obliczenie ilorazu dwóch argumentów
	Absolute	zamiana argumentu na jego wartość bezwzględna
	Negate	zmiana znaku argumentu
	Increment	dodanie 1 do argumentu
	Decrement	odjęcie 1 od argumentu
Logiczne	AND	} wykonanie określonej operacji logicznej na poziomie bitowym
	OR	
	NOT (complement)	
	Exclusive-OR	J
	Test	zbadanie określonego warunku; jako wynik ustawienie znacznika (znaczników) stanu
	Compare	wykonanie logicznego lub arytmetycznego porównania dwóch lub wielu argumentów; na podstawie wyniku ustawienie znacznika (znaczników) stanu
	Set control variables	klasa rozkazów ustalających elementy sterowania dla celów ochrony, przetwarzania przerwań, sterowania zegarowego
	Shift	przesunięcie argumentu w prawo lub w lewo z wprowadzeniem stałej na końcu
	Rotate	przesunięcie argumentu w prawo lub w lewo z łączeniem końca z początkiem
Przeniesienie sterowania	Jump (branch)	przeniesienie bezwarunkowe; ładuje określony adres do PC
	Jump Conditional	zbadanie określonego warunku; albo załadowanie określonego adresu do PC, albo nie robienie niczego, zależnie od warunku
	Jump to Subroutine	umieszczenie informacji kontrolnej bieżącego programu w znanym miejscu; skok do określonego adresu
	Return	zamiana zawartości PC i innych rejestrów na dane przechowywane w znanym miejscu
	Execute	pobranie argumentu z określonego miejsca i wykonanie go jako rozkazu; nie modyfikuje PC
	Skip	inkrementowanie PC w celu pominięcia następnego rozkazu
	Skip conditional	zbadanie określonego warunku; albo pominięcie następnego rozkazu, albo nie robienie niczego, zależnie od warunku
	Halt	zatrzymanie wykonywania programu

TABELA 9.3. cd.

Rodzaj	Nazwa operacji	Opis
Wejście- -wyjście	Wait (hold)	zatrzymanie wykonywania programu; powtarzalne badanie określonego warunku; wykonywanie wznowiane po spełnieniu warunku
	No operation	nie jest wykonywana żadna operacja, kontynuowane jest wykonywanie programu
	Input (read)	przeniesienie danych z określonego portu lub przyrządu we-wy do miejsca przeznaczenia, np. do pamięci głównej lub rejestru procesora
	Output (write)	przeniesienie danych z określonego źródła do portu lub przyrządu we-wy
	Start I/O	przeniesienie rozkazów do procesora we-wy w celu inicjowania operacji we-wy
Konwersja	Test I/O	przeniesienie informacji o stanie z systemu we-wy do określonego miejsca przeznaczenia
	Translate	przetłumaczenie danych w sekcji pamięci na podstawie tabeli odpowiedniości
	Convert	przekształcenie zawartości słowa z jednej postaci na drugą (np. z upakowanej dziesiętnej na binarną)

TABELA 9.4. Czynności wykonywane przez procesor w przypadku różnych rodzajów operacji

Transfer danych	Przeniesienie danych z jednego miejsca do drugiego. Jeśli zaangażowana jest pamięć: określenie adresu pamięci, przekształcenie adresu wirtualnego na rzeczywisty, sprawdzenie pamięci podręcznej, zainicjowanie zapisu/odczytu pamięci.
Arytmetyczne	Mogą powodować transfer danych przed i (lub) po: wykonaniu funkcji w ALU, ustawieniu kodu warunkowego i znacznika stanu.
Logiczne	Jak arytmetyczne.
Konwersja	Podobnie do arytmetycznych i logicznych. Mogą obejmować specjalne funkcje logiczne w celu dokonania konwersji.
Transfer sterowania	Aktualizowanie licznika programu. W przypadku wywołania/powrotu z podprogramu - zarządzanie przenoszeniem i powiązaniem parametrów.
Wejścia-wyjścia	Wydanie rozkazu modułowi wejścia-wyjścia. Jeśli wejście-wyjście jest odwzorowane w pamięci, określenie adresu odwzorowanego w pamięci.

Transfer danych

Najbardziej podstawowym rodzajem rozkazów maszynowych jest rozkaz przesyłania (transferu) danych. Rozkaz taki musi precyzować kilka rzeczy. Po pierwsze, musi być ustalone położenie argumentów źródłowych i wyników. Możliwe położenia obejmują pamięć, rejestr lub wierzchołek stosu. Po drugie, musi być określona długość danych przeznaczonych do przesłania. Po trzecie, tak jak w przypadku wszystkich rozkazów z argumentami, musi być określony

tryb adresowania każdego argumentu. To ostatnie zagadnienie jest omówione w rozdz. 10.

Wybór rozkazów transferu danych, które mają być włączone do listy rozkazów, stanowi egzemplifikację wyborów, jakich musi dokonywać projektant. Na przykład ogólne położenie (pamięć lub rejestr) argumentu może być wskazane albo w specyfikacji kodu operacji, albo samego argumentu. W tabeli 9.5 są podane przykłady najczęściej stosowanych rozkazów transferu danych IBM S/370. Zauważmy, że występują tutaj różne warianty ilości danych przeznaczonych do przesłania (8, 16, 32 i 64 bity). Istnieją też różne rozkazy dotyczące transferów z rejestru do rejestru, z rejestru do pamięci i z pamięci do rejestru. Dla kontrastu, w maszynie VAX występuje rozkaz przeniesienia (MOV) z wariantami dotyczącymi różnych ilości danych, jednak określa on, czy argument jest w rejestrze czy w pamięci. Podejście stosowane w komputerze VAX jest nieco łatwiejsze dla programisty, który operuje mniejszą ilością mnemoników. Jest ono jednak mniej zwarte niż rozwiązanie rodem z IBM S/370, ponieważ lokalizacja (rejestr czy pamięć) każdego argumentu musi być wyspecyfikowana w rozkazie. Wrócimy do tego rozróżnienia przy dyskusji nad formatami rozkazów w następnym rozdziale.

TABELA 9.5. Przykłady operacji transferu danych w IBM S/370

Mnemonik operacji	Nazwa	Liczba przenoszonych bitów	Opis
L	Load	32	przeniesienie z pamięci do rejestru
LH	Load halfword	16	przeniesienie z pamięci do rejestru
LR	Load	32	przeniesienie z rejestru do rejestru
LER	Load(short)	32	przeniesienie z rejestru zmiennopozycyjnego do rejestru zmiennopozycyjnego
LE	Load (short)	32	przeniesienie z pamięci do rejestru zmiennopozycyjnego
LDR	Load (long)	64	przeniesienie z rejestru zmiennopozycyjnego do rejestru zmiennopozycyjnego
LD	Load (long)	64	przeniesienie z pamięci do rejestru zmiennopozycyjnego
ST	Store	32	przeniesienie z rejestru do pamięci
STH	Store halfword	16	przeniesienie z rejestru do pamięci
STC	Store character	8	przeniesienie z rejestru do pamięci
STE	Store (short)	32	przeniesienie z rejestru zmiennopozycyjnego do pamięci
STD	Store (long)	64	przeniesienie z rejestru zmiennopozycyjnego do pamięci

Rozpatrując zagadnienie w kategoriach działań procesora, operacje transferu danych są być może rodzajem najprostszym. Jeśli zarówno źródłem, jak i miejscem przeznaczenia są rejestry, to procesor po prostu powoduje przeniesienie danych z jednego rejestru do drugiego; jest to operacja wewnętrzna procesora. Jeśli jeden lub oba argumenty znajdują się w pamięci, to procesor musi wykonać niektóre lub wszystkie z następujących działań:

1. Obliczanie adresu pamięci na podstawie określonego trybu adresowania (omówionego w rozdz. 10).
2. Jeśli adres dotyczy pamięci wirtualnej, przekształcenie adresu z wirtualnego na rzeczywisty.
3. Sprawdzenie, czy adresowana jednostka znajduje się w pamięci podręcznej.
4. Jeśli nie, wydanie rozkazu modułowi pamięci.

Operacje arytmetyczne

Większość maszyn realizuje podstawowe operacje arytmetyczne dodawania, odejmowania, mnożenia i dzielenia. Zawsze są one przewidziane dla liczb całkowitych (stałopozycyjnych) ze znakiem. Często są również przewidziane dla liczb zmiennopozycyjnych i upakowanych dziesiętnych.

Inne możliwe operacje obejmują różne rozkazy jednoargumentowe. Są to na przykład:

- a Wartość bezwzględna.** Określenie Wartości bezwzględnej argumentu.
- Q Negacja.** Zanegowanie argumentu.
- a Inkrementacja.** Przyrost argumentu o 1.
- a Dekrementacja.** Zmniejszenie argumentu o 1.

Wykonywanie rozkazu arytmetycznego może obejmować operacje przesłania danych w celu umieszczenia ich na wejściu ALU oraz w celu wyprowadzenia ich z wyjścia ALU. Na rysunku 3.5 został pokazany ruch danych objęty operacjami przesyłania danych oraz operacjami arytmetycznymi. Ponadto, oczywiście, część procesora w postaci ALU wykonuje żadaną operację.

Operacje logiczne

Większość maszyn umożliwia różne operacje manipulowania (ang. *twiddling*) pojedynczymi bitami słowa lub innych adresowalnych jednostek. Są one oparte na operacjach Boole'a (patrz dodatek do książki).

Niektóre z podstawowych operacji logicznych, które mogą być dokonywane na danych boolowskich lub binarnych, są podane w tabeli 9.6. Operacja NIE (NOT) odwraca bit. Operacje I (AND), LUB (OR) i LUB wykluczające (Exclusive OR, XOR) są najczęściej używanymi funkcjami logicznymi z dwoma argumentami. Operacja RÓWNY (EQUAL) jest użytecznym testem binarnym.

TABELA 9.6. Podstawowe operacje logiczne

P	Q	NOT P	NOT Q	P AND Q	P OR Q	P XOR Q	P = Q
0	0	1	1	0	0	0	1
0	1	1	0	0	1	1	0
1	0	0	1	0	1	1	0
1	1	0	0	1	1	0	1

Te operacje logiczne mogą być stosowane do bitów należących do n -bitowych logicznych jednostek danych. Jeśli więc dwa rejestry zawierają następujące dane:

(R1) = 10100101

(R2) = 00001111

to

(R1)AND(R2)=0000101

Wobec tego operacja AND może być użyta jako *maska*, umożliwiającą wybranie pewnych bitów w słowie i zerowanie pozostałych bitów. Weźmy inny przykład: jeśli rejestry zawierają:

(R1) = 10100101

(R2) = 11111111

to

(R1) XOR (R2) = 01011010

Gdy jedno ze słów składa się z samych jedynek, to operacja XOR odwraca wszystkie bity w drugim słowie (uzupełnienie jedynek).

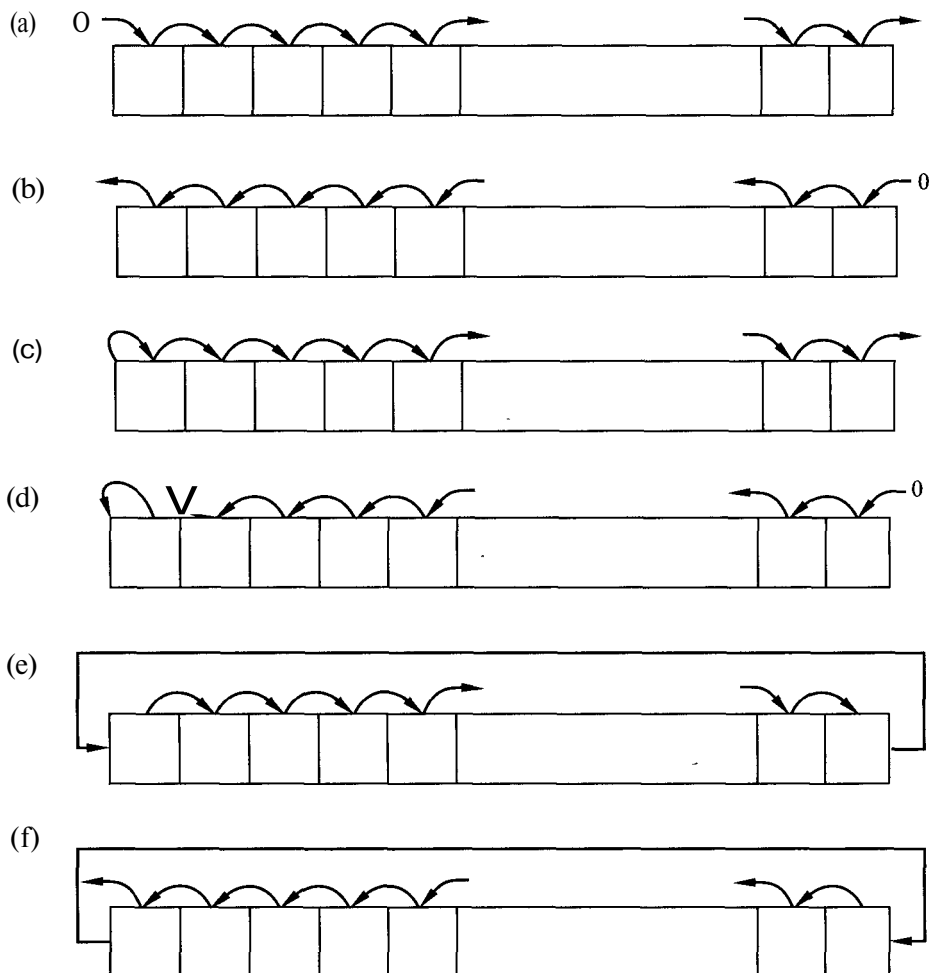
Poza bitowymi operacjami logicznymi większość maszyn umożliwia wiele funkcji przesuwania i rotacji. Najbardziej podstawowe operacje są przedstawione na rys. 9.5. Operacja **przesunięcia logicznego** powoduje przesunięcie bitów słowa w lewo lub w prawo. Z jednej strony jeden bit jest tracony, z drugiej wprowadzane jest 0. Przesunięcia logiczne są używane głównie do izolowania pola wewnątrz słowa. Zera wprowadzane do słowa zastępują niepożądaną informację, która jest wyprowadzana.

Założmy na przykład, że chcemy transmitować znaki do urządzenia wejścia-wyjścia metodą znak po znaku. Jeśli każde słowo pamięci ma długość 16 bitów i zawiera 2 znaki, to musimy „odpakować” znaki, zanim zostaną one wysłane. W celu wysłania 2 znaków zawartych w słowie są potrzebne następujące działania:

1. Załadowanie słowa do rejestru.
2. Wykonanie operacji AND z wartością 1111111100000000. Maskuje to znak po prawej.
3. Przesunięcie w prawo osiem razy. Powoduje to przesunięcie pozostałego znaku do prawej połowy rejestru.
4. Wykonanie operacji wejścia-wyjścia. Moduł wejścia-wyjścia odczyta 8 najmłodszych bitów z magistrali danych.

W wyniku wykonania tych czynności zostanie wysłany lewy znak. W celu wysłania prawego znaku należy:

1. Załadować słowo ponownie do rejestru.
2. Wykonać AND z 0000000011111111.
3. Wykonać operację wejścia-wyjścia.



RYSUNEK 9.5. Operacje przesunięcia i rotacji: (a) logiczne przesunięcie w prawo; (b) logiczne przesunięcie w lewo; (c) arytmetyczne przesunięcie w prawo; (d) arytmetyczne przesunięcie w lewo; (e) rotacja w prawo; (f) rotacja w lewo

Przy wykonaniu operacji **przesunięcia arytmetycznego** są dane traktowane jako liczby całkowite ze znakami; bit znaku nie jest przesuwany. W przypadku przesunięcia arytmetycznego w prawo, bit znaku jest zwykle replikowany w najbliższej pozycji bitowej na prawo. Operacje te mogą przyspieszać pewne operacje arytmetyczne. W odniesieniu do liczb w notacji uzupełnienia do dwóch przesunięcie w lewo lub w prawo odpowiada mnożeniu lub dzieleniu przez 2 pod warunkiem, że nie występuje przepełnienie lub niedomiar.

Operacje **rotacji lub przesunięcia cyklicznego** zachowują wszystkie bity, na których są przeprowadzane. Jednym z możliwych zastosowań rotacji jest sukcesywne doprowadzanie każdego bitu do skrajnej lewej pozycji, gdzie może

on być zidentyfikowany przez sprawdzenie znaku danych (traktowanych jako liczba).

Podobnie jak operacje arytmetyczne, operacje logiczne angażują ALU i mogą obejmować operacje przesyłania danych.

Konwersja

Rozkazy konwersji są rozkazami, które zmieniają format lub operują na formatach danych. Przykładem jest konwersja liczb z dziesiętnych na binarne.

Przykładem bardziej złożonego rozkazu redagowania jest rozkaz tłumaczenia (*Translate*, TR) stosowany w S/370. Może on być użyty do konwersji jednego kodu 8-bitowego na inny i operuje na trzech argumentach:

TR R1, R2, L

Argument R2 zawiera adres początkowy tablicy kodów 8-bitowych. Tłumaczone są bajty L począwszy od adresu określonego w R1, przy czym każdy bajt jest zastępowany przez zawartość wpisu w tablicy indeksowanego przez ten bajt. Na przykład, w celu tłumaczenia z EBCDIC na ASCII najpierw tworzymy 256-bajtową tablicę w lokacjach pamięci oznaczonych - powiedzmy $-1000 \div 10FF$ w notacji szesnastkowej. Tablica zawiera znaki w kodzie ASCII w ciągu według reprezentacji binarnej kodu EBCDIC; oznacza to, że kod ASCII jest umieszczany w tablicy w położeniu względnym równym binarnej wartości kodu EBCDIC tego samego znaku. Wobec tego, lokacje $10F0$ do $10F9$ będą zawierały wartości od 30 do 39, ponieważ FO jest kodem EBCDIC cyfry O, a 30 jest kodem ASCII tej samej cyfry (itd. aż do cyfry 9). Załóżmy teraz, że mamy kod EBCDIC dla cyfr 1984 począwszy od lokacji 2100 i chcemy go przełożyć na kod ASCII. Przyjmijmy, co następuje:

Lokacje $2100 \div 2103$ zawierają F1 F9 F8 F4

R1 zawiera 2100

R2 zawiera 1000.

Jeśli zrealizujemy operację

TR R1, R2, 4

to lokacje $2100 \div 2103$ będą zawierały 31 39 38 34.

Operacje wejścia-wyjścia

Rozkazy wejścia-wyjścia dość szczegółowo omówiliśmy w rozdz. 6. Jak widzieliśmy, jest stosowanych wiele rozwiązań, obejmujących izolowane programowane wejście-wyjście, programowane wejście-wyjście odwzorowane w pamięci, DMA oraz procesory wejścia-wyjścia. W wielu implementacjach przewidziano tylko niewiele rozkazów wejścia-wyjścia, przy czym poszczególne działania są określane za pomocą parametrów, kodów lub słów rozkazu.

Sterowanie systemowe

Rozkazy sterowania systemowego są na ogół rozkazami uprzywilejowanymi, które mogą być wykonywane tylko wtedy, gdy procesor znajduje się w pewnym stanie uprzywilejowanym lub gdy realizuje on program w specjalnym, uprzywilejowanym obszarze pamięci. Zwykle rozkazy te są zarezerwowane do użytku systemu operacyjnego.

Przykłady operacji sterowania systemowego opiszemy dalej. Rozkaz sterowania systemowego może czytać lub zmieniać zawartość rejestru sterowania; rejestry sterowania omówimy w rozdz. 11. Inny rozkaz może czytać lub modyfikować klucz ochrony pamięci, taki jak stosowany w systemie pamięciowym S/370. Jeszcze inny może przetwarzać bloki sterowania w systemie wieloprogramowym.

Przekazywanie sterowania

W przypadku wszystkich dotąd przedyskutowanych rodzajów rozkazów, rozkaz precyzuje operację, która ma być przeprowadzona, oraz argumenty. Domyślnie następny rozkaz przewidziany do wykonania jest to ten rozkaz, który następuje w pamięci bezpośrednio po właśnie wykonywanym rozkazie. Wobec tego, przy normalnym biegu zdarzeń, wykonywane są kolejne rozkazy z pamięci. Procesor po prostu powoduje elementarny przyrost (inkrementację) stanu licznika programu przed pobraniem następnego rozkazu.

Jednak znaczna część poleceń dowolnego programu ma za zadanie zmienić sekwencję wykonywania rozkazów. W wyniku tego procesor aktualizuje licznik programu, aby zawierał on w pamięci adres pożądanego rozkazu.

Istnieje wiele przyczyn, dla których wymagane są operacje przekazywania sterowania. Do najważniejszych należą:

1. Podczas praktycznego używania komputerów podstawowe znaczenie ma możliwość wykonywania każdego rozkazu więcej niż raz, a być może nawet wiele tysięcy razy. Są zastosowania wymagające tysięcy lub nawet milionów rozkazów. Jest nie do pomyślenia odrębne wypisywanie każdego rozkazu. Jeśli ma być przetwarzana tablica lub lista jednostek danych, wymagana jest pętla programowa. Jedna sekwencja rozkazów jest wykonywana powtarzalnie, aż będą przetworzone wszystkie dane.
2. Praktycznie wszystkie programy obejmują podejmowanie decyzji. Chcemy, żeby komputer wykonywał określone działania w pewnych warunkach, zaś inne działania w innych warunkach. Przyjmijmy na przykład, że sekwencja rozkazów powoduje obliczanie pierwiastka kwadratowego pewnej liczby. Na początku sekwencji badany jest znak liczby. Jeśli liczba jest ujemna, obliczenia nie są przeprowadzane i sygnalizowane jest wystąpienie błędu.
3. Prawidłowe ułożenie dużego lub nawet średniego programu komputerowego jest wyjątkowo trudnym zadaniem. Jest korzystne, jeśli dysponujemy

sposobami podziału takiego zadania na mniejsze części, nad którymi można pracować oddzielnie.

Przedyskutujemy teraz operacje przekazywania sterowania najczęściej spotykane w listach rozkazów:

- rozgałęzienie,
- pominięcie,
- a wywołanie podprogramu standardowego.

Rozkazy rozgałęzienia

Jednym z argumentów rozkazu rozgałęzienia jest adres następnego rozkazu przewidzianego do wykonania. Najczęściej mamy do czynienia z rozkazem **rozgałęzienia warunkowego**. Oznacza to, że rozgałęzienie (aktualizacja licznika programu w postaci wyrównania z adresem występującym w argumente) jest wykonywane tylko wtedy, gdy spełniony jest pewien warunek. W przeciwnym razie jest wykonywany następny rozkaz w sekwencji (następuje jak zwykle inkrementacja stanu licznika programu).

Istnieją dwie powszechnie stosowane metody generowania warunków, które mają być testowane w rozkazach rozgałęzienia warunkowego. Po pierwsze, większość maszyn przewiduje 1-bitowy lub wielobitowy kod warunkowy, który jest ustalany jako wynik pewnych operacji. Na przykład operacja arytmetyczna (DODAJ, ODEJMIJ itd.) może ustalić 2-bitowy kod warunkowy, który może przyjmować jedną z czterech wartości: 0, dodatni, ujemny, przepełnienie. W takiej maszynie mogłyby występować cztery różne rozkazy rozgałęzienia warunkowego:

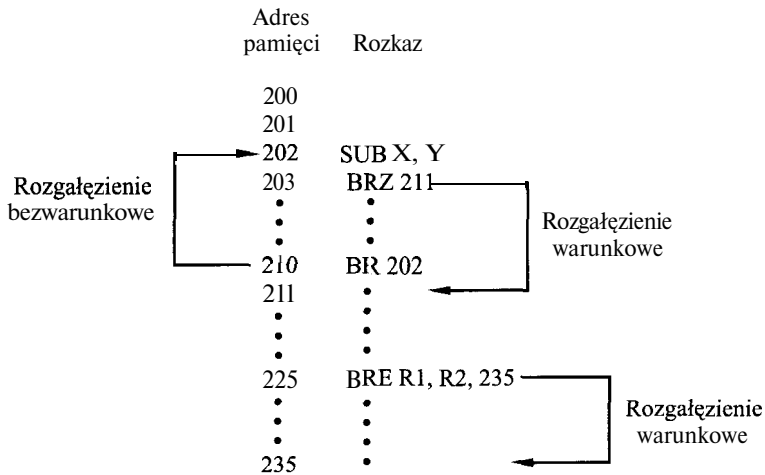
BRP X	Rozgałęzienie do lokacji X, jeśli wynik jest dodatni.
BRN X	Rozgałęzienie do lokacji X, jeśli wynik jest ujemny.
BRZ X	Rozgałęzienie do lokacji X, jeśli wynik jest zerem.
BRO X	Rozgałęzienie do lokacji X, jeśli występuje przepełnienie.

We wszystkich tych przypadkach wynik, o którym mowa, jest wynikiem ostatnio wykonanej operacji, w ramach której został też ustalony kod warunku.

Innym rozwiązaniem, które może występować łącznie z 3-adresowym formatem rozkazu, jest przeprowadzenie porównania i określenie rozgałęzienia w ramach jednego rozkazu. Na przykład:

BRE R1, R2, X Rozgałęzienie do X, jeśli zawartość R1 = zawartość R2.

Na rysunku 9.6 są pokazane przykłady tych operacji. Zauważmy, że rozgałęzienie może następować *do przodu* (do rozkazu o wyższym adresie) lub *wstecz* (do niższego adresu). Przykład pokazuje, jak można użyć rozgałęzień bezwarunkowych i warunkowych w celu utworzenia powtarzającej się pętli rozkazów. Rozkazy w lokacjach 202 do 210 będą wykonywane powtarzalnie, aż wynikiem odejmowania Y od X będzie 0.



RYSUNEK 9.6. Rozkazy rozgałęzienia

Rozkazy pominięcia

Inną często spotykaną formą rozkazu przekazania sterowania jest rozkaz pominięcia. Rozkaz pominięcia zawiera adres domyślny. Zwykle rozkaz pominięcia oznacza pominięcie jednego rozkazu; wobec tego adres domyślny jest równy adresowi następnego rozkazu plus długość jednego rozkazu.

Ponieważ rozkaz pominięcia nie wymaga określania adresu miejsca przeznaczenia, może on zmieścić dodatkowe działania. Typowym przykładem jest rozkaz inkrementacja-i-pominięcie-jeśli-zero (ISZ). Rozważmy następujący fragment programu:

```

301
.
.
.
309  ISZ    R1
310  BR     301
311

```

W tym fragmencie użyto dwóch rozkazów przekazania sterowania w celu zrealizowania pętli iteracyjnej. Wartość R1 jest ustalana za pomocą zanegowanej liczby iteracji, które mają być przeprowadzone. Na jednym końcu pętli następuje inkrementacja R1. Jeśli R1 nie jest równe zero, to program rozgałęzia się do tyłu, na początek pętli. W przeciwnym razie rozgałęzienie jest pomijane, a program jest kontynuowany przez wykonywanie rozkazu następującego po zakończeniu pętli.

Rozkazy wywołania podprogramu standardowego

Być może najważniejszą innowacją w rozwoju języków programowania jest **podprogram standardowy** (ang. *subroutine*). Podprogram standardowy jest stanowiącym całość programem komputerowym, który jest wbudowywany do większego programu. Podprogram standardowy może być **wywołany** w dowolnym punkcie programu. Oznacza to, że w tym punkcie rozkazuje się komputerowi zrealizowanie całego podprogramu standardowego, a następnie powrót do punktu, w którym miało miejsce wywołanie.

Dwoma zasadniczymi powodami stosowania podprogramów standardowych są ekonomia i modułowość. Dzięki podprogramowi standardowemu ten sam fragment kodu może być użyty wielokrotnie. Jest to ważne dla ekonomii programowania oraz dla efektywniejszego wykorzystania przestrzeni pamięci w systemie (program musi być przechowywany). Podprogramy standardowe umożliwiają również podział dużych zadań programowania na mniejsze części. Tak rozumiana modułowość znacznie ułatwia programowanie.

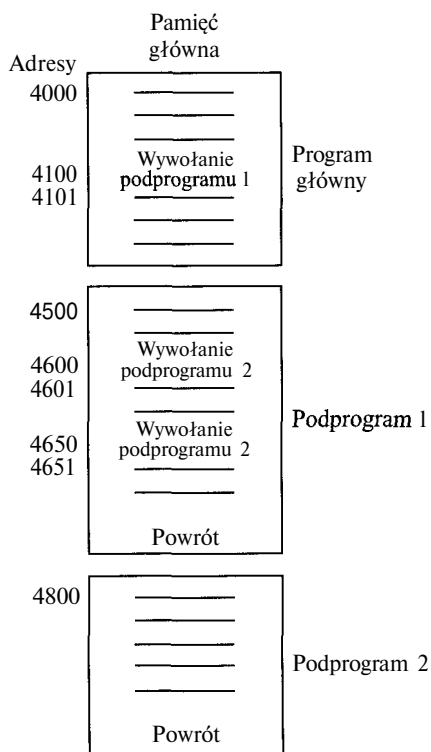
Mechanizm podprogramu standardowego obejmuje dwa podstawowe rozkazy: rozkaz wywołania, który powoduje skok z aktualnej lokacji do podprogramu, oraz rozkaz powrotu powodujący powrót od podprogramu do miejsca, w którym nastąpiło wywołanie. Oba te rozkazy należą do rozkazów rozgałęzień.

Na rysunku 9.7 jest pokazane użycie podprogramów standardowych do budowy programu. W tym przykładzie istnieje główny program, rozpoczynający się od lokacji 4000. Program ten zawiera wywołanie podprogramu SUB1, rozpoczynającego się od lokacji 4500. Gdy ma miejsce rozkaz wywołania, procesor zawieszając realizację programu głównego i rozpoczyna wykonywanie SUB1 przez pobranie następnego rozkazu z lokacji 4500. Wewnątrz SUB1 istnieją dwa wywołania SUB2 w lokacji 4800. W każdym przypadku wykonywanie SUB1 jest zawieszane i wykonywany jest SUB2. Dyrektywa RETURN (powrót) powoduje powrót procesora do programu wywołującego i kontynuowanie jego realizacji począwszy od rozkazu następnego po wywołaniu (CALL). Zachowanie to jest zilustrowane na rys. 9.8.

Warto odnotować kilka uwag:

1. Podprogram standardowy może być wywołany z więcej niż jednej lokacji.
2. Wywołanie podprogramu standardowego może nastąpić również wewnątrz podprogramu standardowego. Umożliwia to **zagnieżdżanie** (ang. *nesting*) podprogramów na dowolnej głębokości.
3. Każdemu wywołaniu podprogramu towarzyszy rozkaz powrotu zawarty w wywołanym podprogramie.

Ponieważ pożądana jest możliwość wywoływania podprogramu standardowego z wielu punktów, procesor musi zachowywać adres powrotu. Najczęściej używane są trzy miejsca przechowywania adresu powrotu:



RYSUNEK 9.7. Podprogramy zagnieżdżone

- a rejestr,
- a początek podprogramu,
- a wierzchołek stosu.

Rozważmy rozkaz w języku maszynowym **CALL X**, który oznacza *wywołaj podprogram standardowy w lokacji X*. Jeśli używa się rejestru, **CALL X** powoduje następujące działania:

$$RN \leftarrow PC + A$$

$$PC \leftarrow X$$

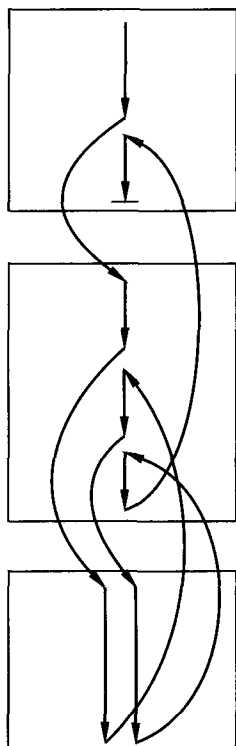
gdzie **RN** jest rejestrem, który zawsze jest używany do tego celu, **PC** jest licznikiem programu, a **A** - długością rozkazu. Wywołany podprogram może teraz zachować zawartość **RN** do użycia przy późniejszym powrocie.

Drugą możliwością jest przechowywanie adresu powrotu na początku podprogramu. W tym przypadku **CALL X** powoduje:

$$X \leftarrow PC + A$$

$$PC \leftarrow X + 1$$

Jest to całkiem poręczne rozwiązanie. Adres powrotu został bezpiecznie zachowany.



RYСУNEK 9.8. Sekwencja wykonywania zagnieżdżonych podprogramów z rys. 9.7

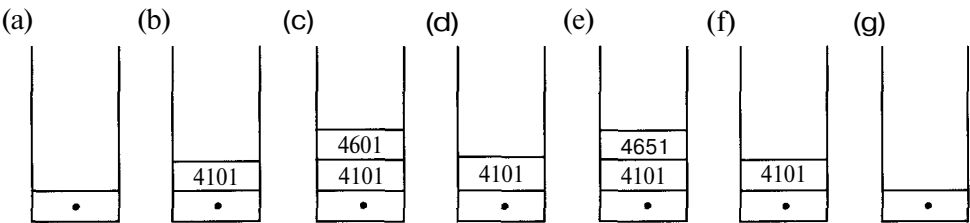
Oba przedstawione rozwiązania działają i były używane. Jedynym ich ograniczeniem jest to, że uniemożliwiają one używanie podprogramów **wielowejściowych** (ang. *reentrant*) [HIGM67], [POHL81]. Podprogram wielowejściowy umożliwia jednocześnie otwarcie kilku wywołań. Przykładem zastosowania tej własności jest procedura rekurencyjna (ang. *recursive procedure*) [BARR68], [BURG75].

Ogólniejszym i stwarzającym wiele możliwości rozwiązaniem jest użycie stosu (definicja stosu znajduje się w dodatku 9A). Gdy procesor realizuje wywołanie, umieszcza adres powrotu na stosie. Gdy wykonuje powrót, używa adresu ze stosu. Na rysunku 9.9 jest pokazane użycie stosu.

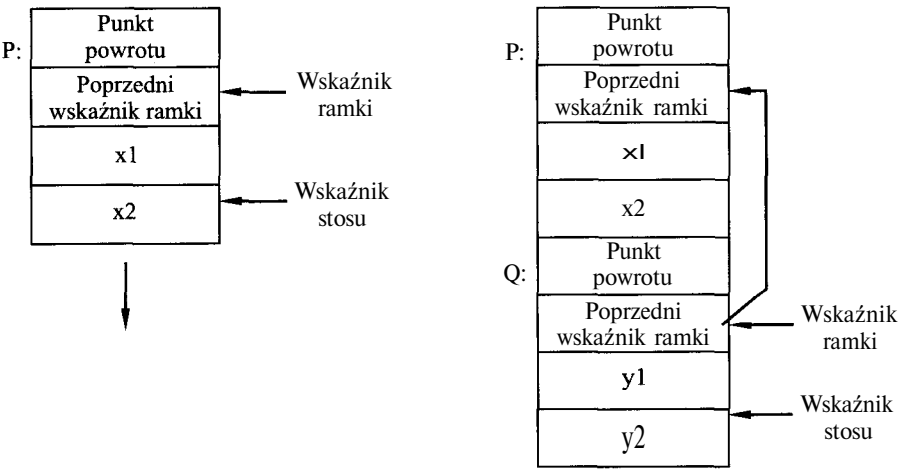
Poza zachowaniem adresu powrotu często wraz z wywołaniem podprogramu konieczne jest również przekazywanie parametrów. Mogą one być przekazywane w rejestrach. Inną możliwością jest przechowywanie parametrów w pamięci bezpośrednio po rozkazie wywołania. W takim przypadku powrót musi następować do lokacji poza parametrami. Oba te rozwiązania mają wady. Jeśli używane są rejestry, program wywołany i program wywołujący muszą być napisane tak, żeby rejestry były właściwie używane. Przechowywanie parametrów w pamięci utrudnia wymianę zmiennej liczby parametrów. Poza tym oba rozwiązania uniemożliwiają użycie podprogramów wielowejściowych.

Wykorzystując stos, można opracować bardziej elastyczne podejście do przekazywania parametrów. Gdy procesor realizuje wywołanie, przekazuje na stos nie tylko adres powrotu, ale także parametry, które mają być przekazane do podprogramu. Wywołany podprogram może mieć dostęp do parametrów ze stosu. W c/asie powrotu parametry powrotne też mogą być umieszczone na stosie, pod adresem powrotu. Cały zestaw parametrów, łącznie z adresem powrotu, który jest przechowywany w celu wywołania procedury, określany jest jako **ramka stosu** (ang. *stack frame*).

Przykład takiego rozwiązania jest pokazany na rys. 9.10. Odnosi się on do procedury P, w której są opisane zmienne lokalne x1 i x2, oraz procedury Q, która może być wywołana przez P i w której są opisane zmienne lokalne y1 i y2. Na tym rysunku punktem powrotu każdej procedury jest pierwsza jednostka przechowywana w odpowiedniej ramce stosu. Następną przechowywaną jednostką jest wskaźnik początku poprzedniej ramki. Jest to konieczne, jeśli liczba parametrów na stosie lub ich długość są zmienne.



RYSUNEK 9.9. Użycie stosu do implementacji zagnieżdżonych podprogramów z rys. 9.7: (a) początkowa zawartość stosu; (b) po wywołaniu podprogramu 1; (c) po wywołaniu podprogramu 2; (d) po powrocie; (e) po wywołaniu podprogramu 2; (f) po powrocie; (g) po powrocie



RYSUNEK 9.10. Zwiększenie ramki stosu przy zastosowaniu przykładowych procedur P i Q [DEWA90]

Rodzaje operacji Pentium

Dla Pentium przewidziano złożony zestaw rodzajów operacji, łącznie z pewną liczbą rozkazów specjalnych. Intencją było dostarczenie narzędzi autorom programów kompilujących w celu optymalnego tłumaczenia programów w języku wysokiego poziomu na język maszynowy. W tabeli 9.7. są wymienione rodzaje operacji wraz z przykładami. Większość z nich to konwencjonalne rozkazy, jakie można znaleźć w większości list rozkazów. Jednak kilka rodzajów rozkazów dostosowano do architektury 80×86/Pentium i są one szczególnie interesujące.

TABELA 9.7. Rodzaje operacji procesora Pentium (wraz z przykładami typowych operacji)

Rozkaz	Opis
<i>Przenoszenie danych</i>	
MOV	Przeniesienie argumentu między rejestrami lub między rejestrem a pamięcią.
PUSH	Umieszczenie argumentu na stosie.
PUSHA	Umieszczenie zawartości wszystkich rejestrów na stosie.
MOVSX	Przeniesienie bajta, słowa, podwójnego słowa z rozszerzeniem znaku. Przenoszony jest bajt do słowa lub słowo do podwójnego słowa z rozszerzeniem znaku w notacji uzupełnienia do dwóch.
LEA	Ładowanie adresu efektywnego. Ładowane jest wyrównanie argumentu źródłowego - a nie jego wartość - do miejsca argumentu docelowego.
XLAT	Translacja zapisów tablicy przeglądowej. Bajt w AL jest zastępowany przez bajt z tablicy adresowej kodowanej przez użytkownika. Gdy wykonywany jest XLAT, AL powinna mieć indeks bez znaku w stosunku do tablicy. XLAT zmienia zawartość AL z indeksu tablicy na zapis w tablicy.
IN, OUT	Wejście lub wyjście argumentu z przestrzeni wejścia-wyjścia.
<i>Arytmetyczne</i>	
ADD	Obliczenie sumy argumentów.
SUB	Obliczenie różnicy argumentów.
MUL	Obliczanie iloczynu z podwójną lub pojedynczą dokładnością.
IDIV	Obliczenie ilorazu.
<i>Logiczne</i>	
AND	Wykonanie operacji AND na argumentach.
BTS	Testowanie i ustawienie bitu. Operuje na polu bitowym argumentu. Kopiowana jest bieżąca wartość bitu do znacznika CF, a oryginalny bit jest ustawiany na 1.
BSF	Sprawdzanie bitów w przód. Sprawdzane jest słowo lub podwójne słowo w poszukiwaniu bitu 1 a numer pierwszego bitu 1 jest zapisywany w rejestrze.
SHL/SHR	Przesunięcie logiczne w lewo lub w prawo.
SAL/SAR	Przesunięcie arytmetyczne w lewo lub w prawo.
ROL/ROR	Wykonanie rotacji w lewo lub w prawo.
SETcc	Ustawienie bajta na zero lub na 1 zależnie od dowolnego z 16 warunków określonych przez znaczniki stanu.
<i>Przekazywanie sterowania</i>	
JMP	Skok bezwarunkowy.
CALL	Przeniesienie sterowania do innej lokacji. Przed przeniesieniem adres rozkazu następującego po CALL jest umieszczany na stosie.
JE/JZ	Wykonanie skoku, jeśli równy (lub jeśli zero).

TABELA 9.7. cd.

Rozkaz	Opis
LOOPE/ LOOPZ	Wykonanie pętli, jeśli równy (lub jeśli zero). Jest to skok warunkowy z wykorzystaniem wartości zapisanej w rejestrze ECX. Rozkaz najpierw powoduje dekrementację ECX przed testowaniem ECX pod kątem warunku rozgałęzienia.
INT/INTO	Przerwanie (lub przerwanie w razie przepełnienia). Przeniesienie sterowania do podprogramu obsługi przerwań.
<i>Operacje łańcuchowe</i>	
MOVS	Przeniesienie bajta, słowa lub podwójnego z łańcucha. Rozkaz operuje na jednym elemencie łańcucha, indeksowanym przez rejestry ESI i EDI. Po każdej operacji na łańcuchu rejestry te są automatycznie inkrementowane lub dekrementowane, aby wskazywały następny element łańcucha.
LODS	Ładowanie bajta, słowa lub podwójnego słowa z łańcucha.
<i>Wspieranie języka wysokiego poziomu</i>	
ENTER	Utworzenie ramki stosu, która może być użyta do wdrażania reguł języka wysokiego poziomu zorientowanego blokowo.
LEAVE	Odwrócenie działania uprzedniego ENTER.
BOUND	Sprawdzenie granicy tablicy. Sprawdzenie, czy wartość argumentu 1 znajduje się wewnątrz granic. Granice są zawarte w dwóch sąsiednich lokacjach pamięci wskazanych przez argument 2. Jeśli wartość ta nie mieści się w granicach, następuje przerwanie. Rozkaz jest używany do sprawdzania indeksu tablicy.
<i>Sterowanie za pomocą znaczników</i>	
STC	Ustawienie znacznika przeniesienia.
LAHF	Ładowanie rejestru A ze znaczników stanu. Kopiuje bity SF, ZF, AF, PF i CF do rejestru A.
<i>Rejestr segmentowy</i>	
LDS	Ładowanie wskaźnika do rejestru segmentowego D.
<i>Sterowanie systemowe</i>	
HLT	Zatrzymanie.
LOCK	Potwierdzenie utrzymania pamięci wspólnej. Dzięki temu Pentium ma wyłączne prawo używania jej podczas rozkazu, który następuje bezpośrednio po LOCK.
ESC	Ucieczka związana z rozszerzeniem procesora. Kod ucieczki, który wskazuje, że następne rozkazy mają być wykonywane przez koprocesor numeryczny, wspierający obliczenia całkowitoliczbowe i zmiennopozycyjne o dużej dokładności.
WAIT	Czekanie aż BUSY# będzie zanegowane. Wykonywanie programu jest zawieszane przez Pentium, aż procesor wykryje, że końcówka BUSY nie jest aktywna, co wskazuje, że koprocesor numeryczny zakończył obliczenia.
<i>Ochrona</i>	
SGDT	Zapisanie globalnej tablicy deskryptorów.
LSL	Ładowanie granicy segmentu do rejestru określonego przez użytkownika.
VERR/ VERW	Weryfikowanie segmentu w przypadku odczytu/zapisu.
<i>Zarządzanie pamięcią podręczną</i>	
INVD	Opróżnienie wewnętrznej pamięci podręcznej.
WBINVD	Opróżnienie wewnętrznej pamięci podręcznej po zapisaniu w pamięci „zanieczyszczonych” wierszy.
INVLPG	Unieważnienie zapisu w buforze translacji adresów tablic stron.

Rozkazy wywołania/powrotu

Pentium przewiduje cztery rozkazy obsługujące procedurę wywołania i powrotu: CALL, ENTER, LEAVE i RETURN. Pouczające jest zapoznanie się z ich funkcjonowaniem¹. Przypomnijmy sobie na podstawie rys. 9.10, że typowym środkiem wdrażania procedury wywołanie/powrót jest użycie ramek stosu. Gdy wywoływana jest nowa procedura, muszą być wykonane następujące działania podczas przechodzenia do nowej procedury:

- a Umieszczenie punktu powrotu na stosie.
- a Umieszczenie bieżącego wskaźnika ramki na stosie.
- a Skopiowanie wskaźnika stosu jako nowej wartości wskaźnika ramki.
- a Skorygowanie wskaźnika stosu w celu alokacji ramki.

Rozkaz CALL powoduje umieszczenie bieżącej wartości wskaźnika rozkazu na stosie oraz skok do początkowego punktu procedury przez umieszczenie adresu tego punktu we wskaźniku rozkazów. W maszynach 8088 i 8086 typowa procedura rozpoczynała się od sekwencji

```
PUSH    EBP
MOV     EBP, ESP
SUB     ESP, space_for_locals
```

gdzie EBP jest wskaźnikiem ramki, a ESP - wskaźnikiem stosu. W procesorze 80286 i w późniejszych procesorach rozkaz ENTER powodował wykonanie wszystkich powyższych operacji za pomocą jednego rozkazu.

Rozkaz ENTER został dodany do listy rozkazów w celu zapewnienia bezpośredniego wsparcia kompilatora. Rozkaz ten umożliwia realizację tzw. procedur zagnieżdżonych (*nested procedures*) w takich językach, jak Pascal, COBOL i Ada (nie występujących w C i w FORTRAN-ie). Okazało się, że istnieją lepsze sposoby postępowania z procedurami zagnieżdżonymi dla tych właśnie języków. Ponadto, chociaż rozkaz ENTER oszczędza kilka bajtów pamięci w porównaniu z sekwencją PUSH, MOV, i SUB (4 bajty wobec 2 bajtów), w rzeczywistości wydłuża czas realizacji (11 cykli zegara wobec 1,5 cyklu). Chociaż więc wydawało się dobrym pomysłem uzupełnienie listy rozkazów przez ENTER, skomplikowało to implementację procesora, wnosząc przy tym niewielkie lub żadne korzyści. Zobaczmy, że w przeciwieństwie do tego rozwiązanie RISC przy projektowaniu procesora zapobiega tak złożonym rozkazom, jak ENTER, umożliwiając przy tym efektywniejszą implementację za pomocą sekwencji prostszych rozkazów.

Zarządzanie pamięcią

Inny zestaw rozkazów specjalnych dotyczy segmentowania pamięci. Są to rozkazy uprzywilejowane, które mogą być wykonywane tylko przez system operacyjny. Umożliwiają one ładowanie i czytanie lokalnych oraz globalnych tablic

¹ Omówienie to jest oparte na [DEWA90], gdzie można znaleźć bardziej szczegółową analizę.

segmentów (zwanych tablicami deskryptorów) oraz na sprawdzanie i zmienianie poziomu uprzywilejowania segmentów.

Rozkazy specjalne dotyczące wewnętrznej pamięci podręcznej zostały omówione w rozdz. 5.

Kody warunkowe

Wiemy już, że kody warunkowe są bitami w specjalnych rejestrach, które mogą być ustawiane przez pewne operacje i używane w rozkazach rozgałęzienia warunkowego. Warunki te są ustalane przez operacje arytmetyczne i porównania. Operacja porównania w większości języków polega na odjęciu dwóch argumentów, podobnie jak w operacji odejmowania. Różnica polega na tym, że operacja porównania ustala tylko kody warunkowe, podczas gdy operacja odejmowania zapisuje również wynik odejmowania w miejscu przeznaczenia.

W tabeli 9.8 są wymienione kody warunkowe stosowane w Pentium. Każdy warunek lub kombinacja warunków mogą być testowane pod kątem rozgałęzienia warunkowego. W tabeli 9.9 są podane kombinacje warunków, dla których zdefiniowano kody operacyjne rozgałęzień warunkowych.

TABELA 9.8. Kody warunkowe procesora Pentium

Bit stanu	Nazwa	Opis
C	Carry	wskazuje przeniesienie lub przeniesienie zanegowane do lewej pozycji bitowej w wyniku operacji arytmetycznej. Modyfikowany również przez niektóre operacje przesunięcia lub rotacji.
P	Parity	parzystość wyniku operacji arytmetycznej lub logicznej. 1 wskazuje parzystość, 0 - nieparzystość.
A	Auxiliary Carry	reprezentuje przeniesienie lub przeniesienie zanegowane między półbajtami w wyniku 8-bitowej operacji arytmetycznej lub logicznej z użyciem rejestru AL.
Z	Zero	wskazuje, że wynik operacji arytmetycznej lub logicznej jest równy 0.
S	Sign	wskazuje znak wyniku operacji arytmetycznej lub logicznej.
O	Overflow	wskazuje przepełnienie arytmetyczne po dodawaniu lub odejmowaniu.

Na podstawie tej listy można poczynić kilka interesujących spostrzeżeń. Po pierwsze, może być potrzebne sprawdzenie dwóch argumentów w celu stwierdzenia, czy jedna liczba jest większa od drugiej. Zależy to jednak od tego, czy liczby te mają znaki. Na przykład, 8-bitowa liczba 11111111 jest większa od 00000000, jeśli obie liczby są interpretowane jako bezznakowe liczby całkowite ($255 > 0$), jest jednak mniejsza, jeśli są traktowane jako 8-bitowe liczby w notacji uzupełnienia do dwóch ($-1 < 0$). Wiele języków assemblerowych wprowadza dlatego dwa zestawy wyrażeń w celu odróżnienia obu przypadków: jeśli porównujemy dwie liczby jako liczby całkowite ze znakiem, używamy wyrażeń *większa niż* lub *mniejsza niż*; jeśli natomiast porównujemy je jako liczby całkowite bezznakowe, używamy wyrażeń *poniżej* i *ponad*.

TABELA 9.9. Stosowane w Pentium warunki dla skoków warunkowych i rozkazów SETcc

Symbol	Testowany warunek	Komentarz
A, NBE	$C = 0 \text{ AND } Z = 0$	ponad; nie poniżej lub równe (większa niż, bez znaku)
AE, NB, NC	$C = 0$	ponad lub równe; nie poniżej (większa niż lub równa, bez znaku); nie przeniesienie
B, NAE, C	$C = 1$	poniżej; nie powyżej lub równe (mniejsza niż, bez znaku); ustawione przeniesienie
BE, NA	$C = 1 \text{ OR } Z = 1$	poniżej lub równe; nie powyżej (mniejsza lub równa, bez znaku)
E, Z	$Z = 1$	równe; zero (ze znakiem lub bez znaku)
G, NLE	$[(S = 1 \text{ AND } O = 1) \text{ OR } (S = 0 \text{ AND } O = 0)]$	większa niż; nie mniejsza lub równa (ze znakiem)
GE, NL	$(S = 1 \text{ AND } O = 1) \text{ OR } (S = 0 \text{ AND } O = 0)$	większa niż lub równa; nie mniejsza niż (ze znakiem)
L, NGE	$(S = 1 \text{ AND } O = 0) \text{ OR } (S = 0 \text{ AND } O = 1)$	mniejsza niż; nie większa lub równa (ze znakiem)
LE, NG	$(S = 1 \text{ AND } O = 0) \text{ OR } (S = 0 \text{ AND } O = 1) \text{ OR } (Z = 1)$	mniejsza niż lub równa; nie większa (ze znakiem)
NE, NZ	$Z = 0$	nie równa; nie zero (ze znakiem lub bez znaku)
NO	$O = 0$	brak przepełnienia
NS	$S = 0$	nie znak (nie ujemna)
NP, PO	$P = 0$	brak parzystości; nieparzystość
O	$O = 1$	przepełnienie
P	$P = 1$	parzystość
S	$S = 1$	znak (ujemna)

Druga operacja wiąże się ze złożonością porównywania liczb całkowitych ze znakami. Wynik ze znakiem jest większy lub równy zeru, jeżeli (a) bit znaku jest zerem i nie ma przepełnienia ($S = O$ oraz $O = 0$), lub (b) bit znaku jest jedynką i występuje przepełnienie. Analiza rys. 8.4. powinna przekonać czytelnika, że warunki określone dla różnych operacji ze znakami są właściwe (patrz problem 9.14.).

Rodzaje operacji PowerPC

PowerPC zapewnia duży zbiór rodzajów operacji. W tabeli 9.10 są wymienione te rodzaje i odpowiednie przykłady. Warto zwrócić uwagę na kilka własności.

TABELA 9.10. Rodzaje operacji procesora PowerPC (z przykładami typowych operacji)

Rozkaz	Opis
<i>Zorientowane na rozgałęzienia</i>	
b	Rozgałęzienie bezwarunkowe.
bl	Rozgałęzienie do adresu docelowego i umieszczenie adresu efektywnego rozkazu następującego po rozgałęzieniu w rejestrze powiązań.
bc	Rozgałęzienie warunkowe na podstawie rejestru zliczającego i/lub bitu w rejestrze warunkowym.

TABELA 9.9. Stosowane w Pentium warunki dla skoków warunkowych i rozkazów SETcc

Symbol	Testowany warunek	Komentarz
A, NBE	$C = 0 \text{ AND } Z = 0$	ponad; nie poniżej lub równe (większa niż, bez znaku)
AE, NB, NC	$C = 0$	ponad lub równe; nie poniżej (większa niż lub równa, bez znaku); nie przeniesienie
B, NAE, C	$C = 1$	poniżej; nie powyżej lub równe (mniejsza niż, bez znaku); ustawioneprzeniesienie
BE, NA	$C = 1 \text{ OR } Z = 1$	poniżej lub równe; nie powyżej (mniejsza lub równa, bez znaku)
E, Z	$Z = 1$	równe; zero (ze znakiem lub bez znaku)
G, NLE	$[(S = 1 \text{ AND } O = 1) \text{ OR } (S = 0 \text{ AND } O = 0)]$ $\text{AND } [Z = 0]$	większa niż; nie mniejsza lub równa (ze znakiem)
GE, NL	$(S = 1 \text{ AND } O = 1) \text{ OR } (S = 0 \text{ AND } O = 0)$	większa niż lub równa; nie mniejsza niż (ze znakiem)
L, NGE	$(S = 1 \text{ AND } O = 0) \text{ OR } (S = 0 \text{ AND } O = 1)$	mniejsza niż; nie większa lub równa (ze znakiem)
LE, NG	$(S = 1 \text{ AND } O = 0) \text{ OR } (S = 0 \text{ AND } O = 1)$ $\text{OR } (Z = 1)$	mniejsza niż lub równa; nie większa (ze znakiem)
NE, NZ	$Z = 0$	nie równa; nie zero (ze znakiem lub bez znaku)
NO	$O = 0$	brak przepełnienia
NS	$S = 0$	nie znak (nie ujemna)
NP, PO	$P = 0$	brak parzystości; nieparzystość
O	$O = 1$	przepełnienie
P	$P = 1$	parzystość
S	$S = 1$	znak (ujemna)

Druga operacja wiąże się ze złożonością porównywania liczb całkowitych ze znakami. Wynik ze znakiem jest większy lub równy zeru, jeżeli (a) bit znaku jest zerem i nie ma przepełnienia ($S = O$ oraz $O = 0$), lub (b) bit znaku jest jedynką i występuje przepełnienie. Analiza rys. 8.4. powinna przekonać czytelnika, że warunki określone dla różnych operacji ze znakami są właściwe (patrz problem 9.14.).

Rodzaje operacji PowerPC

PowerPC zapewnia duży zbiór rodzajów operacji. W tabeli 9.10 są wymienione te rodzaje i odpowiednie przykłady. Warto zwrócić uwagę na kilka własności.

TABELA 9.10. Rodzaje operacji procesora PowerPC (z przykładami typowych operacji)

Rozkaz	Opis
<i>Zorientowane na rozgałęzienia</i>	
b	Rozgałęzienie bezwarunkowe.
bl	Rozgałęzienie do adresu docelowego i umieszczenie adresu efektywnego rozkazu następującego po rozgałęzieniu w rejestrze powiązań.
bc	Rozgałęzienie warunkowe na podstawie rejestru zliczającego i/lub bitu w rejestrze warunkowym.

TABELA 9.10. cd.

Rozkaz	Opis
sc	Systemowe wywołanie obsługi systemu operacyjnego.
trap	Porównanie dwóch argumentów i uaktywnienie programu obsługi pułapek, jeśli spełnione są określone warunki.
<i>Ładuj/zapisz</i>	
lwzu	Łaładowanie słowa i rozszerzenie zerami do lewej; zaktualizowanie rejestru źródłowego.
ld	Łaładowanie podwójnego słowa.
lmw	Łaładowanie wielokrotnego słowa; ładowanie kolejnych słów do sąsiednich rejestrów z rejestru docelowego poprzez rejestr roboczy 31.
lswx	Łaładowanie łańcucha bajtów do rejestrów, rozpoczynając od rejestru docelowego; cztery bajty na rejestr; zawinięcie od rejestru 31 do rejestru 0.
<i>Arytmetyka całkowitoliczbowa</i>	
add	Obliczenie sumy zawartości dwóch rejestrów i umieszczenie w trzecim rejestrze.
subf	Obliczenie różnicy zawartości dwóch rejestrów i umieszczenie w trzecim rejestrze.
mullw	Obliczenie iloczynu 32-bitowych zawartości niskiego rzędu dwóch rejestrów i umieszczenie 64-bitowej iloczynu w trzecim rejestrze.
divd	Obliczenie ilorazu 64-bitowej zawartości dwóch rejestrów i umieszczenie wyniku w trzecim rejestrze.
<i>Logiczne i przesunięcia</i>	
cmp	Porównanie dwóch argumentów i ustawienie czterech bitów warunkowych w określonym polu rejestru warunkowego.
crand	Wykonanie operacji AND w rejestrze warunkowym; operacja AND dotyczy dwóch bitów rejestru warunkowego, wynik jest umieszczany w jednej z dwóch pozycji bitowych.
and	Wykonanie operacji AND na zawartości dwóch rejestrów i umieszczenie wyniku w trzecim rejestrze.
cntlzd	Policzenie liczby kolejnych zer, poczynając od bitu zero w rejestrze źródłowym, i umieszczenie wyniku w rejestrze docelowym.
rldic	Wykonanie rotacji w lewo rejestru podwójnego słowa, wykonanie operacji AND z maską i zapisanie wyniku w rejestrze docelowym.
sld	Przesunięcie w lewo bitów w rejestrze źródłowym i zapisanie w rejestrze docelowym.
<i>Zmiennopozycyjne</i>	
lfs	Łaładowanie 32-bitowej liczby zmiennopozycyjnej z pamięci, przetworzenie na format 64-bitowy i zapisanie w rejestrze zmiennopozycyjnym.
fadd	Obliczenie sumy zawartości dwóch rejestrów i umieszczenie wyniku w trzecim rejestrze.
fmadd	Obliczenie iloczynu zawartości dwóch rejestrów, dodanie zawartości trzeciego i umieszczenie wyniku w czwartym rejestrze.
fcmpu	Porównanie dwóch argumentów zmiennopozycyjnych i ustawienie bitów warunkowych.
<i>Zarządzanie pamięcią podręczną</i>	
dcbf	Opróżnienie bloku pamięci podręcznej danych; wykonanie przeglądu pamięci podręcznej pod kątem określonego adresu docelowego i przeprowadzenie operacji opróżniania.
icbi	Unieważnienie bloku pamięci podręcznej rozkazów.

Rozkazy dotyczące rozgałęzień

PowerPC obsługuje zwykłe przypadki rozgałęzień bezwarunkowych i warunkowych. Warunkowe rozkazy rozgałęzienia testują pojedynczy bit rejestru warunkowego (możliwości: „prawdziwy”, „fałszywy” lub „nie dbaj o to”) oraz wartość rejestru licznika (możliwości: „zero”, „nie-zero” lub „nie dbaj o to”). Jeśli rejestr licznika jest testowany pod kątem zero - nie-zero, to przed testem jego stan jest zmniejszany o 1. Jest to wygodne przy tworzeniu pętli iteracyjnych.

Rozkazy rozgałęzienia mogą także wskazywać, że adres lokacji następującej po rozgałęzieniu ma być umieszczony w rejestrze powiązania, opisanym w rozdz. 13. Ułatwia to przetwarzanie wywoływań i powrotów.

Rozkazy ładowania/zapisu

W architekturze procesora PowerPC tylko rozkazy ładowania i zapisu mają dostęp do pamięci; rozkazy arytmetyczne i logiczne są przeprowadzane tylko na rejestrach. Jest to charakterystyczne dla projektów RISC, co omówimy w rozdz. 12.

Istnieją dwie własności, które charakteryzują różne rozkazy ładowania i przechowywania. Są to:

- *Rozmiar danych.* Jednostkami transferu danych są bajt, półsłowo, słowo i podwójne słowo. Są również osiągalne rozkazy ładowania i zapisu łańcuchów bajtów do (i z) wielu rejestrów.
- Q *Rozszerzenie znaku.* W przypadku ładowania półsłów i słów nie używane bity znajdujące się w lewej części 64-bitowego rejestru przeznaczenia są albo wypełniane zerami, albo bitem znaku ładowanej wielkości.

9.4

JĘZYK ASEMBLEROWY

Procesor jest w stanie rozumieć i wykonywać rozkazy maszynowe. Rozkazy takie są po prostu liczbami binarnymi przechowywanymi w komputerze. Jeśli programista chciałby programować bezpośrednio w języku maszynowym, byłoby konieczne wprowadzanie programu w postaci danych binarnych.

Rozważmy prostą instrukcję w języku BASIC:

$$N = I + J + K$$

Założmy, że chcemy zaprogramować tę instrukcję w języku maszynowym oraz nadać zmiennym I, J i K wartość początkową odpowiednio 2, 3 i 4, co widać na rys. 9.1 la. Program rozpoczyna się od lokacji 101 (szesnastkowej).

(a)					(b)	
Adres		Zawartość			Adres	Zawartość
101	0010	0010	0000	0001	101	2201
102	0001	0010	0000	0010	102	1202
103	0001	0010	0000	0011	103	1203
104	0011	0010	0000	0100	104	3204
201	0000	0000	0000	0010	201	0002
202	0000	0000	0000	0011	202	0003
203	0000	0000	0000	0100	203	0004
204	0000	0000	0000	0000	204	0000
(c)			(d)			
101	LDA	201	Etykieta	Operacja	Otwarty	
102	ADD	202	FORMUŁ	LDA	I	
103	ADD	203		ADD	J	
104	STA	204		ADD	K	
				STA	N	
201	DAT	2				
202	DAT	3	I	DATA	2	
203	DAT	4	J	DATA	3	
204	DAT	0	K	DATA	4	
			N	DATA	0	

RYSUNEK 9.11. Obliczanie wyrażenia $N = I + J + K$: (a) program binarny; (b) program szesnastkowy; (c) program symboliczny; (d) assembler

Pamięć zarezerwowano dla czterech zmiennych począwszy od lokacji 201. Program składa się z czterech rozkazów:

1. Ładuj zawartość lokacji 201 do AC.
2. Dodaj zawartość lokacji 202 do AC.
3. Dodaj zawartość lokacji 203 do AC.
4. Zapisz zawartość AC w lokacji 204.

Jest to oczywiście proces nudny i podatny na błędy.

Niewielkim ulepszeniem jest napisanie programu w notacji szesnastkowej zamiast w binarnej (rys. 9.11b). Moglibyśmy napisać program jako ciąg wierszy. Każdy wiersz zawiera adres lokacji pamięci oraz kod szesnastkowy wartości binarnej, która ma być przechowana w tej lokacji. Potrzebujemy następnie programu, który zaakceptuje takie dane wejściowe, przetłumaczy każdy wiersz na liczbę binarną i zapisze ją w określonej lokacji.

Jest to tylko nieznaczne usprawnienie. Aby uzyskać znaczną poprawę, możemy używać nazw symbolicznych (mnemoników) każdego rozkazu. Użyjmy ich zamiast rzeczywistego kodu operacji. Wynikiem jest program **symboliczny** pokazany na rys. 9.11c. Każdy wiersz nadal reprezentuje jedną lokację w pamięci. Każdy wiersz składa się z trzech pól oddzielonych spacjami. Pierwsze pole

zawiera adres lokacji. W przypadku rozkazu drugie pole zawiera 3-literowy symbol kodu operacji. Jeśli jest to rozkaz odnoszący się do pamięci, to trzecie pole zawiera adres. W celu zapisania dowolnych danych w lokacji wynaleźliśmy **pseudorozkaz** o symbolu DAT. Jest to jedynie wskazanie, że trzecie pole w wierszu zawiera liczbę szesnastkową, która ma być zapisana w lokacji określonej przez pierwsze pole.

Dla takiego rodzaju danych wejściowych potrzebujemy nieco bardziej złożonego programu. Program akceptuje każdy wiersz danych wejściowych, generuje liczbę binarną opartą na drugim i trzecim (jeśli występuje) polu i zapisuje ją w lokacji określonej przez pierwsze pole.

Użycie programu symbolicznego ułatwia życie, jednak nadal jest kłopotliwe. Po pierwsze, musimy określać bezwzględne adresy każdego słowa. Oznacza to, że program i dane mogą być ładowane tylko do jednego miejsca w pamięci i że musimy znać to miejsce z wyprzedzeniem. Gorzej, jeśli pewnego dnia chcemy zmienić program, dodając lub usuwając jakiś wiersz. Zmieni to adresy wszystkich następnych słów.

O wiele lepszym, powszechnie obecnie używanym systemem jest użycie adresów symbolicznych. Zilustrowano to na rys. 9.11d. Każdy wiersz nadal składa się z trzech pól. Pierwsze pole nadal jest przeznaczone na adres, jednak zamiast bezwzględnego adresu numerycznego jest używany symbol. Niektóre wiersze nie mają adresu, co powoduje, że adres tego wiersza jest o 1 większy niż adres poprzedniego wiersza. W przypadku adresów odnoszących się do pamięci trzecie pole również zawiera adres symboliczny.

Poprzez to ostatnie ulepszenie wynaleźliśmy **język assemblerowy**. Programy napisane w języku assemblerowym (programy **assemblerowe**) są tłumaczone na język maszyny przez assembler. Program ten musi nie tylko tłumaczyć symbole przedyskutowane wcześniej, ale także przypisywać pewną postać adresów pamięci adresom symbolicznym.

Opracowanie języka assemblerowego było kamieniem milowym w ewolucji techniki komputerowej. Był to pierwszy krok do języków wysokiego poziomu używanych dzisiaj. Chociaż niewielu programistów używa języka assemblerowego, praktycznie wszystkie maszyny umożliwiają to. Możliwość taka jest potrzebna (jeżeli w ogóle) w przypadku programów systemowych, takich jak kompilatory i procedury wejścia-wyjścia.

POLECANA LITERATURA

Wiele książek zawiera wystarczające omówienie języka maszynowego i projektowania list rozkazów, w tym [HENN90], [TANE90] i [HAYE88]. Lista rozkazów Pentium jest omówiona w [BREY95] i [DEWA90]. Listę rozkazów PowerPC przedstawiono w [IBM94] i [WEIS94].

- BREY95 Brey B.: *The Intel 32-bit Microprocessors: 80386, 80486 and Pentium*. Englewood Cliffs, Prentice Hall, 1995.
- DEWA90 Dewar R., Smosna M.: *Microprocessors: A Programmer's View*. New York, McGraw-Hill, 1990.
- HAYE88 Hayes J.: *Computer Architecture and Organization*. Wyd. II. New York, McGraw-Hill, 1988.
- HENN90 Hennesy J., Patterson D.: *Computer Architecture: a Quantitative Approach*. San Mateo, Morgan Kaufmann, 1990.
- IBM94 International Business Machines, Inc.: *The PowerPC Architecture: A Specification for a New Family of RISC Processors*. San Francisco, Morgan-Kaufmann, 1994.
- TANE90 Tannenbaum A.: *Structured Computer Organization*. Englewood Cliffs, Prentice Hall, 1990.
- WEIS94 Weiss S., Smith J.: *POWER and PowerPC*. San Francisco, Morgan-Kaufmann, 1994.

1.6

PROBLEMY DO ROZWIĄZANIA

- 9.1. Wiele procesorów umożliwia prowadzenie działań arytmetycznych na upakowanych liczbach dziesiętnych. Chociaż reguły arytmetyki dziesiętnej są podobne do reguł operacji binarnych, wyniki dziesiętne mogą wymagać pewnych poprawek dotyczących pojedynczych cyfr, jeśli jest używana logika binarna.

Rozważmy dziesiętne dodawanie dwóch liczb bezznakowych. Jeśli każda liczba składa się z N cyfr, to w każdej liczbie jest $4N$ bitów. Obie liczby mają być dodane za pomocą sumatora binarnego. Zaproponuj prostą regułę korekty wyniku. Przeprowadź tym sposobem dodawanie liczb 1698 i 1786.

- 9.2. Uzupełnienie do 10 liczby dziesiętnej X jest definiowane jako $10^N - X$, gdzie N jest liczbą cyfr dziesiętnych w tej liczbie. Opisz użycie reprezentacji uzupełnienia do 10 w celu przeprowadzenia odejmowania dziesiętne-go. Zilustruj tę procedurę odejmując $(0326)_{10}$ od $(0736)_{10}$.
- 9.3. Powiedzieliśmy, że lista rozkazów określa wiele funkcji zrealizowanych przez procesor. Wymień pewne funkcje procesora niezależne od projektu listy rozkazów.
- 9.4. Porównaj maszyny O, 1, 2 i 3-adresowe, pisząc programy do obliczenia
- $$X = (A + B \times C) / (D - E \times F)$$

dla każdej z czterech maszyn. Dostępne są rozkazy:

0-adresowa	1-adresowa	2-adresowa	3-adresowa
PUSH M	LOAD M	MOV($X \leftarrow Y$)	MOV($X \leftarrow Y$)
POPM	STOREM	ADD($X \leftarrow X + Y$)	ADD($X \leftarrow Y + Z$)
ADD	ADD M	SUB($X \leftarrow X - Y$)	SUB($X \leftarrow Y - Z$)
SUB	SUB M	MUL($X \leftarrow X \times Y$)	MUL($X \leftarrow Y \times Z$)
MUL	MUL M	DIV($X \leftarrow X/Y$)	DIV($X \leftarrow Y/Z$)
DIV	DIV M		

- 9.5. Rozważmy hipotetyczny komputer z listą rozkazów obejmującą tylko dwa n -bitowe rozkazy. Pierwszy bit określa kod operacji, a pozostałe określają jedno z 2^{n-1} n -bitowych słów w pamięci głównej. Dwoma rozkazami są:

SUBS X Odejmij zawartość lokacji X od akumulatora i przechowaj wynik w lokacji X oraz w akumulatorze.

JUMP X Umieść adres X w liczniku programu.

Słowo w pamięci głównej może zawierać albo rozkaz, albo liczbę binarną w notacji uzupełnienia do dwóch. Zademonstruj, że ten repertuar rozkazów jest rozsądnie wystarczający, pokazując, jak mogą być zaprogramowane następujące operacje:

- transfer danych: lokacja X do akumulatora, akumulator do lokacji X;
 - Dodawanie: dodaj zawartość lokacji X do akumulatora;
 - logiczne LUB;
 - operacje wejścia-wyjścia.
- 9.6. Wiele list rozkazów zawiera rozkaz NOOP oznaczający brak operacji, który nie ma innego wpływu na procesor poza inkrementacją stanu licznika programu. Podaj pewne zastosowania tego rozkazu.
- 9.7. Załóżmy, że do zarządzania wywołaniami i powrotami podprogramów standardowych przez procesor ma być użyty stos. Czy można wyeliminować licznik programu przez zastosowanie wierzchołka stosu jako licznika programu?
- 9.8. W dodatku 9A jest powiedziane, że w liście rozkazów nie występują rozkazy zorientowane na używanie stosu, jeśli stos ma być wykorzystywany tylko przez procesor do takich celów, jak manipulowanie podprogramami. Jak procesor może używać stosu do jakiegokolwiek celu bez rozkazów zorientowanych na używanie stosu?
- 9.9. Przekształć następujące formuły z odwrotnej notacji polskiej na notację wrostkową (ang. *infix*):
- $AB + C + Dx$
 - $AB/CD/ +$
 - $ABCDE + xx /$
 - $ABCDE + F/ + G - H/x +$
- 9.10. Przekształć następujące formuły z notacji wrostkowej na odwrotną polską:
- $A + B + C + D$
 - $(A + B) \times (C + D) + E$
 - $(A \times B) + (C \times D) + E$
 - $(A - B) \times (((C - D \times E)/F)/G) \times H$
- 9.11. Przekształć wyrażenie $A + B - C$ na notację przyrostkową, stosując algorytm Dijkstry. Pokaż kolejne kroki. Czy wynik jest równoważny $(A + B) + - C$ lub $A + (B - C)$? Czy to ma znaczenie?

- 9.12. W architekturach IBM S/360 i S/370 każdy bajt pamięci głównej ma jednoznaczny adres. Oryginalna architektura IBM S/360 wymaga, aby argumenty w pamięci miały *wyrównane adresy*. Oznacza to, że argument 2-bajtowy musi zaczynać się pod adresem będącym wielokrotnością 2, argument 4-bajtowy musi *rozpocząć* się pod adresem będącym wielokrotnością 4, a argument 8-bajtowy musi rozpoczynać się pod adresem będącym wielokrotnością 8. Począwszy od IBM 360/85 oraz w serii IBM 370 wymaganie wyrównania adresów zostało usunięte. W późniejszym czasie w IBM 370 zostało wprowadzone stronicowanie z rozmiarami stron 2048 lub 4096 bajtów (z adresami wyrównanymi na 2 K i 4 K). Zastanów się, w jakim stopniu niewyrównane argumenty oraz kontrola błędu strony oddziałują wzajemnie, spowalniając *wszystkie* odniesienia do pamięci w tych późniejszych maszynach.
- 9.13. Architektura Pentium obejmuje rozkaz nazywany DAA - wyrównanie dziesiętne po dodawaniu (skrót od *decimal adjust after addition*). DAA obejmuje następującą sekwencję rozkazów:

```
if((ALAND 0FH) > 9) OR (AF = 1) then
```

```
    AL ← AL + 6;
```

```
    AF ← 1;
```

```
else
```

```
    AF ← 0;
```

```
endif:
```

```
if((AL > 9FH) OR (CF = 1) then
```

```
    AL ← AL + 60H;
```

```
    CF ← 1;
```

```
else
```

```
    CF ← 0;
```

```
endif:
```

„H” oznacza notację szesnastkową. AL jest rejestrem 8-bitowym, który zachowuje wynik dodawania dwóch beznakowych 8-bitowych liczb całkowitych. AF jest znacznikiem stanu ustawianym, jeśli w wyniku dodawania wystąpi przeniesienie z bitu 3 do bitu 4. SF jest znacznikiem stanu ustawianym, gdy nastąpi przeniesienie z bitu 7 do 8. Wyjaśnij funkcję realizowaną przez rozkaz DAA.

- 9.14. Występujący w Pentium rozkaz porównania CMP (ang. *Compare*) powoduje: wykonanie odejmowania argumentu źródłowego od argumentu docelowego, aktualizację znacznika stanu (C, P, A, Z, S, O), jednak nie powoduje zmiany żadnego z argumentów. Po rozkazie CMP może wystąpić rozkaz skoku warunkowego Jcc (ang. *Conditional Jump*) lub zerowania warunkowego SETcc (ang. *Set Condition*), gdzie cc odnosi się do jednego z 16 warunków wymienionych w tab. 9.9. Zademonstruj, że warunki testowane przy porównywaniu liczby mającej znak są prawidłowe.

9.14. Większość list rozkazów mikroprocesorów zawiera rozkaz, który powoduje sprawdzenie warunku i ustalenie argumentu docelowego, jeśli warunek jest spełniony. Przykładami są SETcc w Pentium, Scc w MC6800 Motoroli oraz Scond w NS32000 firmy National.

- (a) Istnieją pewne różnice między tymi rozkazami:
- SETcc i Scc operują tylko na bajtach, podczas gdy Scond operuje na bajtach, słowach i podwójnych słowach.
 - a SETcc i Scond ustalają argument w postaci liczby całkowitej 1, jeśli warunek jest spełniony, oraz na 0, jeśli nie jest. Scc ustala bajt w postaci wszystkich binarnych jedynek, jeśli warunek jest spełniony, oraz wszystkich zer, gdy nie jest. Jakie są względne zalety i wady wynikające z tych różnic?
- (b) Żaden z tych rozkazów nie ustawia znaczników stanu kodu warunkowego i wobec tego jest wymagane odrębne sprawdzenie wyniku rozkazu w celu określenia jego wartości. Przedyskutuj, czy kody warunkowe powinny być ustawiane jako wynik tego rozkazu.
- (b) Prosta instrukcja IF, taka jak IF $a > b$ THEN może być zaimplementowana za pomocą metody reprezentacji numerycznej, to znaczy za pomocą ujawniania wartości Boole'a, w przeciwieństwie do metody przepływu sterowania (ang. *flow of control*), w której wartość wyrażenia Boole'a jest reprezentowana przez punkt osiągnięty w programie. Kompilator może wdrożyć IF $a > b$ THEN za pomocą następującego kodu procesora 80x86:

	SUB	CX, CX	;ustaw rejestr CX na 0
	MOV	AX, B	;przesuń zawartość lokacji B do rejestru AX
	CMP	AX, A	;porównaj zawartość rejestru AX i lokacji A
	JLE	TEST	;skocz, jeśli $A \leq B$
	INC	CX	;dodaj 1 do zawartości rejestru CX
TEST	JCXZ	OUT	;skocz, jeśli zawartość CX równa 0
THEN			
OUT			

Wynik ($A > B$) jest wartością Boole'a zachowaną w rejestrze i dostępną później, poza kontekstem pokazanego wyżej strumienia kodów. Wygodne jest używanie do tego celu rejestru CX, ponieważ wiele kodów operacyjnych skoku i pętli ma wbudowane sprawdzanie rejestru CX.

Pokaż alternatywną implementację, używając rozkazu SETcc, które oszczędzałyoby pamięć i czas wykonywania. (Wskazówka: poza SETcc nie są potrzebne dodatkowe rozkazy procesora 80x86).

- (d) Rozważmy teraz instrukcję w języku wysokiego poziomu:

$A := (B > C) \text{ OR } (D = F)$

Kompilator może generować następujący kod:

```

MOV    EAX, B    ;przenieś z lokacji B do rejestru EAX
CMP    EAX, C
MOV    BL, 0      ;0 reprezentuje fałsz (niespełnienie warunku)
JLE    N1
MOV    BL, 1      ;1 reprezentuje prawdę (spełnienie warunku)
N1:    MOV    EAX, D
        CMP    EAX, F
        MOV    BH, 0
        JNE    N2
        MOV    BH, 1
N2:    OR     BL, BH

```

Pokaż alternatywną implementację z zastosowaniem rozkazu SETcc, które oszczędza pamięć i czas wykonywania.

- 9.16. Przekształć strukturę typu *najpierw najmłodszybajt* („cienkokończową”) na rys. 9.18 tak, żeby bajty wyglądały na liczone według schematu *najpierw najstarszy bajt* („grubokończową”). To znaczy, pokaż pamięć w postaci wierszy 64-bitowych, z bajtami liczonymi od lewej do prawej oraz od góry do dołu.
- 9.17. Dla następujących struktur danych wykreśl struktury typu *najpierw najstarszy bajt* („grubokońcowe”) i *najpierw najmłodszy bajt* („cienkokońcowe”), stosując format z rys. 9.18. Skomentuj wyniki.

```

(a) struct {
    double i;    //0x1112131415161718
} s1;
(b) struct {
    int i;       //0x11121314
    int j;       //0x15161718
} s2;
(c) struct {
    short i;     //0x1112
    short j;     //0x1314
    short k;     //0x1516
    short l;     //0x1718
} s3;

```

- 9.18. Specyfikacja architektury PowerPC nie dyktuje, jak powinien być wdrażany tryb *najpierw najmłodszybajt*. Określa ona tylko widok pamięci, jaki musi postrzegać procesor, pracując w tym trybie. Podczas konwersji struktury danych z *najpierw najstarszy bajt* na *najpierw najmłodszy bajt* istnieje wolność wyboru między implementacją prawdziwego mechanizmu wymiany bajtów lub użyciem pewnego rodzaju mechanizmu modyfikacji

		Odwzorowanie adresu typu <i>najpierw najmłodszy bajt</i>							
Adres bajta						11	12	13	14
		00	01	02	03	04	05	06	07
00		21	22	23	24	25	26	27	28
08		08	09	0A	0B	0C	0D	0E	0F
10		'D'	'C'	'B'	'A'	31	32	33	34
		10	11	12	13	14	15	16	17
18				51	52		'G'	'F'	'E'
		18	19	1A	1B	1C	1D	1E	1F
20						61	62	63	64
		20	21	22	23	24	25	26	27

RYSUNEK 9.12. Cienkokońcowe struktury w pamięci PowerPC

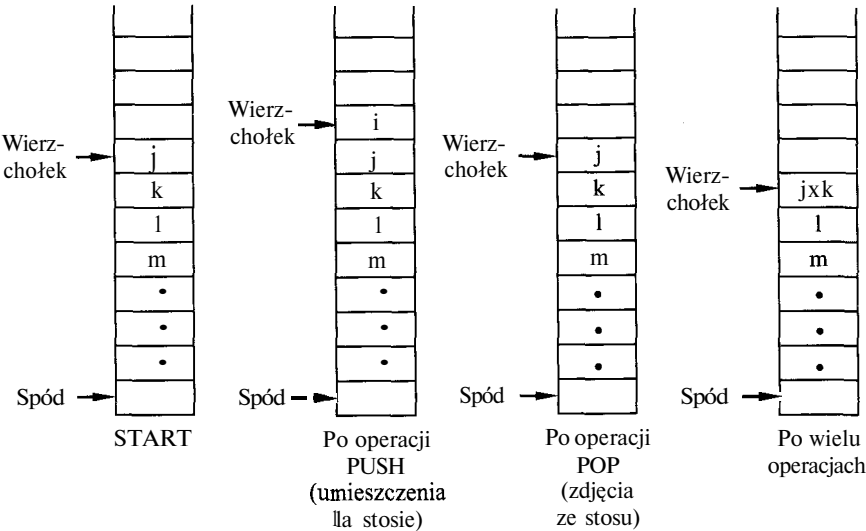
adresu. Obecnie procesory PowerPC wszystkie są maszynami pracującymi w trybie *najpierwnajstarszybajt*, stosującymi metodę modyfikacji adresu, aby dane były traktowane jako dane typu *najpierw najmłodszy bajt*. Rozważmy strukturę *s* zdefiniowaną na rys. 9.18. Prawa dolna część rysunku pokazuje strukturę *s* widzianą przez procesor. W rzeczywistości, jeśli struktura *s* jest utworzona w trybie *najpierw najmłodszy bajt*, jej rozkład w pamięci wygląda tak, jak na rys. 9.12. Wyjaśnij wykorzystane odwzorowanie, opisz prosty sposób wdrożenia odwzorowania i przedyskutuj efektywność tego rozwiązania.

DODATEK 9A

Stosy

Stos jest uporządkowanym zestawem elementów, z których tylko do jednego można mieć dostęp w określonej chwili. Punkt dostępu nazywa się **wierzchołkiem** stosu. Liczba elementów w stosie, czyli **długość** stosu, jest zmienna. Obiekty mogą być dodawane lub odejmowane tylko z wierzchołka stosu. Z tego powodu stos jest również znany jako lista „spychania” (ang. *pushdown list*) lub **lista typu ostatni na wejściu - pierwszy na wyjściu** (ang. *last-in-first-out list* - LIFO).

Najprostszym sposobem wyjaśnienia struktury i działania stosu jest ilustracja. Na rysunku 9.13 są pokazane podstawowe operacje, które mogą być realizowane na stosie. Rozpocniemy w chwili, w której stos zawiera pewną liczbę



RYSUNEK 9.13. Podstawowe operacje dotyczące stosu

elementów. Operacja PUSH to dodanie nowego obiektu na wierzchołek stosu; operacja POP - usunięcie wierzchołkowego elementu ze stosu. W obu przypadkach wierzchołek stosu ulega odpowiedniemu przesunięciu. Ponadto operacje binarne wymagające dwóch argumentów (np. mnożenie, dzielenie, dodawanie, odejmowanie) używają dwóch wierzchołkowych elementów stosu jako argumentów, po czym wprowadzają wynik na stos. Operacje jednoargumentowe (np. logiczne NIE) używają wierzchołkowego elementu stosu. Wszystkie te operacje są zebrane w tabeli 9.11.

TABELA 9.11. Operacje dotyczące stosu

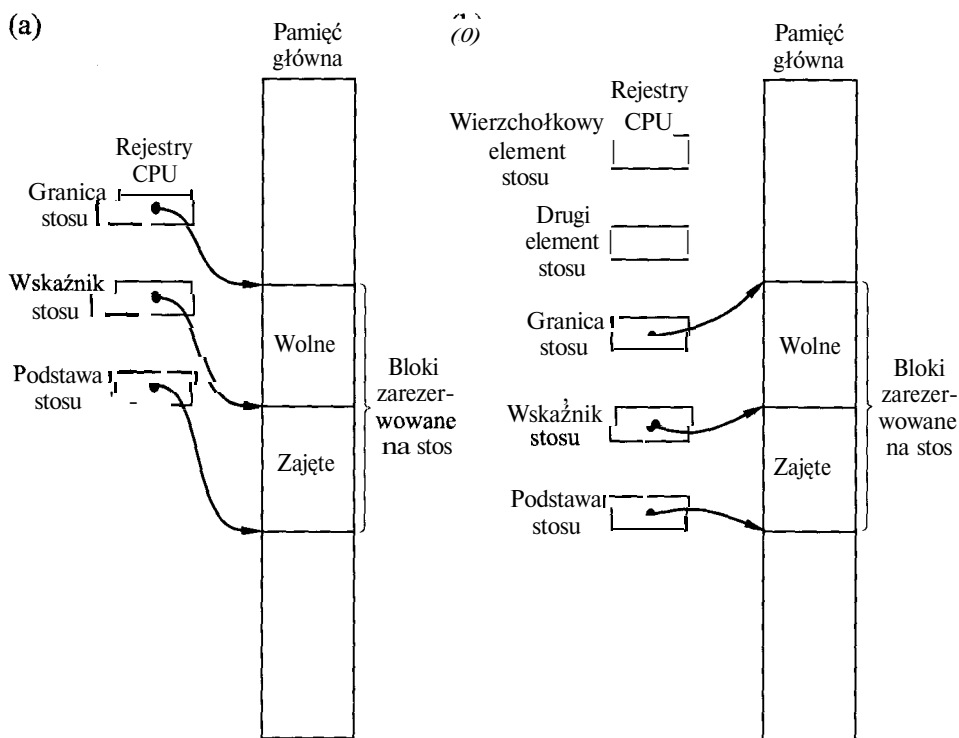
PUSH	Umieszczenie nowego elementu na wierzchołku stosu.
POP	Usunięcie wierzchołkowego elementu stosu.
Operacja jednoargumentowa	Wykonanie operacji na wierzchołkowym elemencie stosu. Zamiana elementu wierzchołkowego na wynik.
Operacja dwuargumentowa	Wykonanie operacji na dwóch wierzchołkowych elementach stosu. Usunięcie dwóch wierzchołkowych elementów stosu i umieszczenie wyniku operacji na wierzchołku stosu.

9A.1. Implementacja stosu

Stos jest na tyle użyteczną strukturą, że stanowi część implementacji procesora. Jednym z zastosowań, przedyskutowanym w podrozdz. 9.3, jest zarządzanie wywołaniami i powrotami podprogramów. Stosy mogą też być użyteczne dla programistów. Przykładem tego jest obliczanie wyrażeń, omówione w dalszej części tego punktu.

Implementacja stosu zależy częściowo od jego potencjalnych zastosowań. Jeśli jest pożądane, aby operacje na stosie były dostępne dla programisty, to lista rozkazów będzie zawierała specjalne operacje dotyczące stosu, włącznie z PUSH, POP i operacjami, które używają jednego lub dwóch wierzchołkowych elementów jako argumentów. Ponieważ wszystkie te operacje odnoszą się do jednoznacznej lokacji, mianowicie do wierzchołka stosu, adres argumentu lub argumentów jest domyślny i nie musi być włączany do rozkazu. Są to więc rozkazy 0-adresowe omówione w p. 9.1.

Jeśli mechanizm stosu ma być używany tylko przez procesor, do takiego celu jak manipulowanie podprogramami, w liście rozkazów nie będą występowały rozkazy ewidentnie dotyczące stosu. W każdym przypadku implementacja stosu wymaga, żeby istniała grupa lokacji używana do przechowywania elementów stosu. Typowe rozwiązanie zilustrowano na rys. 9.14a. W pamięci głównej lub w wirtualnej jest zarezerwowany blok sąsiadujących lokacji przeznaczony na stos. Przez większość czasu blok jest częściowo wypełniony elementami stosu, pozostałość zaś jest przewidziana na rozrastanie się stosu. Do poprawnego działania są wymagane trzy adresy; często są one przechowywane w rejestrach procesora:



RYSUNEK 9.14. Typowe organizacje stosu: (a) całość stosu w pamięci; (b) dwa elementy wierzchołkowe w rejestrach

- a **Wskaźnik stosu.** Zawiera adres wierzchołka stosu. Jeśli element jest dodawany lub usuwany ze stosu, wskaźnik jest odpowiednio inkrementowany lub dekrementowany, aby w dalszym ciągu pokazywał wierzchołek stosu.
- a **Podstawa stosu.** Zawiera adres najniższej lokacji w zarezerwowanym bloku. Jeśli dokonywana jest próba operacji POP, gdy stos jest pusty, zgłaszany jest błąd.
- a **Granica stosu.** Zawiera adres drugiego końca zarezerwowanego bloku. Jeśli czyniona jest próba operacji PUSH, gdy stos jest zapełniony, zgłaszany jest błąd.

W celu przyspieszenia operacji na stosie dwa wierzchołkowe elementy stosu są często przechowywane w rejestrach, co widać na rys. 9.14b. W tym przypadku, wskaźnik stosu zawiera adres trzeciego elementu stosu.

9A.2. Obliczanie wyrażeń

Wzory matematyczne są zwykle wyrażane za pomocą notacji nazywanej **wrostkową** (*infix*). W tej postaci operator binarny występuje między argumentami (np. $a + b$). W przypadku złożonych wyrażeń do określania porządku obliczeń używa się nawiasów. Na przykład $a + (b \times c)$ da inny wynik niż $(a + b) \times c$. Aby zminimalizować liczbę użytych nawiasów, operacje mają domyślne pierwszeństwo. Na ogół mnożenie ma pierwszeństwo przed dodawaniem, więc $a + b \times c$ jest równoważne $a + (b \times c)$.

Alternatywną metodą jest **odwrotna notacja polska** zwana **notacją przyrostkową**. W tym przypadku operator następuje po swoich argumentach. Na przykład

$a + b$	staje się	$ab +$
$a + (b \times c)$	staje się	$abc \times +$
$(a + b) \times c$	staje się	$ab + c \times$

Zauważmy, że pomijając złożoność wyrażenia, w notacji odwrotnej polskiej nie są wymagane żadne nawiasy.

Zaletą notacji przyrostkowej jest to, że wyrażenie w tej formie łatwo jest obliczać za pomocą stosu. Jest ono skanowane od lewej do prawej. W odniesieniu do każdego elementu wyrażenia są stosowane następujące reguły:

1. Jeśli element jest zmienną lub stałą, należy go umieścić na stosie.
2. Jeżeli element jest operatorem, należy pobrać dwa elementy ze stosu, wykonać operację i umieścić wynik na stosie.

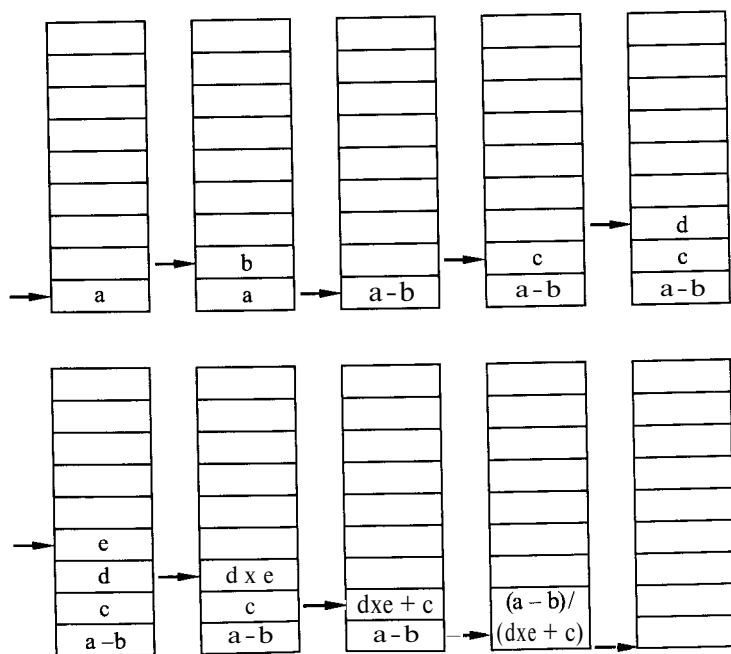
Po przejrzaniu całego wyrażenia wynik znajduje się na wierzchołku stosu.

Prostota tego algorytmu powoduje, że jest on wygodny przy obliczaniu wyrażeń. Dlatego też wiele kompilatorów dokonuje konwersji wyrażeń w języku wysokiego poziomu na notację przyrostkową, a następnie generuje rozkazy maszynowe odnoszące się do tej notacji. Na rysunku 9.15 widać sekwencję rozka-

zów maszynowych służącą do obliczenia $f = (a - b) / (c + d \times e)$ za pomocą rozkazów wykorzystujących stos. Jest także pokazane użycie rozkazów 1- i 2-adresowych. Reguły stosu nie były używane w ostatnich dwóch przypadkach; zauważmy że mimo to notacja przyrostkowa służyła jako przewodnik do generowania rozkazów maszynowych. Sekwencję zdarzeń dla programu wykorzystującego stos widać na rys. 9.16.

	Stos	Rejestry robocze	Pojedynczy rejestr
	Umieść a	Ładuj G[1], a	Ładuj d
	Umieść b	Odejmij G[1], b	Pomnóż e
	Odejmij	Ładuj G[2], d	Dodaj c
	Umieść c	Pomnóż G[2], e	Zapisz f
	Umieść d	Dodaj G[2], c	Ładuj a
	Umieść e	Podziel G[1], G[2]	Odejmij b
	Pomnóż	Zapisz G[1], f	Podzielf
	Dodaj		Zapisz f
	Podziel		
	Zdejmij f		
Liczba rozkazów	10	7	8
Dostęp do pamięci	10 op + 6d	7 op + 6d	8 op + 8d

RYSUNEK 9.15. Porównanie trzech programów do obliczania $f = (a - b) / (c + d \times e)$



RYSUNEK 9.16. Użycie stosu do obliczenia $f = (a - b) / (d \times e + c)$

Proces konwersji wyrażenia wrostkowego na przyrostkowe może być najłatwiej zrealizowany również za pomocą stosu. Następujący algorytm zawdzięczamy Dijkstrze [DIJK63]. Wyrażenie wrostkowe jest przeglądane od lewej do prawej, a wyrażenie przyrostkowe jest tworzone i wyprowadzane podczas tego przeglądania. Etapy tego procesu są następujące:

1. Zbadaj następny element na wejściu.
2. Jeśli jest on argumentem, skieruj go do wyjścia.
3. Jeśli jest on nawiasem otwierającym, umieść go na stosie.
4. Jeśli jest on operatorem, to
 - jeśli na wierzchołku stosu znajduje się nawias otwierający, to umieść operator na stosie;
 - jeśli ma on wyższy priorytet niż wierzchołek stosu (mnożenie i dzielenie mają wyższy priorytet niż dodawanie i odejmowanie), to umieść operator na stosie;
 - w innym przypadku skieruj operację ze stosu do wyjścia i powtórz krok 4.
5. Jeśli jest on nawiasem zamykającym, to pobieraj operatory do wyjścia, aż napotkasz nawias otwierający. Pobierz i wyrzuć nawias otwierający.
6. Jeśli istnieją jeszcze dane na wejściu, przejdź do kroku 1.
7. Jeśli nie ma danych na wejściu, pobierz ze stosu pozostałe argumenty.

Rysunek 9.17 ilustruje użycie tego algorytmu. Przykład ten powinien dać czytelnikowi pewne odczucie potęgi algorytmów opartych na stosach.

Wejście	Wyjście	Stos (wierzchołek po prawej)
$A + B * C + (D + E) * F$	pusty	pusty
$+ B * C + (D + E) * F$	A	pusty
$B * C + (D + E) * F$	A	+
$* C + (D + E) * F$	AB	+
$C + (D + E) * F$	AB	+*
$+ (D + E) * F$	ABC	+*
$(D + E) * F$	ABC*+	+
$D + E) * F$	ABC*+	+(
$+ E) * F$	ABC*+D	+(
$E) * F$	ABC*+D	+(+
$) * F$	ABC*+DE	+(+)
$* F$	ABC*+DE+	+
F	ABC*+DE+	+*
pusty	ABC*+DE+F	+*
pusty	ABC*+DE+F*+	

RYSUNEK 9.17. Konwersja wyrażenia z notacji wrostkowej na przyrostkową

DODATEK 9B

Struktury „cienko-”, „grubo-” i „dwukońcowe”

Zjawisko (przykre i osobliwe), o którym będziemy dyskutować, wiąże się ze sposobem reprezentowania i odnoszenia się do bajtów wewnątrz słowa oraz do bitów wewnątrz bajta.

9B.1. Porządkowanie bajtów

Koncepcja „końcowości” była po raz pierwszy przedyskutowana w literaturze przez Cohena [COHE81]. W odniesieniu do bajtów dotyczy to porządkowania bajtów w wielobajtowych wartościach skalarnych. Najlepiej jest przedstawić to zagadnienie za pomocą przykładu. Załóżmy, że mamy 32-bitową wartość w notacji szesnastkowej 12345678 i że jest ona przechowywana w 32-bitowym słowie w pamięci umożliwiającej adresowanie bajtowe, w lokacji bajtowej 184. Wartość składa się z 4 bajtów, przy czym najmniej znaczący bajt zawiera wartość 78, zaś najbardziej znaczący bajt zawiera wartość 12. Istnieją dwa sposoby przechowywania tej wartości:

Adres	Wartość	Adres	Wartość
184	12	184	78
185	34	185	56
186	56	186	34
187	78	187	12

Odwzorowanie przedstawione po lewej stronie polega na przechowywaniu najbardziej znaczącego bajta pod najniższym numerycznym adresem bajtowym; jest to równoważne kierunkowi pisania od lewej do prawej przyjętemu w kulturze zachodniej. Takie odwzorowanie jest określane jako „grubokońcowe”.

Odwzorowanie po prawej przewiduje umieszczenie najmniej znaczącego bajta pod najniższym numerycznym adresem bajtowym; jest ono znane jako „cienkokońcowe” i jest reminiscencją porządku od prawej do lewej operacji arytmetycznych w jednostkach arytmetycznych². „Grubokońcowa” struktura wielobajtowej wielkości skalarnej jest odwróconym odwzorowaniem bajtowym struktury „cienkokońcowej”.

Problem końcowości powstaje, gdy jest konieczne traktowanie wielobajtowej wielkości jako jednostki danych o jednym adresie, jeśli nawet składa się ona z mniejszych, adresowalnych elementów. Niektóre maszyny, takie jak Intel

² Wyrażenia „grubokońcowy” i „cienkokońcowy” pochodzą z rozdziału 4 części I „Podróży Guliwera do wielu odległych narodów świata” Jonathana Swifta. Odnoszą się do wojny religijnej między dwiema grupami (Grubokońców i Cienkokońców), jedną tłukącą jąka od grubszego końca i drugą tłukącą je od cieńszego końca. Wydanie polskie w przekładzie Macieja Słomczyńskiego, Wydawnictwo Literackie, Kraków 1979), (*przypłum.*).

80x86, Pentium i VAX są maszynami „cienkokońcowymi”, podczas gdy inne, takie jak IBM System 370, Motorola 680x0 i większość maszyn RISC są „grubokońcowe”. Wynikają stąd problemy, gdy dane są przenoszone z maszyny jednego rodzaju do maszyny drugiego rodzaju i gdy programista chce manipulować indywidualnymi bajtami lub bitami wewnątrz wielobajtowej wielkości skalarnej.

Własność końcowości nie rozciąga się poza indywidualną jednostkę danych. W dowolnej maszynie zbiory, takie jak pliki, struktury danych i matryce są złożone z wielu jednostek danych, z których każda ma określoną „kończowość”. Wobec tego konwersja bloku pamięci z jednego stylu „kończowości” do drugiego wymaga znajomości struktury danych.

Na rysunku 9.18 jest pokazane, jak końcowość determinuje adresowanie i porządek bajtów. Struktura C w górnej części rysunku zawiera pewną liczbę rodzajów danych. Rozplanowanie pamięci pokazane w lewej dolnej części rysunku wynika z ułożenia tej struktury pod kątem maszyny „grubokońcowej”, a pokazane w prawej dolnej części - pod kątem maszyny „cienkokońcowej”. W każdym przypadku pamięć jest przedstawiona w postaci ciągu wierszy 64-bitowych. W przypadku „grubokońcowym” pamięć jest układana od lewej do prawej i od góry do dołu, podczas gdy w przypadku „cienkokońcowym” - od prawej do lewej i od dołu do góry. Zauważmy, że te rozplanowania są arbitralne. Każdy z tych schematów mógłby używać wewnątrz wiersza kierunku od lewej do prawej lub od prawej do lewej; jest to problem zobrazowania, a nie przyporządkowania pamięci. W rzeczywistości, przeglądając podręczniki programowania dla różnych maszyn, obserwujemy zadziwiającą kolekcję zobrazowań, nawet wewnątrz tego samego podręcznika.

Możemy poczynić kilka spostrzeżeń w odniesieniu do tej struktury danych:

- ❑ Każdy element danych ma taki sam adres w obu schematach. Na przykład adresem podwójnego słowa o wartości szesnastkowej 2122232425262728 jest 08.
- ❑ Wewnątrz dowolnej wielobajtowej wartości skalarnej uporządkowanie bajtów w strukturze „cienkokońcowej” jest odwrotne do uporządkowania w strukturze „grubokońcowej”.
- a Kończowość nie wpływa na uporządkowanie elementów danych wewnątrz struktury. Wobec tego 4-znakowe słowo c ma odwrócony porządek bajtów, a 7-znakowy zespół bajtów d nie. W związku z tym adres każdego indywidualnego elementu d jest taki sam w obu strukturach.

Wpływ końcowości można jaśniej zademonstrować, spoglądając na pamięć jako na pionowy zespół bajtów, co widać na rys. 9.19.

Nie istnieje zgodny pogląd co do tego, który styl końcowości jest lepszy.³ Oto co przemawia za stylem „grubokońcowym”:

³ Prorok szanowany przez obie grupy w wojnach Końcowych w „Podróżach Guliwera do wielu odległych narodów świata” miał do powiedzenia, co następuje: „Niechaj wszyscy wierni tłukają się z bardziej dogodnego końca”. Niezbyt pomocne!

Struct{			
int	a;	//0'1112_1314	Słowo
int	pad;		
double	b;	//0'2122_2324-2526_2728	Podwójne słowo
char*	c;	//0'3132_3334	Słowo
char	d [7];	// 'A', 'B', 'C', 'D', 'E', 'F', 'G'	Tablica bajtów
short	e;	//0'5152	Półsłowo
int	f;	//0'6162_6364	Słowo

		Odwzorowanie adresu typu najpierw najstarszy bajt								Odwzorowanie adresu typu najpierw najmłodszy bajt									
Adres	bajta	11	12	13	14									11	12	13	14	Adres	bajta
	00	00	01	02	03	04	05	06	07					07	06	05	04	00	
		21	22	23	24	25	26	27	28					21	22	23	24		
08		08	09	0A	0B	0C	0D	0E	0F					0F	0E	0D	0C	08	
		31	32	33	34	'A'	'B'	'C'	'D'					'D'	'C'	'B'	'A'		
10		10	11	12	13	14	15	16	17					17	16	15	14	10	
		'E'	'F'	'G'		51	52								51	52			
18		18	19	1A	1B	1C	1D	1E	1F					1F	1E	1D	1C	18	
		61	62	63	64														
20		20	21	22	23														
														61	62	63	64		
														23	22	21	20	20	

RYSUNEK 9.18. Przykład struktury danych C oraz jej odwzorowania „grubo-” i „cienkokońcowe” [IBM94]

- Q *Sortowanie łańcuchów znaków.* Procesor „grubokońcowy” szybciej porównuje wyrównane łańcuchy znaków; ALU może równolegle porównywać wiele bajtów.
- Q *Zrzuty dziesiętne/ASCII.* Wszystkie wartości mogą być drukowane od lewej do prawej bez wywoływania nieporozumień.
- *Spójny porządek.* Procesory „grubokońcowe” przechowują łańcuchy liczb całkowitych i znaków w tym samym porządku (jako pierwszy bajt najbardziej znaczący).

Natomiast następujące punkty przemawiają za stylem „cienkokońcowym”:

- *Konwersja adresów liczb całkowitych.* Procesor „grubokońcowy” musi wykonać dodawanie, gdy przekształca adres 32-bitowej liczby całkowitej na adres 16-bitowej liczby całkowitej w celu użycia najmniej znaczących bajtów.
- U *Arytmetyka.* W stylu „cienkokońcowym” łatwiej jest wykonywać działania arytmetyczne z dużą dokładnością; nie musi się szukać najmniej znaczącego bajta i wracać.

Różnice są niewielkie i wybór stylu końcowości jest często raczej sprawą przystosowania poprzednich maszyn niż czegokolwiek innego.

(a)		(b)	
00	11	00	14
	12		13
	13		12
	14		11
04		04	
08	21	08	28
	22		27
	23		26
	24		25
0C	25	0C	24
	26		23
	27		22
	28		21
10	31	10	34
	32		33
	33		32
	34		31
14	'A'	14	'A'
	'B'		'B'
	'C'		'C'
	'D'		'D'
18	'E'	18	'E'
	'F'		'F'
	'G'		'G'
1C	51	1C	52
	52		51
20	61	20	64
	62		63
	63		62
	64		61

RYSUNEK 9.19. Inne ujęcie rys. 9.18: (a) „grubokońcowe”; (b) „cienkokońcowe”

PowerPC jest procesorem „dwukońcowym” (*bi-endian*), gdyż może pracować zarówno w trybie „grubokońcowym”, jak i „cienkokońcowym”. Architektura „dwukońcowa” umożliwia programistom wybieranie dowolnego trybu, gdy przenoszony jest system operacyjny i programy użytkowe z innych maszyn. System operacyjny ustala tryb końcowości, w których realizuje procesy. Gdy już tryb został wybrany, wszystkie następne załadowania pamięci są wyznaczone przez model adresowania wynikający z tego trybu. Aby umożliwić obsługę tej własności sprzętowej, pozostawione zostały dwa bity w rejestrze stanu maszyny (MSR). Jeden bit służy do określenia trybu końcowości, w jakim pracuje jądro; drugi - do określenia bieżącego trybu pracy procesora. Wobec tego tryb może być zmieniany od procesu do procesu.

9.B.2. Porządkowanie bitów

Porządkując bity wewnątrz bajta, natychmiast stykamy się z dwiema kwestiami:

1. Czy liczymy pierwszy bit jako bit 0, czy jako bit 1?
2. Czy przypisujemy najniższy numer bitowy najmniej znaczącemu bitowi bajta (styl „cienkokońcowy”), czy też najbardziej znaczącemu bitowi bajta (styl „grubokońcowy”)?

Odpowiedzi na te pytania nie są jednakowe w przypadku wszystkich maszyn. W rzeczywistości w przypadku niektórych maszyn odpowiedzi są różne w różnych okolicznościach. Ponadto wybór „grubokońcowego” lub „cienkokońcowego” porządku bitów w bajcie nie zawsze jest spójny z uporządkowaniem bajtów wewnątrz wielobajtowego składowanego. Programista musi interesować się tymi zagadnieniami, manipulując indywidualnymi bitami.

Innym obszarem zainteresowań jest transmitowanie danych bitową linią szeregową. Gdy jest transmitowany pojedynczy bajt, czy system transmituje najpierw bit najbardziej znaczący, czy najmniej znaczący? Projektant musi się upewnić, że nadchodzące bity są właściwie interpretowane i traktowane. Omówienie tego zagadnienia znajduje się w [JAME90].

Listy rozkazów: tryby adresowania i formaty

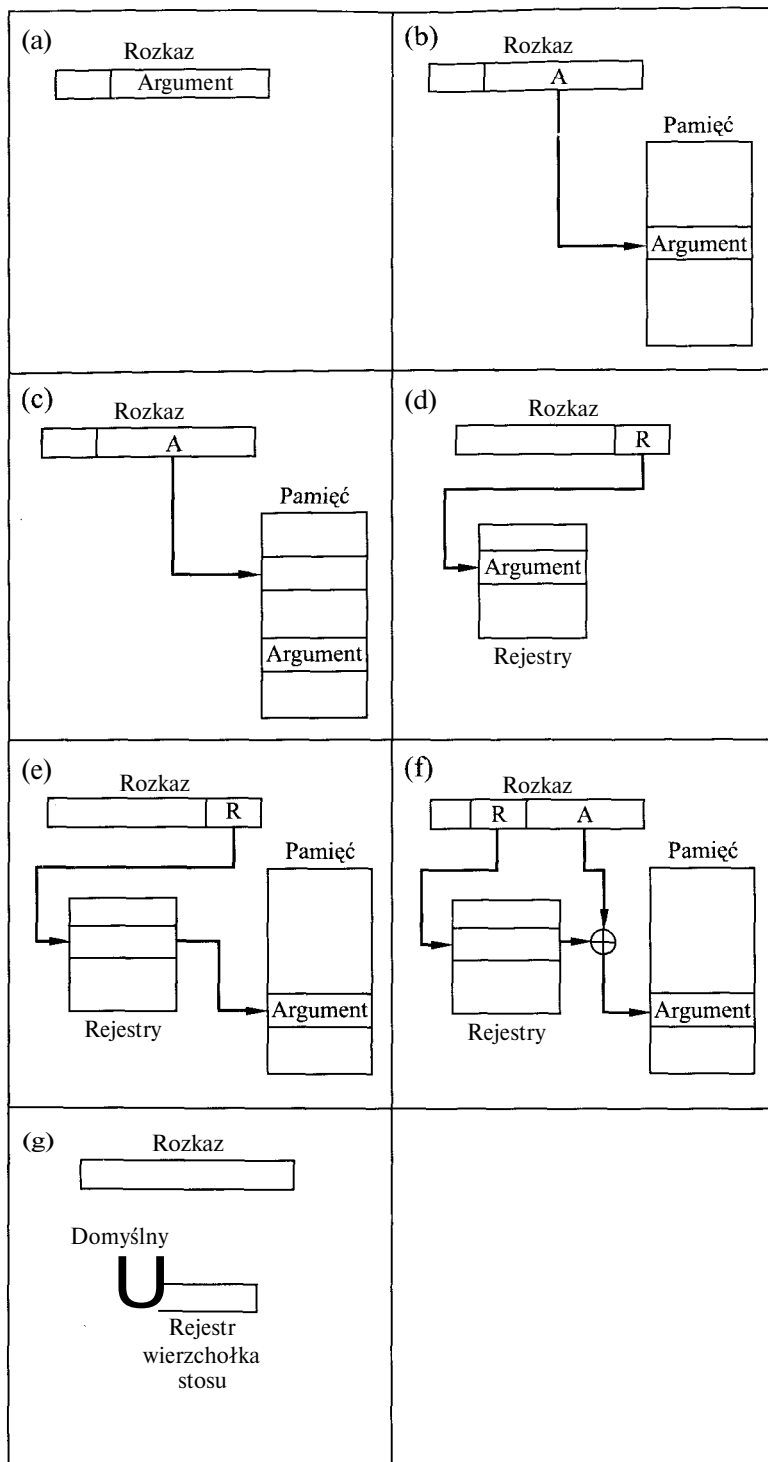
W rozdziale 9 skupiliśmy się na tym, jaką rolę pełni lista rozkazów. W szczególności przeanalizowaliśmy rodzaje argumentów i operacji, które mogą być określane za pomocą rozkazów maszynowych. W tym rozdziale zajmiemy się zagadnieniem, jak określać argumenty i operacje w rozkazie. Występują tu dwa problemy. Po pierwsze, jak jest określany adres argumentu, i po drugie, jak są organizowane bity rozkazu w celu określenia adresów argumentów i operacji.

10.1

ADRESOWANIE

Jak już stwierdziliśmy, pole lub pola adresowe w typowym formacie rozkazu są ograniczone. Chcielibyśmy dysponować możliwością odnoszenia się do dużego zakresu lokacji w pamięci głównej lub - w pewnych systemach - wirtualnej. Aby osiągnąć ten cel, zastosowano różnorodne metody adresowania. Wszystkie te metody opierają się na kompromisie między zakresem adresów i (lub) elastycznością adresowania z jednej strony, a liczbą odniesień do pamięci i (lub) złożonością obliczania adresów z drugiej strony. W tym podrozdziale przeanalizujemy najpowszechniejsze tryby adresowania:

- natychmiastowy,
- Q bezpośredni,
- Q pośredni,
- rejestrowy,
 - a rejestrowy pośredni ,
 - a z przesunięciem,
 - a stosowy.



RYSUNEK 10.1. Tryby adresowania: (a) natychmiastowy; (b) bezpośredni; (c) pośredni; (d) rejestrowy; (e) rejestrowy pośredni; (f) z przesunięciem; (g) stosowy

Tryby te są zilustrowane na rys. 10.1. W tym podrozdziale będziemy używać następujących oznaczeń:

A - zawartość pola adresowego w rozkazie,

EA - rzeczywisty (efektywny) adres lokacji zawierającej odniesiony argument,

(X) - zawartość lokacji X.

W tabeli 10.1 są podane obliczenia adresów dla każdego trybu adresowania.

Przed rozpoczęciem dyskusji konieczne są dwie uwagi. Po pierwsze, praktycznie wszystkie architektury komputerowe przewidują więcej niż jeden spośród wymienionych trybów adresowania. Powstaje kwestia, jak jednostka sterująca może stwierdzić, który tryb adresowania został użyty w poszczególnych rozkazach. Stosuje się kilka rozwiązań. Często różne kody operacyjne używają różnych trybów adresowania. Wobec tego jeden lub więcej bitów w formacie rozkazu może być użytych jako *pole trybu*. Wartość w polu trybu określa, który tryb adresowania jest używany.

TABELA 10.1. Podstawowe tryby adresowania

Tryb	Algorytm	Główna zaleta	Główna wada
Natychmiastowy	$\text{Argument} = A$	brak odniesienia do pamięci	ograniczona wielkość argumentu
Bezpośredni	$EA = A$	prostota	ograniczona przestrzeń adresowa
Pośredni	$EA = (A)$	duża przestrzeń adresowa	wiele odniesień do pamięci
Rejestrowy	$EA = R$	brak odniesienia do pamięci	ograniczona przestrzeń adresowa
Rejestrowy pośredni	$EA = (R)$	duża przestrzeń adresowa	dodatkowe odniesienie do pamięci
Z przesunięciem	$EA = A + (R)$	elastyczność	złożoność
Stosowe	$EA = \text{wierzchołek stosu}$	brak odniesienia do pamięci	ograniczona stosowalność

Druga uwaga dotyczy interpretacji *adresów efektywnych* (EA). W systemach bez pamięci wirtualnej adres efektywny jest albo adresem w pamięci głównej albo rejestrem. W systemie z pamięcią wirtualną adres efektywny jest adresem wirtualnym lub rejestrem. Rzeczywiste odwzorowanie na adres fizyczny jest funkcją mechanizmu stronicowania i jest niewidzialne dla programisty.

Adresowanie natychmiastowe

Najprostszą formą adresowania jest **adresowanie natychmiastowe**, w którym argument jest w rzeczywistości obecny w rozkazie.

$\text{ARGUMENT} = A$

Tryb ten może być stosowany do definiowania i używania stałych lub do ustalenia początkowych wartości zmiennych. Liczba jest zwykle przechowywana

w postaci uzupełnienia do dwóch; lewy bit pola argumentu jest używany jako bit znaku. Gdy argument jest ładowany do rejestru danych, bit znaku jest rozszerzany na lewo w celu wypełnienia słowa danych.

Zaletą adresowania natychmiastowego jest to, że żadne odniesienie do pamięci poza pobraniem rozkazu nie jest potrzebne do uzyskania argumentu, co pozwala *zaoszczędzić* jeden cykl pamięci głównej lub podręcznej w cyklu rozkazu. Wadą jest to, że rozmiar liczby jest ograniczony do rozmiaru pola adresowego, które w przypadku większości list rozkazów jest małe w porównaniu z długością słowa.

Adresowanie bezpośrednie

Bardzo prostą formą adresowania jest **adresowanie bezpośrednie**, w którym pole adresowe zawiera efektywny adres argumentu:

$$EA = A$$

Metoda ta była powszechna we wcześniejszych generacjach komputerów i jest nadal spotykana w małych systemach komputerowych. Wymaga tylko jednego odniesienia do pamięci i nie wymaga żadnych obliczeń. Oczywiście, wspomnianym wcześniej ograniczeniem jest to, że umożliwia ona obsługę tylko ograniczonej przestrzeni adresowej.

Adresowanie pośrednie

W przypadku adresowania pośredniego problemem jest to, że długość pola adresowego jest zwykle mniejsza niż długość słowa, co ogranicza zakres adresów. W jednym z rozwiązań pole adresowe odnosi się do słowa w pamięci, które z kolei zawiera pełnej długości adres argumentu. Metoda taka jest znana jako **adresowanie pośrednie**:

$$EA = (A)$$

(jak zdefiniowaliśmy wcześniej - nawiasy oznaczają zawartość). Oczywiście zaletą tego rozwiązania jest to, że przy długości słowa N dostępna jest przestrzeń adresowa 2^N . Wadą jest to, że wykonanie *rozkazu* wymaga dwóch odniesień do pamięci w celu pobrania argumentu: jednego do otrzymania adresu, a drugiego do uzyskania samej wartości.

Zauważmy, że chociaż *liczba* adresowalnych słów jest obecnie równa 2^N , to liczba różnych adresów efektywnych, do których można się odnosić w dowolnym czasie, jest ograniczona do 2^K , gdzie K jest długością pola adresowego. Zwykle nie jest to uciążliwym ograniczeniem, a może być nawet zaletą. W środowisku pamięci wirtualnej wszystkie lokacje adresów efektywnych mogą być ograniczone do strony O dowolnego procesu. Ponieważ pole adresowe *rozkazu* jest małe, w naturalny sposób będzie określało adresy bezpośrednie o niskich

numerach, które będą dotyczyły właśnie strony O (jedynym ograniczeniem jest to, że rozmiar strony musi być większy lub równy 2^k). Gdy proces jest aktywny, powtarzają się odniesienia do strony O, co powoduje pozostawanie jej w pamięci rzeczywistej. Wobec tego pośrednie odniesienie do pamięci może zawierać co najwyżej jeden błąd strony, a nie dwa.

Rzadko stosowanym wariantem adresowania pośredniego jest **adresowanie wielopoziomowe lub kaskadowe**:

$$EA = (\dots (A) \dots)$$

W tym przypadku jeden bit adresu obejmującego całe słowo jest znacznikiem pośredniości (I). Jeśli bit I jest 0, to słowo zawiera EA. Jeśli bit I jest 1, to wywoływany jest następny poziom pośredniości. Nie wygląda na to, żeby to rozwiązanie przynosiło szczególne korzyści, natomiast jego wadą jest to, że pobranie argumentu wymaga trzech lub więcej odniesień do pamięci.

Adresowanie rejestrowe

Adresowanie rejestrowe jest podobne do bezpośredniego. Jedyną różnicą jest to, że pole adresowe odnosi się do rejestru zamiast do adresu w pamięci głównej:

$$EA = R$$

Pole adresowe odnoszące się do rejestrów ma zwykle 3 lub 4 bity, wobec tego może się odwoływać do 8 lub 16 rejestrów roboczych.

Zaletą adresowania rejestrowego jest to, że (1) w rozkazie potrzebne jest tylko niewielkie pole adresowe, oraz (2) nie są wymagane odniesienia do pamięci. Jak stwierdziliśmy w rozdz. 4, czas dostępu do wewnętrznego rejestru procesora jest znacznie mniejszy niż czas dostępu do pamięci głównej. Wadą adresowania rejestrowego jest bardzo ograniczona przestrzeń adresowa.

Jeśli w liście rozkazów głównie używane jest adresowanie rejestrowe, oznacza to intensywne korzystanie z rejestrów procesora. Ze względu na poważnie ograniczoną liczbę rejestrów (w porównaniu z lokacjami w pamięci głównej), takie korzystanie ma sens tylko wtedy, gdy jest ono efektywne. Jeśli każdy argument jest doprowadzany do rejestru z pamięci głównej, dokonuje się na nim operacji, po czym jest on zwracany do pamięci głównej, to w ten sposób wprowadzamy dodatkowy, czasochłonny krok. Jeśli natomiast argument pozostaje w rejestrze i jest używany w wielu operacjach, osiągana jest realna oszczędność. Przykładem jest pośredni wynik obliczeń. W szczególności założmy, że wdrażany jest w postaci programu algorytm mnożenia w notacji uzupełnienia do dwóch. Do lokacji oznaczonej przez A w sieci działań (patrz rys. 8.11) następuje tak dużo odniesień, że powinna ona być wdrożona raczej w rejestrze niż jako lokacja w pamięci głównej.

Do programisty należy decyzja, które wartości powinny pozostawać w rejestrach, a które powinny być przechowywane w pamięci głównej. Większość

nowoczesnych procesorów zawiera wiele rejestrów ogólnego przeznaczenia, co ma na celu usprawnienie pracy programisty posługującego się językiem assemblerowym (np. piszącego kompilatory).

Pośrednie adresowanie rejestrowe

Podobnie jak adresowanie rejestrowe jest analogiczne do adresowania bezpośredniego, tak **pośrednie adresowanie rejestrowe** jest analogiczne do adresowania pośredniego. W obu przypadkach jedyną różnicą jest to, czy pole adresowe odnosi się do lokacji w pamięci, czy do rejestru. Wobec tego w przypadku pośredniego adresu rejestrowego

$$EA = (R)$$

Zalety i ograniczenia pośredniego adresowania rejestrowego są w zasadzie takie same, jak adresowania pośredniego. W obu przypadkach ograniczenie przestrzeni adresowej (ograniczony zakres adresów) pola adresowego jest eliminowane w ten sposób, że pole to odnosi się do zawierającej adres lokacji o długości słowa. Ponadto, stosując pośrednie adresowanie rejestrowe, używa się o jedno odniesienie do pamięci mniej niż przy adresowaniu pośrednim.

Adresowanie z przesunięciem

Bardzo efektywny tryb adresowania łączy możliwości adresowania bezpośredniego oraz pośredniego adresowania rejestrowego. Jest on znany pod wieloma nazwami zależnie od zastosowania, jednak podstawowy mechanizm jest taki sam. Będziemy go określali jako **adresowanie z przesunięciem** (ang. *displacement addressing*):

$$EA = A + (R)$$

Adresowanie z przesunięciem wymaga dwóch pól adresowych w rozkazie, przy czym przynajmniej jedno z nich jest jawne. Wartość zawarta w jednym polu adresowym (wartość równa A) jest używana bezpośrednio. Drugie pole adresowe, lub odniesienie domyślne oparte na kodzie operacji, odnosi się do rejestru, którego zawartość po dodaniu do A daje adres efektywny.

Opiszemy trzy najczęstsze zastosowania adresowania z przesunięciem:

- ☐ adresowanie względne,
- ☐ adresowanie z rejestrem podstawowym,
- a indeksowanie.

Adresowanie względne

W przypadku adresowania względnego rejestrem domyślnym jest licznik programu (PC). Oznacza to, że adres bieżącego rozkazu jest dodawany do pola adresowego w celu otrzymania EA. Zwykle dla celów tej operacji pole adresowe

jest traktowane jako liczba w notacji uzupełnienia do dwóch. Wobec tego adres efektywny jest przesunięty względem adresu zawartego w rozkazie.

Adresowanie względne opiera się na koncepcji lokalności przedyskutowanej w rozdz. 4 i 7. Jeśli większość odniesień do pamięci znajduje się stosunkowo blisko wykonywanego rozkazu, to używanie adresowania względnego pozwala na zaoszczędzenie bitów adresowych w rozkazie.

Adresowanie z rejestrem podstawowym

Interpretacja adresowania z rejestrem podstawowym jest następująca. Rejestr, do którego się odwołujemy, zawiera adres pamięci, a pole adresowe zawiera przesunięcie (zwykle w postaci liczby całkowitej bez znaku) w stosunku do tego adresu. Odniesienie do rejestru może być jawne lub domyślne.

Adresowanie z rejestrem podstawowym również wykorzystuje lokalność odniesień do pamięci. Jest wygodnym środkiem wdrażania segmentowania, które zostało przedyskutowane w rozdz. 7. W niektórych implementacjach jest stosowany pojedynczy rejestr podstawowy segmentu, co umożliwia stosowanie odniesień domyślnych. W innych implementacjach programista może wybrać rejestr przechowujący adres podstawowy segmentu, a rozkaz musi się do niego odnosić w sposób jawny. W tym ostatnim przypadku, jeśli długość pola adresowego wynosi K , a liczba możliwych rejestrów jest N , to jeden rozkaz może odnosić się do jednego z N obszarów zawierających po 2^K słów.

Indeksowanie

Interpretacja indeksowania jest zwykle następująca. Pole adresowe odnosi się do adresu w pamięci głównej, a wywołany rejestr zawiera dodatnie przesunięcie w stosunku do tego adresu. Zauważmy, że to rozwiązanie jest odwrotne w stosunku do adresowania z rejestrem podstawowym. Oczywiście chodzi o więcej niż tylko o interpretację użytkownika. Ponieważ w przypadku indeksowania pole adresowe jest traktowane jako adres w pamięci, zawiera ono zwykle więcej bitów niż pole adresowe w porównywalnym rozkazie dotyczącym rejestru podstawowego. Zobaczmy także, że istnieją pewne ulepszenia, które nie byłyby tak użyteczne w kontekście rejestru podstawowego. Mimo to metoda obliczania EA jest taka sama w przypadku indeksowania i w przypadku adresowania z rejestrem podstawowym. W obu przypadkach odniesienie do rejestru jest czasem jawne, a czasem domyślne (w różnych typach procesorów).

Ważnym zastosowaniem indeksowania jest umożliwienie efektywnego prowadzenia operacji iteracyjnych. Rozważmy na przykład listę liczb przechowywanych począwszy od lokacji A . Załóżmy, że chcemy dodać 1 do każdego elementu listy. Musimy pobrać każdą wartość, dodać do niej 1 i zwrócić ją do pamięci. Sekwencja efektywnych adresów, jakiej potrzebujemy, jest następująca $A, A+1, A+2, \dots$, aż do ostatniej lokacji listy. W przypadku indeksowania

można to łatwo uzyskać. Wartość A jest przechowywana w polu adresowym rozkazu, a wybrany rejestr, nazywany **rejestrem indeksowym**, jest inicjowany na 0. Po każdej operacji zawartość rejestru indeksu jest zwiększana o 1.

Ponieważ rejestry indeksowe są powszechnie używane do zadań iteracyjnych tego rodzaju, potrzeba zwiększania lub zmniejszania stanu rejestru indeksowego po każdym odniesieniu jest częsta. Ponieważ jest to tak powszechna operacja, niektóre systemy wykonują ją automatycznie, jako część tego samego cyklu rozkazu. Jest to znane jako **autoindeksowanie**. Jeśli pewne rejestry są przypisane wyłącznie do indeksowania, to autoindeksowanie może być wywoływane domyślnie i automatycznie. Jeśli są używane rejestry robocze, operacja autoindeksowania może wymagać sygnalizowania za pomocą bitu rozkazu. Autoindeksowanie oparte na przyrostach można przedstawić następująco:

$$EA = A + R$$

$$R \leftarrow (R) + 1$$

W niektórych maszynach możliwe jest zarówno adresowanie pośrednie, jak i indeksowanie. Możliwe jest też wykorzystywanie obu w tam samym rozkazie. Wówczas indeksowanie może być stosowane albo przed adresowaniem pośrednim, albo po nim.

Jeśli indeksowanie jest zastosowane po adresowaniu pośrednim, to jest określane jako **indeksowanie wtórne** (ang. *postindexing*):

$$EA = (A) + (R)$$

Najpierw zawartość pola adresowego jest użyta do uzyskania dostępu do lokacji pamięci zawierającej adres bezpośredni. Adres ten jest następnie indeksowany za pomocą wartości rejestru. Metoda ta jest użyteczna, jeśli chcemy uzyskać dostęp do jednego z pewnej liczby bloków danych o ustalonym formacie. Na przykład, jak mówiliśmy w rozdz. 7, system operacyjny dla każdego procesu wykorzystuje blok sterowania procesem. Wykonywane operacje są takie same niezależnie od tego, który blok podlega obróbce. Wobec tego adresy w rozkazach odnoszących się do bloku powinny wskazywać lokację (wartość = A) zawierającą zmienny wskaźnik początku bloku sterowania procesem. Rejestr indeksowy zawiera przemieszczenie wewnątrz bloku.

W przypadku **indeksowania wstępnego** (*preindexing*) indeksowanie jest realizowane przed adresowaniem pośrednim:

$$EA = (A + (R))$$

Adres jest obliczany tak, jak przy prostym indeksowaniu. Jednak w tym przypadku obliczany adres nie zawiera argumentu, lecz adres argumentu. Przykładem użycia tej metody jest tworzenie wielodrożnych tablic rozgałęzień. W określonym punkcie programu może następować rozgałęzienie do jednej z wielu lokacji, zależnie od warunków. Tablica adresów może być utworzona począwszy

od lokacji A. Drogą indeksowania może być odnaleziona wymagana lokacja w tablicy.

Zwykle lista rozkazów nie obejmuje ani indeksowania wstępnego, ani wtórnego.

Adresowanie stosowe

Ostatnim trybem adresowania, jaki rozpatrzymy, jest adresowanie stosowe. Jak zdefiniowaliśmy w dodatku 9A, stos jest liniową tablicą lokacji. Jest czasem określany jako lista „spychania” (ang. *pushdown Ust*) lub **kolejka ostatni na wejściu - pierwszy na wyjściu**. Stos jest zarezerwowanym blokiem lokacji. Elementy są dodawane na wierzchołek stosu w ten sposób, że w określonym czasie blok jest częściowo wypełniony. Ze stosem jest związany wskaźnik, którego wartością jest adres wierzchołka stosu. Dwa wierzchołkowe elementy stosu mogą też znajdować się w rejestrach procesora, a wtedy wskaźnik stosu odnosi się do trzeciego elementu stosu (rys. 9.14b). Wskaźnik stosu jest przechowywany w rejestrze. Wobec tego odniesienia do lokacji w stosie są w rzeczywistości bezpośrednimi adresami rejestru.

Stosowy tryb adresowania jest formą adresowania domyślnego. Rozkazy maszynowe nie muszą zawierać odniesienia do pamięci, lecz domyślnie operują na wierzchołku stosu. Tradycyjnie stosy nie były powszechne, jednak stają się powszechne w systemach mikroprocesorowych.

Tryby adresowania Pentium

Przypomnijmy sobie na podstawie rys. 7.25, że mechanizm translacji adresu procesora Pentium tworzy adres zwany wirtualnym lub efektywnym, który jest adresem względnym w segmencie. Suma adresu początkowego segmentu i adresu efektywnego daje adres liniowy. Jeśli jest używane stronicowanie, adres liniowy musi przejść przez mechanizm translacji strony, w wyniku czego otrzymuje się adres fizyczny. W dalszym ciągu będziemy ignorowali ten ostatni krok, ponieważ jest on przezroczysty dla listy rozkazów i dla programisty.

Procesor Pentium jest wyposażony w różne tryby adresowania w celu umożliwienia efektywnej realizacji programów w językach wysokiego poziomu, takich jak C i FORTRAN. Na rysunku 10.2 jest pokazany schemat adresowania. Segment będący przedmiotem odniesienia jest określony przez rejestr segmentowy. Istnieje sześć rejestrów segmentowych; użycie jednego z nich do określonego odniesienia zależy od rozkazu i od kontekstu jego wykonywania. Każdy rejestr segmentowy zawiera adres początkowy odpowiedniego segmentu. Z każdym widzialnym dla użytkownika rejestrem segmentowym jest związany rejestr deskryptora segmentu (niewidzialny dla programisty), który zawiera zapis praw dostępu do segmentu, a także adres początkowy i granicę (długość) segmentu. Ponadto istnieją dwa rejestry, które mogą być użyte do budowania adresu: rejestr podstawowy i rejestr indeksowy (*index register*).

TABELA 10.2. Tryby adresowania procesora Pentium

Tryb	Algorytm
Natychmiastowy	Argument = A
Rejestrowy	LA = R
Z przesunięciem	LA = (SR) + A
Z rejestrem podstawowym	LA = (SR) + (B)
Z rejestrem podstawowym z przesunięciem	LA = (SR) + (B) + A
Skalowane indeksowanie z przesunięciem	LA = (SR) + (I) x S + A
Z rejestrem podstawowym z indeksem i z przesunięciem	LA = (SR) + (B) + (I) + A
Z rejestrem podstawowym ze skalowanym indeksowaniem i z przesunięciem	LA = (SR) + (I) x S + (B) + A
Względny	LA = (PC) + A

LA - adres liniowy

(X) - zawartość X

SR - rejestr segmentowy

PC - licznik rozkazów

A - zawartość pola adresowego w rozkazie

R - rejestr

B - rejestr podstawowy

I - rejestr indeksowy

S - czynnik skalowania

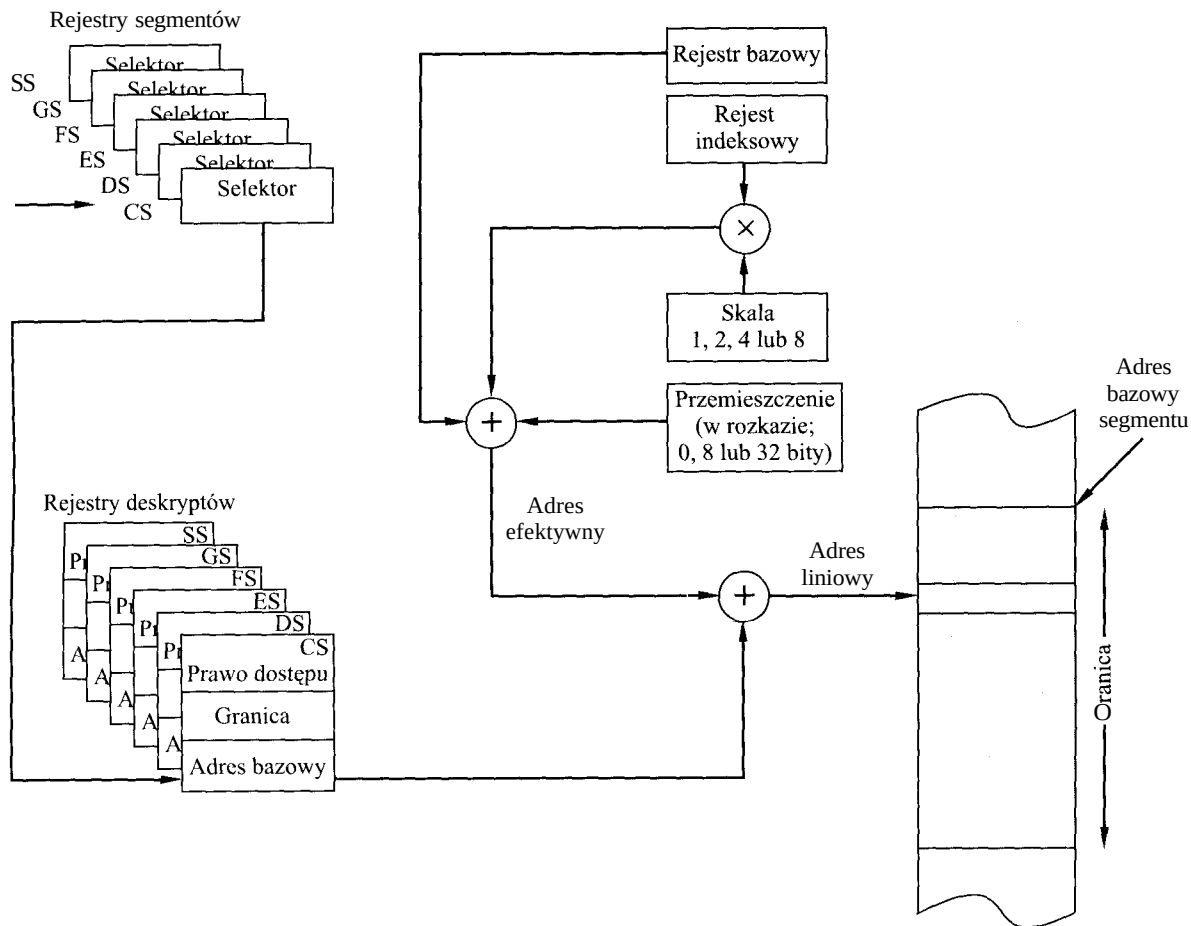
W tabeli 10.2 jest wymienionych 12 trybów adresowania Pentium. Rozpatrzymy kolejno każdy z nich.

W przypadku **trybu natychmiastowego** argument jest włączony do rozkazu. Argument może być bajtem, słowem lub podwójnym słowem danych.

W przypadku **trybu argumentu rejestrowego** (tryb rejestrowy) argument jest umieszczony w rejestrze. Jeśli mamy do czynienia z rozkazami ogólnymi, takimi jak rozkazy transferu danych oraz rozkazy arytmetyczne i logiczne, argument może być umieszczany w jednym z 32-bitowych rejestrów roboczych (EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP), w jednym z 16-bitowych rejestrów roboczych (AX, BX, CX, DX, SI, DI, SP, BP) lub w jednym z 8-bitowych rejestrów roboczych (AH, BH, CH, DH, AL, BL, CL, DL). Dla celów operacji zmiennopozycyjnych argumenty 64-bitowe są formowane w pary za pomocą dwóch rejestrów 32-bitowych. Są również pewne rozkazy, które odwołują się do rejestrów segmentowych (CS, DS, ES, SS, FS, GS).

Pozostałe tryby adresowania odnoszą się do lokacji w pamięci. Lokacje te muszą być określane przez podanie segmentu oraz adresu względnego liczonego od początku segmentu. W pewnych przypadkach segment jest określany jawnie, w innych zaś jest określany za pomocą prostych reguł, które przypisują segment zaocznie.

W **trybie z przesunięciem** adres względny argumentu (adres efektywny na rys. 10.2) jest częścią rozkazu jako przesunięcie 8-, 16- lub 32-bitowe. W przypadku segmentacji wszystkie adresy w rozkazach odnoszą się jedynie do adresu względnego w segmencie. Tryb adresowania z przesunięciem jest spotykany w niewielu maszynach, ponieważ -jak stwierdziliśmy wcześniej - prowadzi on



RYSUNEK 10.2. Obliczanie trybu adresowania Pentium

do długich rozkazów. W przypadku Pentium wartość przesunięcia może wynosić do 32 bitów, co wymaga 6-bajtowego rozkazu. Adresowanie z przesunięciem może być użyteczne przy odnoszeniu się do zmiennych globalnych.

Pozostałe tryby adresowania są pośrednie w tym sensie, że adresowa część rozkazu mówi procesorowi, gdzie należy szukać adresu. Tryb podstawowy ustala, że jeden z rejestrów 8-, 16- lub 32-bitowych zawiera adres efektywny. Jest to równoważne temu, co określiliśmy jako pośrednie adresowanie rejestrowe.

W trybie **z rejestrem podstawowym i z przesunięciem** rozkaz zawiera przesunięcie, które jest dodawane do rejestru podstawowego. Rejestrem podstawowym może być dowolny z rejestrów roboczych. Oto przykłady stosowania tego trybu:

1. Jest używany przez kompilator do wyznaczania początku obszaru zmiennej lokalnej. Na przykład rejestr podstawowy może wskazać początek ramki stosu, która zawiera zmienne lokalne odpowiedniej procedury.
2. Jest używany do indeksowania tablicy, której elementy mają różny rozmiar (od 2, 4 lub 8 bajtów) i która wobec tego nie może być indeksowana za pomocą rejestru indeksowego. W tym przypadku przesunięcie wskazuje początek tablicy, a rejestr podstawowy zawiera wyniki obliczeń określające adres względny elementu wewnątrz tablicy.
3. Jest używany w celu uzyskiwania dostępu do pola rekordu. Rejestr podstawowy wskazuje początek rekordu, a przesunięcie jest adresem względnym pola.

W trybie **skalowanego indeksowania z przesunięciem** rozkaz zawiera przesunięcie, które jest dodawane do rejestru, w tym przypadku nazywanego rejestrem indeksowym. Rejestrem indeksowym może być dowolny z rejestrów roboczych z wyjątkiem ESP, który jest na ogół używany do przetwarzania stosowego. Obliczanie adresu efektywnego polega na pomnożeniu zawartości rejestru indeksowego przez czynnik 1, 2, 4 lub 8, a następnie na dodaniu wyniku do przesunięcia. Tryb ten jest bardzo wygodny do indeksowania tablic. Czynnik skalowania 2 może być użyty w przypadku tablic 16-bitowych liczb całkowitych, czynnik 4 - w przypadku 32-bitowych liczb całkowitych lub zmiennopozycyjnych. Wreszcie czynnik 8 może być użyty w przypadku tablic liczb zmiennopozycyjnych o podwójnej dokładności.

W trybie **indeksowym z rejestrem podstawowym i z przesunięciem** sumuje się zawartość rejestru podstawowego, rejestru indeksowego oraz przesunięcia w celu utworzenia adresu efektywnego. Znowu rejestrem podstawowym może być dowolny z rejestrów roboczych, a rejestrem indeksowym - dowolny z rejestrów roboczych z wyjątkiem ESP. Tryb ten może być na przykład używany do adresowania lokalnej tablicy w ramce stosu. Może być także używany do obsługi tablic dwuwymiarowych; w tym przypadku przesunięcie wskazuje początek tablicy, a każdy rejestr dotyczy jednego wymiaru tablicy.

W trybie z przesunięciem, ze skalowanym indeksowaniem i z rejestrem podstawowym sumuje się zawartość rejestru indeksowego, pomnożoną przez czynnik skalowania, z zawartością rejestru podstawowego i przesunięciem. Jest to użyteczne, gdy tablica jest przechowywana w ramce stosu; w tym przypadku długość elementów tablicy powinna wynosić po 2, 4 lub 8 bajtów. Tryb ten zapewnia również efektywne indeksowanie tablicy dwuwymiarowej, gdy elementy tablicy mają długość po 2, 4 lub 8 bajtów.

Wreszcie **adresowanie względne** może być używane w rozkazach przekazywania sterowania. Przesunięcie jest dodawane do wartości licznika programu, który wskazuje następny rozkaz. W tym przypadku przesunięcie jest traktowane jako bajt, słowo lub podwójne słowo ze znakiem, a jego wartość albo zwiększa, albo zmniejsza adres w liczniku programu.

Tryby adresowania PowerPC

Podobnie jak większość maszyn RISC oraz w przeciwieństwie do Pentium i większości maszyn CISC, procesor PowerPC używa stosunkowo prostego zestawu trybów adresowania. Jak widać w tabeli 10.3, tryby te są dogodnie poklasyfikowane ze względu na rodzaj rozkazu.

TABELA 10.3. Tryby adresowania procesora PowerPC

Tryb	Algorytm
	<i>Adresowanie lądowania/zapisu</i>
Pośredni	$EA = (BR) + D$
Pośredni indeksowany	$EA = (BR) + (IR)$
	<i>Adresowanie rozgałęzień</i>
Bezwzględny	$EA = I$
Względny	$EA = (PC) + I$
Pośredni	$EA = (L/CR)$
	<i>Obliczenia stałopozycyjne</i>
Rejestrowy	$EA = GPR$
Natychmiastowy	$Argument = I$
	<i>Obliczenia zmiennopozycyjne</i>
Rejestrowy	$EA = FPR$

EA - adres efektywny

(X) - zawartość X

BR - rejestr podstawowy

IR - rejestr indeksowy

L/CR - rejestr powiązań lub zliczający

GPR - rejestr roboczy

FPR - rejestr zmiennopozycyjny

D - przesunięcie

I - wartość natychmiastowa

PC - licznik rozkazów

Adresowanie ładowania/zapisu

W procesorze PowerPC przewidziano dwa alternatywne tryby adresowania dla rozkazów ładowania/zapisu (rys. 10.3). W przypadku **adresowania pośredniego** rozkaz zawiera 16-bitowe przesunięcie dodawane do rejestru podstawowego, którym może być dowolny z rejestrów roboczych. Ponadto rozkaz może ustalać, że nowo obliczony adres efektywny ma być skierowany do rejestru podstawowego, aktualizując jego zawartość. Opcja aktualizacji jest użyteczna w przypadku progresywnego indeksowania tablic w pętach.

Inną metodą adresowania w rozkazach ładowania/zapisu jest **pośrednie adresowanie indeksowane**. W tym przypadku rozkaz odnosi się do rejestru podstawowego i rejestru indeksowego, z których oba mogą być dowolnym rejestrem roboczym. Adres efektywny jest sumą zawartości tych dwóch rejestrów. Jak poprzednio opcja aktualizacji umożliwia wprowadzenie nowego adresu efektywnego do rejestru podstawowego.

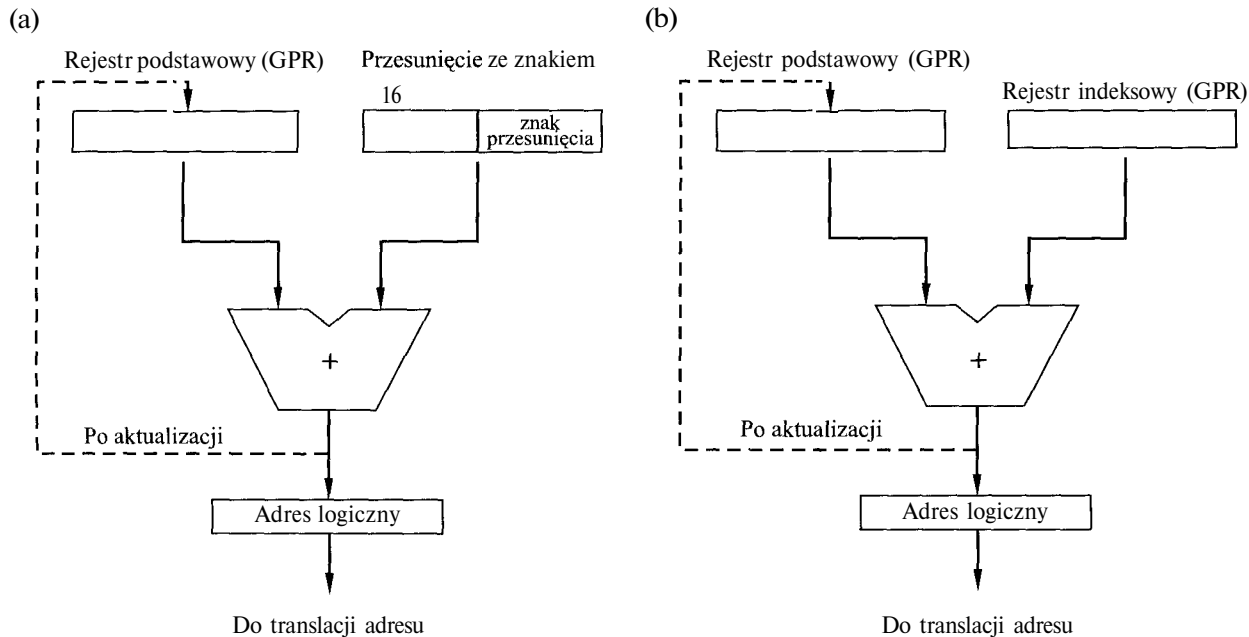
Adresowanie rozgałęzienia

Przewidziane są trzy tryby adresowania rozgałęzienia. Gdy używane jest **adresowanie bezwzględne** w połączeniu z rozkazem rozgałęzienia bezwarunkowego, adres efektywny następnego rozkazu jest otrzymywany z 24-bitowej wartości natychmiastowej (ang. *immediate value*) zawartej w rozkazie. Wartość ta jest rozszerzana do 32 bitów przez dodanie dwóch zer po stronie najmniej znaczących bitów (co jest dopuszczalne, ponieważ wszystkie rozkazy muszą występować w 32-bitowych granicach) oraz przez poszerzenie znaku. W przypadku rozkazów rozgałęzienia warunkowego adres efektywny następnego rozkazu jest wyprowadzany z 16-bitowej wartości natychmiastowej zawartej w rozkazie. Wartość 16-bitowa jest rozszerzana do 32 bitów przez dodanie dwóch zer po stronie najmniej znaczących bitów oraz rozszerzenie znaku.

W przypadku **adresowania względnego** wartość natychmiastowa 24-bitowa (rozkazy rozgałęzienia bezwarunkowego) lub 14-bitowa (rozkazy rozgałęzienia warunkowego) jest rozszerzana jak poprzednio. Wartość wynikowa jest następnie dodawana do licznika programu w celu określenia lokacji w stosunku do bieżącego rozkazu. Innym trybem adresowania rozgałęzienia warunkowego jest **adresowanie pośrednie**. W trybie tym adres następnego rozkazu jest uzyskiwany albo z rejestru powiązania (ang. *link register*), albo z rejestru zliczającego (ang. *count register*). Zauważmy, że w tym przypadku do przechowywania adresu rozkazu rozgałęzienia jest używany rejestr zliczający. Rejestr ten może też być używany do zliczania w pętli, co wyjaśniliśmy wcześniej.

Rozkazy arytmetyczne

W przypadku arytmetyki liczb całkowitych wszystkie argumenty muszą być zawarte albo w rejestrach, albo w samych rozkazach. Przy **adresowaniu rejestrowym** argumenty źródłowe lub docelowe są określane jako zawartość jedne-



RYSUNEK 10.3. Tryby adresowania argumentów w pamięci PowerPC [DIEF94a]:

(a) adresowanie pośrednie; (b) adresowanie pośrednie indeksowe

go z rejestrów roboczych. Przy **adresowaniu natychmiastowym** argument źródłowy występuje w rozkazie jako 16-bitowa wielkość ze znakiem.

W przypadku arytmetyki zmiennopozycyjnej wszystkie argumenty są zawarte w rejestrach zmiennopozycyjnych; oznacza to, że jest stosowane tylko adresowanie rejestrowe.

0.2

FORMATY ROZKAZÓW

Format rozkazu określa rozkład bitów rozkazu odniesiony do jego części składowych. Format rozkazu musi zawierać kod operacji oraz, w sposób jawny lub domyślny, zero lub więcej argumentów. Każdy jawny argument jest adresowany za pomocą jednego z trybów opisanych w podrozdz. 10.1. Format musi, w sposób jawny lub domyślny, wskazywać tryb adresowania każdego argumentu. W większości list rozkazów występuje więcej niż jeden format rozkazu.

Projektowanie formatu rozkazu jest sztuką złożoną i wdrożono dotychczas zdumiewającą różnorodność rozwiązań. W tym podrozdziale przeanalizujemy podstawowe zagadnienia projektowania, odwołując się krótko do pewnych przykładów, po czym szczegółowo przedstawimy rozwiązania przyjęte w procesorach Pentium i PowerPC.

Długość rozkazu

Najbardziej podstawowym problemem projektowym jest długość formatu rozkazu. Decyzja ta wpływa na rozmiar pamięci, organizację pamięci, strukturę magistrali, złożoność i szybkość procesora. Z kolei te czynniki mają wpływ na tę decyzję. Decyzja dotycząca długości rozkazu określa bogactwo i elastyczność maszyny, widziane przez programistę posługującego się językiem assemblerowym.

Najbardziej oczywisty jest kompromis między żądaniem rozbudowanego repertuaru rozkazów a potrzebą oszczędzania przestrzeni pamięci. Programiści domagają się więcej kodów operacji, więcej argumentów, więcej trybów adresowania i większego zakresu adresów. Więcej kodów operacji i więcej argumentów ułatwia życie programiście, ponieważ do osiągnięcia postawionego celu wystarczą krótsze programy. Podobnie więcej trybów adresowania daje programiście większą elastyczność we wdrażaniu pewnych funkcji, takich jak manipulowanie tablicami i wielodrożne rozgałęzienia. Oczywiście jest też, że ze wzrostem pojemności pamięci głównej i przy wzrastającym wykorzystaniu pamięci wirtualnych, programista chciałby móc adresować coraz większe zakresy pamięci. Wszystkie te *rzeczy* (kody operacji, argumenty, tryby adresowania, zakres adresów) wymagają bitów i popychają w kierunku dłuższych rozkazów. Jednak dłuższe rozkazy mogą prowadzić do rozrzutności. Rozkaz 32-bitowy zajmuje dwa razy tyle przestrzeni, co 16-bitowy, jednak prawdopodobnie jego użyteczność wzrasta znacznie mniej niż dwukrotnie.

Obok tego podstawowego kompromisu istnieją jeszcze inne. Albo długość rozkazu powinna być równa długości jednostki transferu z (i do) pamięci (w systemie magistralowym - szerokości magistrali danych), albo jedno powinno być wielokrotnością drugiego. W przeciwnym razie nie otrzymamy całkowitej liczby rozkazów podczas cyklu pobierania. Problemem z tym związanym jest szybkość transferu z (i do) pamięci. Szybkość ta nie nadążała za wzrostem szybkości procesora. Wobec tego pamięć może się stać wąskim gardłem, jeśli procesor może szybciej wykonywać rozkazy, niż je pobierać. Jednym z rozwiązań tego problemu jest użycie pamięci podręcznej (patrz podrozdz. 4.3); innym jest używanie krótszych rozkazów. Jak poprzednio, rozkazy 16-bitowe mogą być pobierane dwukrotnie szybciej niż 32-bitowe, jednak prawdopodobnie nie mogą być wykonywane dwa razy szybciej.

Ewidentnie powszechną, lecz mimo to ważną własnością jest to, że długość rozkazu powinna być wielokrotnością długości znaku, która zwykle wynosi 8 bitów, oraz długości liczb stałopozycyjnych. Aby to dostrzec, musimy posłużyć się tym tak nieszczęśliwie zdefiniowanym pojęciem, jakim jest „słowo” [FRAI83]. Długość słowa pamięci jest w pewnym sensie „naturalną” jednostką organizacji. Rozmiar słowa zwykle determinuje rozmiar liczb stałopozycyjnych (zwykle te dwie jednostki są równe). Rozmiar słowa jest zwykle równy rozmiarowi jednostki transferu z/do pamięci albo przynajmniej jest z nim integralnie związany. Ponieważ powszechną formą danych są dane znakowe, chcielibyśmy dysponować słowem, w którym może być przechowywana całkowita liczba znaków. W przeciwnym razie przechowywanie wielu znaków pociągnie za sobą stratę bitów w każdym słowie albo znak będzie wykraczał poza granice słowa. Znaczenie tego problemu jest tak duże, że gdy IBM wprowadzał System/370 i chciał używać znaków 8-bitowych, podjął bolesną decyzję odejścia od 36-bitowej architektury naukowych komputerów z serii 700/7000 i przyjęcia architektury 32-bitowej.

Przydział bitów

Rozpatrzyliśmy niektóre z czynników wpływających na decyzje dotyczące długości formatu rozkazu. Równie trudnym problemem jest przydział bitów w tym formacie. Mamy tu do czynienia ze skomplikowanymi kompromisami.

Przy danej długości rozkazu istnieje oczywista wymiennność między liczbą kodów operacji a zdolnością adresowania. Więcej kodów operacji oznacza oczywiście więcej bitów w polu kodu operacji. W przypadku formatu rozkazu o danej długości następuje zredukowanie liczby bitów dostępnych na potrzeby adresowania. Istnieje interesujące ulepszenie dotyczące tego problemu: jest nim użycie kodów operacji o zmiennej długości. W tym rozwiązaniu występuje minimalna długość kodu operacji, jednak w przypadku niektórych kodów mogą być specyfikowane dodatkowe operacje za pomocą dodatkowych bitów w rozkazie. Przy rozkazach o ustalonej długości pozostawia to mniej bitów na adre-

sowanie. Własność ta jest wobec tego używana w odniesieniu do tych rozkazów, które wymagają mniej argumentów i (lub) mniej rozbudowanego adresowania. Zapoznamy się z przykładami tej strategii.

Następujące wzajemnie powiązane czynniki muszą być uwzględniane przy decydowaniu o użyciu bitów na potrzeby adresowania:

- ❑ *Liczba trybów adresowania.* Czasem tryb adresowania może być domyślny. Na przykład pewne kody operacji mogą zawsze wymagać indeksowania. W innych przypadkach tryby adresowania muszą być jawne, do czego jest potrzebny jeden lub więcej bitów.
- a *Liczba argumentów.* Widzieliśmy, że zmniejszona liczba adresów może prowadzić do dłuższych, bardziej kłopotliwych programów (np. rys. 9.3). Typowe rozkazy współczesnych maszyn przewidują dwa argumenty. Każdy adres argumentu w rozkazie może wymagać własnego wskaźnika trybu lub też wskaźnik trybu może być związany tylko z jednym polem adresowym.
- ❑ *Rejestry a pamięć.* Maszyna musi mieć rejestry, dzięki którym dane mogą być wprowadzane do procesora w celu przetwarzania. W przypadku rejestru pojedynczego, widzialnego dla użytkownika (nazywanego zwykle akumulatorem), adres jednego argumentu jest domyślny i nie pochłania bitów rozkazu. Jednak programowanie jednorejestrowe jest kłopotliwe i wymaga wielu rozkazów. Nawet w przypadku wielu rejestrów potrzeba tylko kilku bitów do określenia rejestru. Im więcej rejestrów można przeznaczyć na odniesienia do argumentów, tym mniej potrzeba bitów. W wielu badaniach wykazano, że pożądana liczba rejestrów widzialnych dla użytkownika wynosi od 8 do 32 [LUND77], [HUCK83].
- a *Liczba zestawów rejestrów.* Wiele maszyn ma jeden zestaw rejestrów roboczych, zawierający zwykle 8 do 16 rejestrów. Mogą one być używane do przechowywania danych albo też do przechowywania adresów w przypadku adresowania z przesunięciem. Współcześnie występuje tendencja do odchodzenia od jednego zestawu rejestrów roboczych w kierunku dwóch lub większej liczby zestawów wyspecjalizowanych (np. dane i przesunięcia). Tendencja ta występuje wszędzie, od mikroprocesorów jednostrukturalnych do superkomputerów. Zaletą tego rozwiązania jest to, że przy ustalonej liczbie rejestrów ich podział funkcjonalny oznacza mniej bitów potrzebnych w rozkazie. Na przykład przy dwóch zestawach po 8 rejestrów do identyfikowania rejestru potrzebne są tylko 3 bity; kod operacyjny domyślnie określa, do którego zestawu rejestrów następuje odniesienie. Wygląda na to, że rozwiązanie to ma niewiele wad [LUND77]. W systemach takich jak S/370, które mają jeden zestaw rejestrów roboczych, programiści zwykle ustanawiają konwencję, przypisującą około połowy rejestrów danym a pozostałe przesunięciom i pozostawiają do przypisania na stałe [MALL79].
- a *Zakres adresów.* W przypadku adresów odnoszących się do pamięci zakres możliwych adresów jest powiązany z liczbą bitów adresowych. Ponieważ stanowi to poważne ograniczenie, bezpośrednie adresowanie jest rzadko

stosowane. Dzięki adresowaniu z przesunięciem zakres adresów ulega poszerzeniu aż do wielkości wynikającej z długości rejestru adresowego. Nawet w tym przypadku nadal wygodne jest stosowanie raczej dużych przesunięć w stosunku do adresu rejestrowego, co wymaga stosunkowo dużej liczby bitów adresowych w rozkazie.

Q *Stopień granulacji* (ang. *granularity*) *adresu*. W przypadku adresów, które odnoszą się raczej do pamięci niż do rejestrów, dodatkowym czynnikiem jest stopień granulacji adresu. W systemie o słowach 16- lub 32-bitowych adres może odnosić się do słowa lub bajta, zależnie od wyboru projektanta. Adresowanie bajtowe jest wygodne do manipulowania znakami, jednak przy ustalonym rozmiarze pamięci wymaga więcej bitów adresowych.

Wobec tego projektant ma do czynienia z całym zastępem czynników, które musi przeanalizować i zrównoważyć. Nie jest jasne, jak krytyczne są różne wybory. Zacytujemy na przykład jedno z badań [CRAG79], w którym porównano różne rozwiązania formatu rozkazu, włącznie z użyciem stosu, rejestrów roboczych, akumulatora i rozkazów typu pamięć-rejestr. Stosując spójny zespół założeń, nie stwierdzono znaczących różnic przestrzeni kodowej lub czasu realizacji.

Zapoznajmy się krótko z równoważeniem tych różnych czynników, które zastosowano w dwóch projektach maszyn.

PDP-8

Jednym z najprostszych projektów rozkazów dla komputera uniwersalnego było rozwiązanie przyjęte dla PDP-8 [BELL78b]. Komputer PDP8 używa rozkazów 12-bitowych i operuje na słowach 12-bitowych. Występuje w nim jeden rejestr roboczy - akumulator.

Mimo ograniczeń tego projektu, adresowanie jest całkiem elastyczne. Każde odniesienie do pamięci obejmuje 7 bitów plus 2 modyfikatory 1-bitowe. Pamięć jest podzielona na strony o ustalonej długości, po $2^7 = 128$ słów każda. Obliczanie adresu opiera się na odniesieniu do strony O lub strony bieżącej (zawierającej dany rozkaz), zależnie od bitu strony. Drugi modyfikator 1-bitowy wskazuje, czy stosowane jest adresowanie bezpośrednie, czy pośrednie. Oba tryby mogą być używane w połączeniu, tak że adres pośredni jest 12-bitowym adresem zawartym w słowie strony O lub strony bieżącej. Ponadto, 8 dedykowanych słów na stronie O stanowi automatycznie indeksowane „rejestry”. Gdy ma miejsce pośrednie odniesienie do tych lokacji, następuje wstępne indeksowanie.

Na rysunku 10.4 jest pokazany format rozkazu PDP-8. Występuje tu 3-bitowy kod operacji i trzy rodzaje rozkazów. Przy kodach od 0 do 5 formatem jest rozkaz z 1-adresowym odniesieniem do pamięci, zawierający bit strony i bit pośredniości. Wobec tego występuje tylko 6 operacji podstawowych. W celu poszerzenia grupy operacji kod operacji 7 określa odniesienie do rejestru lub **mikrorozkaz**. W tym formacie pozostałe bity służą do kodowania dodatkowych

Rozkazy z odniesieniem do pamięci

Kod operacji	D/I	Z/C	Przemieszczenie			
0	2	3	4	5		11

Rozkazy wejścia/wyjścia

1	1	0	Urządzenie				Kod operacji		
0			2	3			8	9	11

Rozkazy z odniesieniem do rejestrów

Mikrorozkazy grupy 1

1	1	1	0	CLA	CLL	CMA	CML	RAR	RAL	BSW	IAC
0	1	2	3	4	5	6	7	8	9	10	11

Mikrorozkazy grupy 2

1	1	1	1	CLA	SMA	SZA	SNL	RSS	OSR	HLT	0
0	1	2	3	4	5	6	7	8	9	10	11

Mikrorozkazy grupy 3

1	1	1	1	CLA	MQA	0	MQL	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11

Mnemoniki

CLA - wyczyść akumulator

CLL - wyczyść łącze

CMA - dopełnij akumulator

CML - dopełnij łącze

RAR - wykonaj rotację akumulatora w prawo

RAL - wykonaj rotację akumulatora w lewo

BSW - wymień bajt

IAC - inkrementuj akumulator

SMA - przeskocz na minus akumulator

SZA - przeskocz na zero akumulator

SNL - przeskocz na łącze niezerowe

RSS - odwróć odczyt przeskoku

OSR - wykonaj OR z rejestrem przełączania

HLT - zatrzymaj

MQA - mnożnik/iloraz do akumulatora

MQL - ładuj mnożnik/iloraz

RYSUNEK 10.4. Formaty rozkazu PDP-8

operacji. Na ogół każdy bit określa specyficzną operację (np. wyczyszczenie akumulatora), a bity te mogą być łączone w jednym rozkazie. Strategia mikroprogramowania była stosowana przez DEC jeszcze w PDP-1 i jest w pewnym sensie zapowiedzią dzisiejszych maszyn mikroprogramowanych, które będą przedyskutowane w części IV. Kod operacji 6 dotyczy wejścia-wyjścia; 6 bitów używa się do wyboru jednego z 64 urządzeń, a 3 bity określają konkretny rozkaz wejścia-wyjścia.

Efektywność formatu rozkazu PDP-8 jest godna uwagi. Umożliwia on adresowanie pośrednie, adresowanie z przesunięciem i indeksowanie. Wraz z rozszerzeniem kodu operacji obsługuje on łącznie około 35 rozkazów. Biorąc pod uwagę ograniczenia wynikające z 12-bitowego rozkazu, projektanci nie mogli spisać się lepiej.

PDP-10

Z listą rozkazów PDP-8 silnie kontrastuje lista PDP-10. PDP-10 został zaprojektowany jako duży system pracujący z podziałem czasu, z akcentem na łatwość programowania, nawet kosztem dodatkowych wydatków sprzętowych.

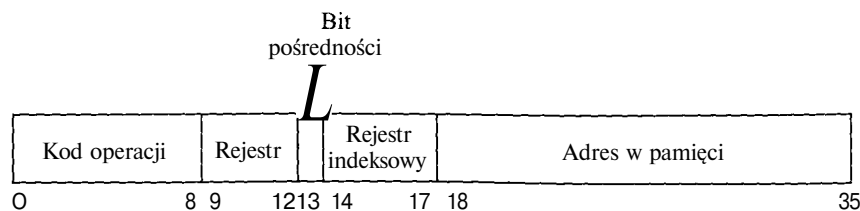
Do zasad wykorzystanych do zaprojektowania listy rozkazów należały [BELL78c]:

- a *Ortogonalność*. Ortogonalność jest zasadą, zgodnie z którą dwie zmienne są od siebie niezależne. W kontekście listy rozkazów termin ten oznacza, że inne elementy rozkazu są niezależne od kodu operacji (nie są przez ten kod determinowane). Projektanci PDP-10 użyli tego terminu w celu podkreślenia faktu, że adres jest zawsze obliczany w ten sam sposób, niezależnie od kodu operacji. Stanowi to kontrast w stosunku do wielu maszyn, w których tryb adresowania wynika czasem domyślnie z użytego operatora.
- a *Kompletność*. Każdy rodzaj danych arytmetycznych (całkowite, stałopozycyjne, rzeczywiste) powinien mieć kompletną i identyczną listę operacji.
- *Adresowanie bezpośrednie*. Zrezygnowano z adresowania przesunięciowego z rejestrem podstawowym, obciążającego programistę problemami organizacji pamięci, na rzecz adresowania bezpośredniego.

Głównym celem każdej z tych zasad było ułatwienie programowania.

PDP-10 ma słowo 36-bitowe oraz 36-bitowy rozkaz. Ustalony format rozkazu widać na rys. 10.5. Kod operacji zajmuje 9 bitów, co pozwala na 512 operacji. W rzeczywistości zdefiniowano 365 różnych rozkazów. Większość rozkazów ma dwa adresy, z których jeden znajduje się w jednym z 16-bitowych rejestrów roboczych. Wobec tego odniesienie do tego argumentu zajmuje 4 bity. Odniesienie do drugiego argumentu rozpoczyna się od 18-bitowego pola adresu pamięci. Może być ono używane jako argument natychmiastowy lub jako adres pamięci. W tym ostatnim przypadku dozwolone jest indeksowanie oraz adresowanie pośrednie. Te same rejestry robocze są używane jako rejestry indeksowe.

Rozkaz długości 36-bitów jest rzeczywistym luksusem. Nie ma potrzeby sprytnych rozwiązań zwiększających liczbę kodów operacji; 9-bitowe pole kodu jest więcej niż wystarczające. Adresowanie jest także proste. Atrakcyjne jest adresowanie bezpośrednie ze względu na 18-bitowe pole adresowe. Dla pamięci większych niż 2^{18} przewidziano adresowanie pośrednie. Aby ułatwić progra-



RYSUNEK 10.5. Formaty rozkazu PDP-10

mowanie, istnieje możliwość indeksowania przy manipulowaniu tablicami i wykonywaniu programów iteracyjnych. Ponieważ pole argumentu ma 18 bitów, staje się atrakcyjne adresowanie natychmiastowe.

Projekt listy rozkazów PDP-10 osiągnął zapowiedziane cele [LUND77]. Lista jest stosunkowo łatwa z punktu widzenia programisty, kosztem nieefektywnego wykorzystania przestrzeni. Był to świadomy wybór dokonany przez projektantów i dlatego nie może być dyskwalifikowany jako słaby projekt.

Rozkazy o zmiennej długości

W dotychczasowych przykładach występowała jedna, ustalona długość rozkazów; przedyskutowane kompromisy dotyczyły tego właśnie kontekstu. Jednak projektanci mogą wybrać różne formaty rozkazów o różnych długościach. Taka taktyka ułatwia uzyskanie dużego repertuaru kodów operacji, o różnych długościach kodów. Adresowanie może być bardziej elastyczne przy różnych kombinacjach odniesień do rejestrów i do pamięci oraz przy różnych trybach adresowania. Rozkazy o zmiennej długości ułatwiają efektywne i zwarte wprowadzanie tych odmian.

Ceną, jaką płacimy za przyjęcie rozkazów o zmiennej długości jest wzrost złożoności procesora. Malejące ceny sprzętu, zastosowanie mikroprogramowania (przedyskutowane w części IV) oraz ogólnie lepsze rozumienie zasad projektowania procesorów przyczyniły się do tego, że tę niewielką cenę warto zapłacić.

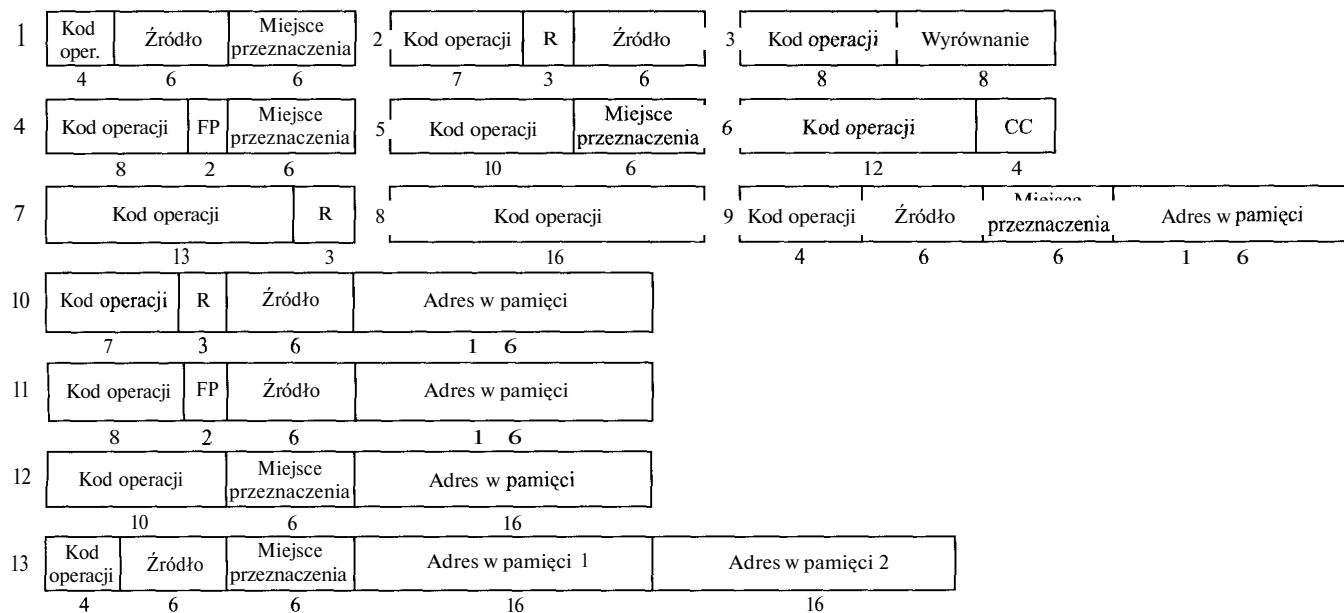
Zastosowanie rozkazów o zmiennej długości nie eliminuje żądania, aby wszystkie długości rozkazów były integralnie związane z długością słowa. Ponieważ procesor nie zna długości następnego rozkazu przewidzianego do pobrania, typową strategią jest pobieranie liczby bajtów lub słów równej przynajmniej najdłuższemu możliwemu rozkazowi. Oznacza to, że czasem pobiera się wiele rozkazów. Jak jednak zobaczymy w rozdz. 11, jest to w dowolnym przypadku strategia właściwa.

PDP-11

PDP-11 został zaprojektowany w celu zapewnienia skutecznej i elastycznej listy rozkazów w ramach ograniczeń minikomputera 16-bitowego [BELL70].

PDP-11 wykorzystuje zestaw ośmiu 16-bitowych rejestrów roboczych. Dwa z nich mają dodatkowe znaczenie: jeden jest używany jako wskaźnik stosu dla specjalnych operacji na stosie, a drugi pełni rolę licznika programu, który zawiera adres następnego rozkazu.

Na rysunku 10.6 są pokazane formaty rozkazu PDP-11. Zastosowano 13 różnych formatów, obejmujących rozkazy O, 1 i 2-adresowe. Długość kodu operacyjnego może się wahać od 4 do 16 bitów. Odniesienia do rejestrów mają długość 6 bitów. Trzy bity identyfikują rejestr a pozostałe 3 określają tryb adresowania.



Pola źródła i miejsca przeznaczenia zawierają po 3 bity określające tryb adresowania i 3-bitowy numer rejestru

FP jest jednym z rejestrów zmiennopozycyjnych O, 1, 2 lub 3

R jest jednym z rejestrów roboczych

CC jest polem kodu warunkowego

RYСУNEK 10.6. Formaty rozkazu stosowane w PDP-11; liczby wskazują długość pól

wania. PDP-11 jest wyposażony w bogaty zestaw trybów adresowania. Zaletą łączenia trybu adresowania raczej z argumentem niż (jak to jest czasem czynione) z kodem operacji jest to, że dowolny tryb adresowania może być używany z dowolnym kodem operacji. Jak już wspomnieliśmy, ta niezależność została określona jako **ortogonalność**.

Rozkazy PDP-11 mają zwykle długość jednego słowa (16 bitów). W przypadku niektórych rozkazów jest dodany jeden lub dwa adresy pamięci, więc częścią repertuaru są rozkazy 32- i 48-bitowe. Zapewnia to dalsze zwiększenie elastyczności adresowania.

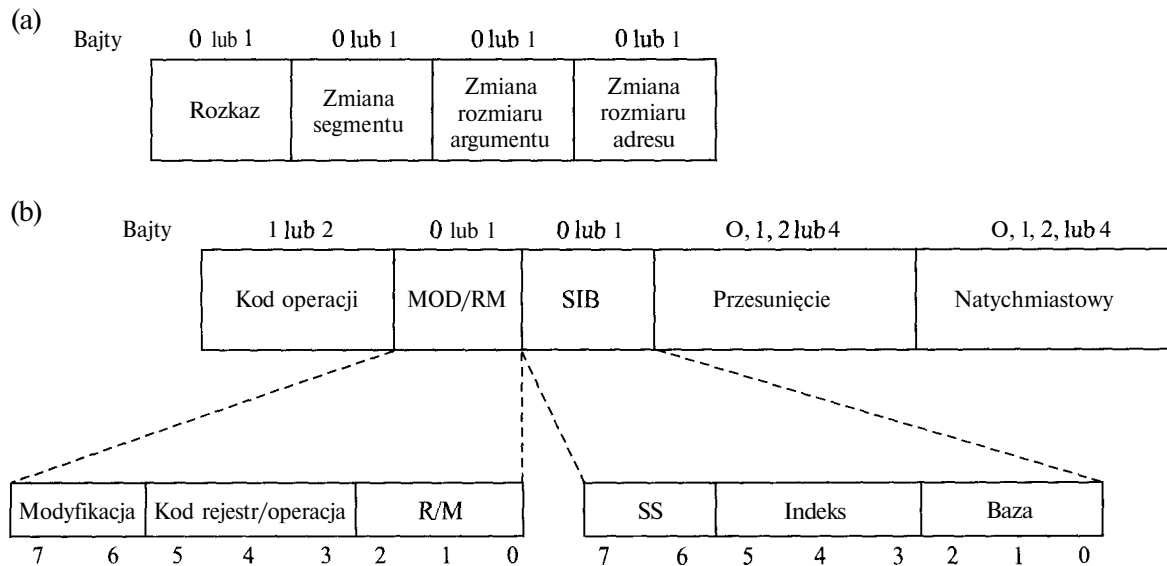
Lista rozkazów i możliwości adresowania PDP-11 są skomplikowane. Zwiększa to zarówno koszt sprzętu, jak i złożoność oprogramowania. Zaletą jest to, że mogą być opracowywane bardziej wydajne i zwarte programy.

Formaty rozkazów Pentium

Pentium został wyposażony w różnorodne formaty rozkazów. Spośród elementów opisanych poniżej, zawsze obecne jest tylko pole kodu operacji. Na rysunku 10.7 jest pokazany ogólny format rozkazu. Rozkazy są złożone z od 0 do 4 opcjonalnych przedrostków, z 1 lub 2-bajtowego kodu operacji, z opcjonalnego specyfikatora adresu składającego się z bajtu trybu r/m i bajtu Scalę Index, z opcjonalnego przesunięcia oraz z opcjonalnego pola natychmiastowego.

Rozpatrzmy najpierw bajty przedrostka:

- *Przedrostki rozkazu.* Przedrostek rozkazu, jeśli jest obecny, zawiera przedrostek LOCK lub jeden z **przedrostków powtarzania** (ang. *repeat prefixes*). Przedrostek LOCK jest używany w celu zapewnienia wyłącznego użycia pamięci wspólnej w środowisku wieloprocesorowym. Przedrostki powtarzania określają powtarzalną operację na łańcuchu, która pozwala Pentium przetwarzać łańcuchy znacznie szybciej niż za pomocą zwykłych pętli programowych. Istnieje pięć różnych przedrostków powtarzania: REP, REPE, REPZ, REPNE i REPNZ. Gdy jest obecny bezwzględny przedrostek REP, operacja określona w rozkazie jest prowadzona powtarzalnie na kolejnych elementach łańcucha; liczba powtórzeń jest określona w rejestrze CX. Przedrostek warunkowy REP powoduje powtarzanie operacji, aż stan CX wyniesie zero lub do czasu spełnienia określonego warunku.
- Q *Pominięcie segmentu.* Jawne ustalenie, którego rejestru segmentu powinien użyć rozkaz, niezależnie od wyboru dokonanego przez Pentium w odniesieniu do tego rozkazu.
- Q *Rozmiar adresu.* Procesor może adresować pamięć, stosując adresy 16- lub 32-bitowe. Rozmiar adresu określa rozmiar przemieszczenia w rozkazach i rozmiar adresu względnego generowanego przy obliczaniu adresu efektywnego. Jeden z tych rozmiarów jest oznaczany jako pomijany, a przedrostek rozmiaru adresu powoduje przeskok między generowaniem adresów 16- a 32-bitowych.



RYŚUNEK 10.7. Format rozkazu procesora Pentium: (a) przedrostek; (b) rozkaz

- **Rozmiar argumentu.** Podobnie jak wyżej, w rozkazie pomijana jest długość argumentu 16 lub 32 bity, a przedrostek argumentu powoduje przeskok między argumentami 32- a 16-bitowymi.

Sam rozkaz zawiera następujące pola:

- a **Kod operacji (opcode).** Jest on 1 lub 2-bajtowy. Kod operacji może także zawierać bity, które precyzują, czy dane są bajtowe, czy pełnowymiarowe (16 lub 32 bity zależnie od kontekstu), bity określające kierunek transferu danych (do lub z pamięci) oraz bity wskazujące czy pole danych natychmiastowych musi być poszerzone o znak.
- a **Tryb r/m (mod r/m).** Ten bajt oraz następny dostarczają informacji adresowej. Bajt „tryb r/m” określa, czy argument znajduje się w rejestrze, czy w pamięci. Jeśli znajduje się w pamięci, to pola wewnątrz bajta identyfikują przewidziany do użytku tryb adresowania. Bajt „tryb r/m” składa się z trzech pól: pole „tryb” (2 bity) łączy się z polem „r/m” tworząc 32 możliwe wartości: 8 rejestrów i 24 tryby adresowania; pole „rejestr/kod operacji” (3 bity) określa albo numer rejestru, albo daje 3 dodatkowe bity kodu operacji; pole „r/m” (3 bity) może wskazywać rejestr jako lokalizację argumentu lub może stanowić część kodu trybu adresowania w połączeniu z polem „tryb”.
- a **Skalowanie-indeks-podstawa (SIB).** Jeden z kodów bajta „tryb r/m” oznacza włączenie bajta SIB w celu pełnego określenia trybu adresowania. Bajt SIB składa się z trzech pól: pole SS (2 bity) identyfikuje czynnik skalowania przy skalowanym indeksowaniu, pole „indeks” (3 bity) - rejestr indeksowy, a pole Base (3 bity) - rejestr podstawowy.
- **Przesunięcie (ang. Displacement).** Gdy specyfikator trybu adresowania wskazuje, że jest używane przesunięcie, dodawane jest pole przesunięcia zawierające 8-, 16- lub 32-bitową liczbę całkowitą ze znakiem.
- a **Natychmiastowe (ang. immediate).** Zawiera wartość 8-, 16- lub 32-bitowego argumentu.

Pożyteczne może być teraz przeprowadzenie kilku porównań. W formacie Pentium tryb adresowania stanowi raczej część sekwencji kodu operacji niż argumentu. Ponieważ tylko jeden argument może nieść informację o trybie adresowania, w rozkazie może występować tylko jedno odniesienie do argumentu w pamięci. W przeciwieństwie do tego VAX przenosi informację o trybie adresowania z każdym argumentem, co pozwala na operacje pamięć-do-pamięci. Rozkazy Pentium są bardziej zwarte. Jeśli jednak jest wymagana operacja pamięć-do-pamięci, VAX może ją uruchomić za pomocą jednego rozkazu.

Format Pentium pozwala na użycie przy indeksowaniu adresów względnych nie ograniczających się do 1, 2 czy 4 bajtów. Chociaż zastosowanie długiego adresu względnego powoduje wydłużenie rozkazu, własność ta pozwala na uzyskanie potrzebnej elastyczności. Jest ona na przykład użyteczna przy adresowaniu dużych tablic lub ramek stosów. W przeciwieństwie do tego, format rozkazu

IBM S/370 umożliwia adresowanie lokalne nie więcej niż 4 KB (12-bitowy adres względny), a przesunięcie musi być dodatnie. Gdy lokacja znajduje się poza zasięgiem tego adresu względnego, kompilator musi generować dodatkowy kod w celu uzyskania potrzebnego adresu. Problem ten jest szczególnie widoczny podczas operowania ramkami stosu, które mają zmienne lokalne zajmujące ponad 4 KB. Jak stwierdzono w [DEWA90], „generowanie kodów dla 370 jest tak bolesne, że istniały nawet kompilatory dla tych maszyn, które po prostu ograniczały rozmiar ramki stosu do 4 KB”.

Jak można zauważyć, dekodowanie listy rozkazów Pentium jest bardzo skomplikowane. Wynikło to częściowo z potrzeby wstecznej kompatybilności z maszyną 8086, a częściowo z dążenia części projektantów do zapewnienia piszącym kompilatory wszelkiej możliwej pomocy w tworzeniu efektywnych kodów. Jest sprawą do dyskusji, czy tak skomplikowana lista rozkazów może stanowić lepszy wybór w stosunku do znajdującej się na przeciwnym biegunie listy rozkazów RISC.

Formaty rozkazu PowerPC

Wszystkie rozkazy procesora PowerPC mają długość 32 bity i odpowiadają pewnemu regularnemu formatowi. Pierwsze 6 bitów rozkazu określa operację, która ma być wykonana. W pewnych przypadkach w innej części rozkazu znajduje się rozszerzenie kodu operacji, które określa szczególny przypadek operacji. Na rysunku 10.8 bity kodu operacji są reprezentowane przez zacieniowane obszary każdego z formatów.

Zwróćmy uwagę na regularną strukturę formatów, która ułatwia pracę jednostkom wykonującym rozkazy. W przypadku wszystkich rozkazów arytmetycznych, logicznych oraz ładuj/zapisz po kodzie operacji następuje 25-bitowe odniesienie do rejestru, co pozwala na używanie 32 rejestrów roboczych.

Rozkazy rozgałęzienia zawierają bit powiązania (L), który wskazuje, że efektywny adres rozkazu następującego po rozkazie rozgałęzienia ma być umieszczony w rejestrze powiązania. Dwie postacie rozkazów zawierają także bit (A) wskazujący, czy tryb adresowania jest bezwzględny, czy zależny od PC (licznika programu). W przypadku warunkowych rozkazów rozgałęzienia pole bitowe CR określa bit, który ma być testowany w rejestrze warunku. Pole opcji określa warunki dokonania rozgałęzienia. Mogą być ustalone następujące warunki:

- a rozgałęziaj zawsze;
- a rozgałęziaj, gdy wynik obliczeń jest różny od 0 i warunek jest fałszywy (nie spełniony);
- a rozgałęziaj, gdy wynik obliczeń jest różny od 0 i warunek jest prawdziwy (spełniony);
- Q rozgałęziaj, gdy wynik obliczeń jest równy od 0 i warunek jest fałszywy;
- rozgałęziaj, gdy wynik obliczeń jest równy od 0 i warunek jest prawdziwy;
- a rozgałęziaj, gdy wynik obliczeń jest różny od 0;

do długich rozkazów. W przypadku Pentium wartość przesunięcia może wynosić do 32 bitów, co wymaga 6-bajtowego rozkazu. Adresowanie z przesunięciem może być użyteczne przy odnoszeniu się do zmiennych globalnych.

Pozostałe tryby adresowania są pośrednie w tym sensie, że adresowa część rozkazu mówi procesorowi, gdzie należy szukać adresu. Tryb podstawowy ustala, że jeden z rejestrów 8-, 16- lub 32-bitowych zawiera adres efektywny. Jest to równoważne temu, co określiliśmy jako pośrednie adresowanie rejestrowe.

W trybie **z rejestrem podstawowym i z przesunięciem** rozkaz zawiera przesunięcie, które jest dodawane do rejestru podstawowego. Rejestrem podstawowym może być dowolny z rejestrów roboczych. Oto przykłady stosowania tego trybu:

1. Jest używany przez kompilator do wyznaczania początku obszaru zmiennej lokalnej. Na przykład rejestr podstawowy może wskazać początek ramki stosu, która zawiera zmienne lokalne odpowiedniej procedury.
2. Jest używany do indeksowania tablicy, której elementy mają różny rozmiar (od 2, 4 lub 8 bajtów) i która wobec tego nie może być indeksowana za pomocą rejestru indeksowego. W tym przypadku przesunięcie wskazuje początek tablicy, a rejestr podstawowy zawiera wyniki obliczeń określające adres względny elementu wewnątrz tablicy.
3. Jest używany w celu uzyskiwania dostępu do pola rekordu. Rejestr podstawowy wskazuje początek rekordu, a przesunięcie jest adresem względnym pola.

W trybie **skalowanego indeksowania z przesunięciem** rozkaz zawiera przesunięcie, które jest dodawane do rejestru, w tym przypadku nazywanego rejestrem indeksowym. Rejestrem indeksowym może być dowolny z rejestrów roboczych z wyjątkiem ESP, który jest na ogół używany do przetwarzania stosowego. Obliczanie adresu efektywnego polega na pomnożeniu zawartości rejestru indeksowego przez czynnik 1, 2, 4 lub 8, a następnie na dodaniu wyniku do przesunięcia. Tryb ten jest bardzo wygodny do indeksowania tablic. Czynnik skalowania 2 może być użyty w przypadku tablic 16-bitowych liczb całkowitych, czynnik 4 - w przypadku 32-bitowych liczb całkowitych lub zmiennopozycyjnych. Wreszcie czynnik 8 może być użyty w przypadku tablic liczb zmiennopozycyjnych o podwójnej dokładności.

W trybie **indeksowym z rejestrem podstawowym i z przesunięciem** sumuje się zawartość rejestru podstawowego, rejestru indeksowego oraz przesunięcia w celu utworzenia adresu efektywnego. Znowu rejestrem podstawowym może być dowolny z rejestrów roboczych, a rejestrem indeksowym — dowolny z rejestrów roboczych z wyjątkiem ESP. Tryb ten może być na przykład używany do adresowania lokalnej tablicy w ramce stosu. Może być także używany do obsługi tablic dwuwymiarowych; w tym przypadku przesunięcie wskazuje początek tablicy, a każdy rejestr dotyczy jednego wymiaru tablicy.

W trybie z przesunięciem, ze skalowanym indeksowaniem i z rejestrem podstawowym sumuje się zawartość rejestru indeksowego, pomnożoną przez czynnik skalowania, z zawartością rejestru podstawowego i przesunięciem. Jest to użyteczne, gdy tablica jest przechowywana w ramce stosu; w tym przypadku długość elementów tablicy powinna wynosić po 2, 4 lub 8 bajtów. Tryb ten zapewnia również efektywne indeksowanie tablicy dwuwymiarowej, gdy elementy tablicy mają długość po 2, 4 lub 8 bajtów.

Wreszcie adresowanie względne może być używane w rozkazach przekazywania sterowania. Przesunięcie jest dodawane do wartości licznika programu, który wskazuje następny rozkaz. W tym przypadku przesunięcie jest traktowane jako bajt, słowo lub podwójne słowo ze znakiem, a jego wartość albo zwiększa, albo zmniejsza adres w liczniku programu.

Tryby adresowania PowerPC

Podobnie jak większość maszyn RISC oraz w przeciwieństwie do Pentium i większości maszyn CISC, procesor PowerPC używa stosunkowo prostego zestawu trybów adresowania. Jak widać w tabeli 10.3, tryby te są dogodnie poklasyfikowane ze względu na rodzaj rozkazu.

TABELA 10.3. Tryby adresowania procesora PowerPC

Tryb	Algorytm
	<i>Adresowanie ładowania/zapisu</i>
Pośredni	$EA = (BR) + D$
Pośredni indeksowany	$EA = (BR) + (IR)$
	<i>Adresowanie rozgałęzień</i>
Bezwzględny	$EA = I$
Względny	$EA = (PC) + I$
Pośredni	$EA = (L/CR)$
	<i>Obliczenia stałopozycyjne</i>
Rejestrowy	$EA = GPR$
Natychniastowy	$Argument = I$
	<i>Obliczenia zmiennopozycyjne</i>
Rejestrowy	$EA = FPR$

EA - adres efektywny

(X) - zawartość X

BR - rejestr podstawowy

IR - rejestr indeksowy

L/CR - rejestr powiązań lub zliczający

GPR - rejestr roboczy

FPR - rejestr zmiennopozycyjny

D - przesunięcie

I - wartość natychmiastowa

PC - licznik rozkazów

Adresowanie ładowania/zapisu

W procesorze PowerPC przewidziano dwa alternatywne tryby adresowania dla rozkazów ładowania/zapisu (rys. 10.3). W przypadku **adresowania pośredniego** rozkaz zawiera 16-bitowe przesunięcie dodawane do rejestru podstawowego, którym może być dowolny z rejestrów roboczych. Ponadto rozkaz może ustalać, że nowo obliczony adres efektywny ma być skierowany do rejestru podstawowego, aktualizując jego zawartość. Opcja aktualizacji jest użyteczna w przypadku progresywnego indeksowania tablic w pętłach.

Inną metodą adresowania w rozkazach ładowania/zapisu jest **pośrednie adresowanie indeksowane**. W tym przypadku rozkaz odnosi się do rejestru podstawowego i rejestru indeksowego, z których oba mogą być dowolnym rejestrem roboczym. Adres efektywny jest sumą zawartości tych dwóch rejestrów. Jak poprzednio opcja aktualizacji umożliwia wprowadzenie nowego adresu efektywnego do rejestru podstawowego.

Adresowanie rozgałęzienia

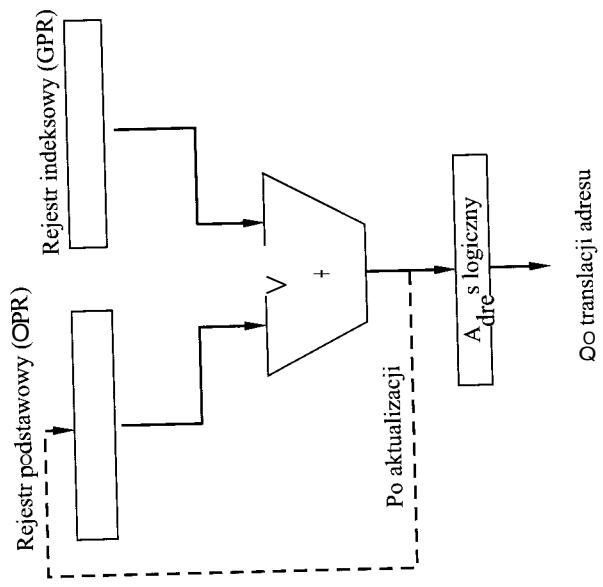
Przewidziane są trzy tryby adresowania rozgałęzienia. Gdy używane jest **adresowanie bezwzględne** w połączeniu z rozkazem rozgałęzienia bezwarunkowego, adres efektywny następnego rozkazu jest otrzymywany z 24-bitowej wartości natychmiastowej (ang. *immediate value*) zawartej w rozkazie. Wartość ta jest rozszerzana do 32 bitów przez dodanie dwóch zer po stronie najmniej znaczących bitów (co jest dopuszczalne, ponieważ wszystkie rozkazy muszą występować w 32-bitowych granicach) oraz przez poszerzenie znaku. W przypadku rozkazów rozgałęzienia warunkowego adres efektywny następnego rozkazu jest wyprowadzany z 16-bitowej wartości natychmiastowej zawartej w rozkazie. Wartość 16-bitowa jest rozszerzana do 32 bitów przez dodanie dwóch zer po stronie najmniej znaczących bitów oraz rozszerzenie znaku.

W przypadku **adresowania względnego** wartość natychmiastowa 24-bitowa (rozkazy rozgałęzienia bezwarunkowego) lub 14-bitowa (rozkazy rozgałęzienia warunkowego) jest rozszerzana jak poprzednio. Wartość wynikowa jest następnie dodawana do licznika programu w celu określenia lokacji w stosunku do bieżącego rozkazu. Innym trybem adresowania rozgałęzienia warunkowego jest **adresowanie pośrednie**. W trybie tym adres następnego rozkazu jest uzyskiwany albo z rejestru powiązania (ang. *link register*), albo z rejestru zliczającego (ang. *count register*). Zauważmy, że w tym przypadku do przechowywania adresu rozkazu rozgałęzienia jest używany rejestr zliczający. Rejestr ten może też być używany do zliczania w pętli, co wyjaśniliśmy wcześniej.

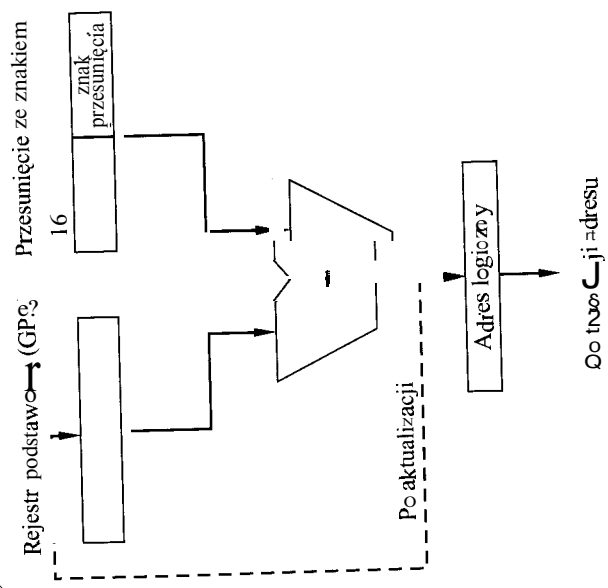
Rozkazy arytmetyczne

W przypadku arytmetyki liczb całkowitych wszystkie argumenty muszą być zawarte albo w rejestrach, albo w samych rozkazach. Przy **adresowaniu rejestrowym** argumenty źródłowe lub docelowe są określane jako zawartość jedne-

(b)



(a)



Rysunek 10.3. Tryby adresowania argumentów w procesorze PowerPC [6EF94a]:
(a) pośrednie; (b) bezpośrednie

go z rejestrów roboczych. Przy **adresowaniu natychmiastowym** argument źródłowy występuje w rozkazie jako 16-bitowa wielkość ze znakiem.

W przypadku arytmetyki zmiennopozycyjnej wszystkie argumenty są zawarte w rejestrach zmiennopozycyjnych; oznacza to, że jest stosowane tylko adresowanie rejestrowe.

10.2

FORMATY ROZKAZÓW

Format rozkazu określa rozkład bitów rozkazu odniesiony do jego części składowych. Format rozkazu musi zawierać kod operacji oraz, w sposób jawny lub domyślny, zero lub więcej argumentów. Każdy jawny argument jest adresowany za pomocą jednego z trybów opisanych w podrozdz.10.1. Format musi, w sposób jawny lub domyślny, wskazywać tryb adresowania każdego argumentu. W większości list rozkazów występuje więcej niż jeden format rozkazu.

Projektowanie formatu rozkazu jest sztuką złożoną i wdrożono dotychczas zdumiewającą różnorodność rozwiązań. W tym podrozdziale przeanalizujemy podstawowe zagadnienia projektowania, odwołując się krótko do pewnych przykładów, po czym szczegółowo przedstawimy rozwiązania przyjęte w procesorach Pentium i PowerPC.

Długość rozkazu

Najbardziej podstawowym problemem projektowym jest długość formatu rozkazu. Decyzja ta wpływa na rozmiar pamięci, organizację pamięci, strukturę magistrali, złożoność i szybkość procesora. Z kolei te czynniki mają wpływ na tę decyzję. Decyzja dotycząca długości rozkazu określa bogactwo i elastyczność maszyny, widziane przez programistę posługującego się językiem assemblerowym.

Najbardziej oczywisty jest kompromis między żądaniem rozbudowanego repertuaru rozkazów a potrzebą oszczędzania przestrzeni pamięci. Programiści domagają się więcej kodów operacji, więcej argumentów, więcej trybów adresowania i większego zakresu adresów. Więcej kodów operacji i więcej argumentów ułatwia życie programiście, ponieważ do osiągnięcia postawionego celu wystarczą krótsze programy. Podobnie więcej trybów adresowania daje programiście większą elastyczność we wdrażaniu pewnych funkcji, takich jak manipulowanie tablicami i wielodrożne rozgałęzienia. Oczywiście jest też, że ze wzrostem pojemności pamięci głównej i przy wzrastającym wykorzystaniu pamięci wirtualnych, programista chciałby móc adresować coraz większe zakresy pamięci. Wszystkie te rzeczy (kody operacji, argumenty, tryby adresowania, zakres adresów) wymagają bitów i popychają w kierunku dłuższych rozkazów. Jednak dłuższe rozkazy mogą prowadzić do rozrzutności. Rozkaz 32-bitowy zajmuje dwa razy tyle przestrzeni, co 16-bitowy, jednak prawdopodobnie jego użyteczność wzrasta znacznie mniej niż dwukrotnie.

Obok tego podstawowego kompromisu istnieją jeszcze inne. Albo długość rozkazu powinna być równa długości jednostki transferu z (i do) pamięci (w systemie magistralowym - szerokości magistrali danych), albo jedno powinno być wielokrotnością drugiego. W przeciwnym razie nie otrzymamy całkowitej liczby rozkazów podczas cyklu pobierania. Problemem z tym związanym jest szybkość transferu z (i do) pamięci. Szybkość ta nie nadążała za wzrostem szybkości procesora. Wobec tego pamięć może się stać wąskim gardłem, jeśli procesor może szybciej wykonywać rozkazy, niż je pobierać. Jednym z rozwiązań tego problemu jest użycie pamięci podręcznej (patrz podrozdz. 4.3); innym jest używanie krótszych rozkazów. Jak poprzednio, rozkazy 16-bitowe mogą być pobierane dwukrotnie szybciej niż 32-bitowe, jednak prawdopodobnie nie mogą być wykonywane dwa razy szybciej.

Ewidentnie powszechną, lecz mimo to ważną własnością jest to, że długość rozkazu powinna być wielokrotnością długości znaku, która zwykle wynosi 8 bitów, oraz długości liczb stałopozycyjnych. Aby to dostrzec, musimy posłużyć się tym tak nieszczęśliwie zdefiniowanym pojęciem, jakim jest „słowo” [FRA183]. Długość słowa pamięci jest w pewnym sensie „naturalną” jednostką organizacji. Rozmiar słowa zwykle determinuje rozmiar liczb stałopozycyjnych (zwykle te dwie jednostki są równe). Rozmiar słowa jest zwykle równy rozmiarowi jednostki transferu z/do pamięci albo przynajmniej jest z nim integralnie związany. Ponieważ powszechną formą danych są dane znakowe, chcielibyśmy dysponować słowem, w którym może być przechowywana całkowita liczba znaków. W przeciwnym razie przechowywanie wielu znaków pociągnie za sobą stratę bitów w każdym słowie albo znak będzie wykraczał poza granice słowa. Znaczenie tego problemu jest tak duże, że gdy IBM wprowadzał System/370 i chciał używać znaków 8-bitowych, podjął bolesną decyzję odejścia od 36-bitowej architektury naukowych komputerów z serii 700/7000 i przyjęcia architektury 32-bitowej.

Przydział bitów

Rozpatrzyliśmy niektóre z czynników wpływających na decyzje dotyczące długości formatu rozkazu. Równie trudnym problemem jest przydział bitów w tym formacie. Mamy tu do czynienia ze skomplikowanymi kompromisami.

Przy danej długości rozkazu istnieje oczywista wymiennność między liczbą kodów operacji a zdolnością adresowania. Więcej kodów operacji oznacza oczywiście więcej bitów w polu kodu operacji. W przypadku formatu rozkazu o danej długości następuje zredukowanie liczby bitów dostępnych na potrzeby adresowania. Istnieje interesujące ulepszenie dotyczące tego problemu: jest nim użycie kodów operacji o zmiennej długości. W tym rozwiązaniu występuje minimalna długość kodu operacji, jednak w przypadku niektórych kodów mogą być specyfikowane dodatkowe operacje za pomocą dodatkowych bitów w rozkazie. Przy rozkazach o ustalonej długości pozostawia to mniej bitów na adre-

sowanie. Własność ta jest wobec tego używana w odniesieniu do tych rozkazów, które wymagają mniej argumentów i (lub) mniej rozbudowanego adresowania. Zapoznamy się z przykładami tej strategii.

Następujące wzajemnie powiązane czynniki muszą być uwzględniane przy decydowaniu o użyciu bitów na potrzeby adresowania:

- a *Liczba trybów adresowania.* Czasem tryb adresowania może być domyślny. Na przykład pewne kody operacji mogą zawsze wymagać indeksowania. W innych przypadkach tryby adresowania muszą być jawne, do czego jest potrzebny jeden lub więcej bitów.
- a *Liczba argumentów.* Widzieliśmy, że zmniejszona liczba adresów może prowadzić do dłuższych, bardziej kłopotliwych programów (np. rys. 9.3). Typowe rozkazy współczesnych maszyn przewidują dwa argumenty. Każdy adres argumentu w rozkazie może wymagać własnego wskaźnika trybu lub też wskaźnik trybu może być związany tylko z jednym polem adresowym.
- a *Rejestry a pamięć.* Maszyna musi mieć rejestry, dzięki którym dane mogą być wprowadzane do procesora w celu przetwarzania. W przypadku rejestru pojedynczego, widzialnego dla użytkownika (nazywanego zwykle akumulatorem), adres jednego argumentu jest domyślny i nie pochłania bitów rozkazu. Jednak programowanie jednorejestrowe jest kłopotliwe i wymaga wielu rozkazów. Nawet w przypadku wielu rejestrów potrzeba tylko kilku bitów do określenia rejestru. Im więcej rejestrów można przeznaczyć na odniesienia do argumentów, tym mniej potrzeba bitów. W wielu badaniach wykazano, że pożądana liczba rejestrów widzialnych dla użytkownika wynosi od 8 do 32 [LUND77], [HUCK83].
- Q *Liczba zestawów rejestrów.* Wiele maszyn ma jeden zestaw rejestrów roboczych, zawierający zwykle 8 do 16 rejestrów. Mogą one być używane do przechowywania danych albo też do przechowywania adresów w przypadku adresowania z przesunięciem. Współcześnie występuje tendencja do odchodzenia od jednego zestawu rejestrów roboczych w kierunku dwóch lub większej liczby zestawów wyspecjalizowanych (np. dane i przesunięcia). Tendencja ta występuje wszędzie, od mikroprocesorów jednostrukturalnych do superkomputerów. Zaletą tego rozwiązania jest to, że przy ustalonej liczbie rejestrów ich podział funkcjonalny oznacza mniej bitów potrzebnych w rozkazie. Na przykład przy dwóch zestawach po 8 rejestrów do identyfikowania rejestru potrzebne są tylko 3 bity; kod operacyjny domyślnie określa, do którego zestawu rejestrów następuje odniesienie. Wygląda na to, że rozwiązanie to ma niewiele wad [LUND77]. W systemach takich jak S/370, które mają jeden zestaw rejestrów roboczych, programiści zwykle ustanawiają konwencję, przypisującą około połowy rejestrów danym a pozostałe przesunięciom i pozostawiają to przypisanie na stałe [MALL79].
- *Zakres adresów.* W przypadku adresów odnoszących się do pamięci zakres możliwych adresów jest powiązany z liczbą bitów adresowych. Ponieważ stanowi to poważne ograniczenie, bezpośrednie adresowanie jest rzadko

stosowane. Dzięki adresowaniu z przesunięciem zakres adresów ulega poszerzeniu aż do wielkości wynikającej z długości rejestru adresowego. Nawet w tym przypadku nadal wygodne jest stosowanie raczej dużych przesunięć w stosunku do adresu rejestrowego, co wymaga stosunkowo dużej liczby bitów adresowych w rozkazie.

- *Stopień granulacji* (ang. *granularity*) *adresu*. W przypadku adresów, które odnoszą się raczej do pamięci niż do rejestrów, dodatkowym czynnikiem jest stopień granulacji adresu. W systemie o słowach 16- lub 32-bitowych adres może odnosić się do słowa lub bajta, zależnie od wyboru projektanta. Adresowanie bajtowe jest wygodne do manipulowania znakami, jednak przy ustalonym rozmiarze pamięci wymaga więcej bitów adresowych.

Wobec tego projektant ma do czynienia z całym zastępem czynników, które musi przeanalizować i zrównoważyć. Nie jest jasne, jak krytyczne są różne wybory. Zacytujemy na przykład jedno z badań [CRAG79], w którym porównano różne rozwiązania formatu rozkazu, włącznie z użyciem stosu, rejestrów roboczych, akumulatora i rozkazów typu pamięć-rejestr. Stosując spójny zespół założeń, nie stwierdzono znaczących różnic przestrzeni kodowej lub czasu realizacji.

Zapoznajmy się krótko z równoważeniem tych różnych czynników, które zastosowano w dwóch projektach maszyn.

PDP-8

Jednym z najprostszych projektów rozkazów dla komputera uniwersalnego było rozwiązanie przyjęte dla PDP-8 [BELL78b]. Komputer PDP8 używa rozkazów 12-bitowych i operuje na słowach 12-bitowych. Występuje w nim jeden rejestr roboczy - akumulator.

Mimo ograniczeń tego projektu, adresowanie jest całkiem elastyczne. Każde odniesienie do pamięci obejmuje 7 bitów plus 2 modyfikatory 1-bitowe. Pamięć jest podzielona na strony o ustalonej długości, po $2^7 = 128$ słów każda. Obliczanie adresu opiera się na odniesieniu do strony O lub strony bieżącej (zawierającej dany rozkaz), zależnie od bitu strony. Drugi modyfikator 1-bitowy wskazuje, czy stosowane jest adresowanie bezpośrednie, czy pośrednie. Oba tryby mogą być używane w połączeniu, tak że adres pośredni jest 12-bitowym adresem zawartym w słowie strony O lub strony bieżącej. Ponadto, 8 dedykowanych słów na stronie O stanowi automatycznie indeksowane „rejestry”. Gdy ma miejsce pośrednie odniesienie do tych lokacji, następuje wstępne indeksowanie.

Na rysunku 10.4 jest pokazany format rozkazu PDP-8. Występuje tu 3-bitowy kod operacji i trzy rodzaje rozkazów. Przy kodach od 0 do 5 formatem jest rozkaz z 1-adresowym odniesieniem do pamięci, zawierający bit strony i bit pośredniości. Wobec tego występuje tylko 6 operacji podstawowych. W celu poszerzenia grupy operacji kod operacji 7 określa odniesienie do rejestru lub **mikrorozkaz**. W tym formacie pozostałe bity służą do kodowania dodatkowych

Rozkazy z odniesieniem do pamięci

Kod operacji	D/I	Z/C	Przemieszczenie
0	2	3	4 5 11

Rozkazy wejścia/wyjścia

1 1 0	Urządzenie	Kod operacji
0	2 3	8 9 11

Rozkazy z odniesieniem do rejestrów

Mikrorozkazy grupy 1

1	1	1	0	CLA	CLL	CMA	CML	RAR	RAL	BSW	IAC
0	1	2	3	4	5	6	7	8	9	10	11

Mikrorozkazy grupy 2

1	1	1	1	CLA	SMA	SZA	SNL	RSS	OSR	HLT	0
0	1	2	3	4	5	6	7	8	9	10	11

Mikrorozkazy grupy 3

1	1	1	1	CLA	MQA	0	MQL	0	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	

Mnemoniki

CLA - wyczyść akumulator

CLL - wyczyść łącze

CMA - dopełnij akumulator

CML - dopełnij łącze

RAR - wykonaj rotację akumulatora w prawo

RAL - wykonaj rotację akumulatora w lewo

BSW - wymień bajt

IAC - inkrementuj akumulator

SMA - przeskoż na minus akumulator

SZA - przeskoż na zero akumulator

SNL - przeskoż na łącze niezerowe

RSS - odwróć odczyt przeskożu

OSR - wykonaj OR z rejestrem przełączania

HLT - zatrzymaj

MQA - mnożnik/iloraz do akumulatora

MQL - ładuj mnożnik/iloraz

RYSUNEK 10.4. Formaty rozkazu PDP-8

operacji. Na ogół każdy bit określa specyficzną operację (np. wyczyszczenie akumulatora), a bity te mogą być łączone w jednym rozkazie. Strategia mikroprogramowania była stosowana przez DEC jeszcze w PDP-1 i jest w pewnym sensie zapowiedzią dzisiejszych maszyn mikroprogramowanych, które będą przedyskutowane w części IV. Kod operacji 6 dotyczy wejścia-wyjścia; 6 bitów używa się do wyboru jednego z 64 urządzeń, a 3 bity określają konkretny rozkaz wejścia-wyjścia.

Efektywność formatu rozkazu PDP-8 jest godna uwagi. Umożliwia on adresowanie pośrednie, adresowanie z przesunięciem i indeksowanie. Wraz z rozszerzeniem kodu operacji obsługuje on łącznie około 35 rozkazów. Biorąc pod uwagę ograniczenia wynikające z 12-bitowego rozkazu, projektanci nie mogli spisać się lepiej.

PDP-10

Z listą rozkazów PDP-8 silnie kontrastuje lista PDP-10. PDP-10 został zaprojektowany jako duży system pracujący z podziałem czasu, z akcentem na łatwość programowania, nawet kosztem dodatkowych wydatków sprzętowych.

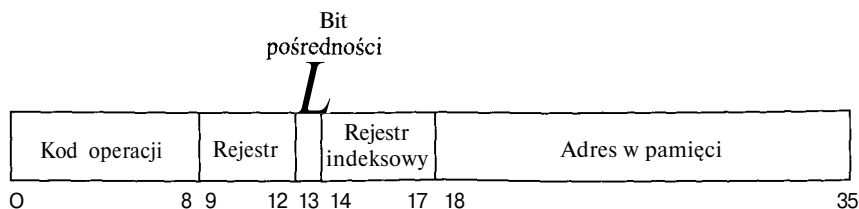
Do zasad wykorzystanych do zaprojektowania listy rozkazów należały [BELL78c]:

- a *Ortogonalność*. Ortogonalność jest zasadą, zgodnie z którą dwie zmienne są od siebie niezależne. W kontekście listy rozkazów termin ten oznacza, że inne elementy rozkazu są niezależne od kodu operacji (nie są przez ten kod determinowane). Projektanci PDP-10 użyli tego terminu w celu podkreślenia faktu, że adres jest zawsze obliczany w ten sam sposób, niezależnie od kodu operacji. Stanowi to kontrast w stosunku do wielu maszyn, w których tryb adresowania wynika czasem domyślnie z użytego operatora.
- a *Kompletność*. Każdy rodzaj danych arytmetycznych (całkowite, stałopozytywne, rzeczywiste) powinien mieć kompletną i identyczną listę operacji.
- Q *Adresowanie bezpośrednie*. Zrezygnowano z adresowania przesunięciowego z rejestrem podstawowym, obciążającego programistę problemami organizacji pamięci, na rzecz adresowania bezpośredniego.

Głównym celem każdej z tych zasad było ułatwienie programowania.

PDP-10 ma słowo 36-bitowe oraz 36-bitowy rozkaz. Ustalony format rozkazu widać na rys. 10.5. Kod operacji zajmuje 9 bitów, co pozwala na 512 operacji. W rzeczywistości zdefiniowano 365 różnych rozkazów. Większość rozkazów ma dwa adresy, z których jeden znajduje się w jednym z 16-bitowych rejestrów roboczych. Wobec tego odniesienie do tego argumentu zajmuje 4 bity. Odniesienie do drugiego argumentu rozpoczyna się od 18-bitowego pola adresu pamięci. Może być ono używane jako argument natychmiastowy lub jako adres pamięci. W tym ostatnim przypadku dozwolone jest indeksowanie oraz adresowanie pośrednie. Te same rejestry robocze są używane jako rejestry indeksowe.

Rozkaz długości 36-bitów jest rzeczywistym luksusem. Nie ma potrzeb; sprytnych rozwiązań zwiększających liczbę kodów operacji; 9-bitowe pole kodu jest więcej niż wystarczające. Adresowanie jest także proste. Atrakcyjne jest adresowanie bezpośrednie ze względu na 18-bitowe pole adresowe. Dla pamięci większych niż 2^{18} przewidziano adresowanie pośrednie. Aby ułatwić progra-



mowanie, istnieje możliwość indeksowania przy manipulowaniu tablicami i wykonywaniu programów iteracyjnych. Ponieważ pole argumentu ma 18 bitów, staje się atrakcyjne adresowanie natychmiastowe.

Projekt listy rozkazów PDP-10 osiągnął zapowiedziane cele [LUND77]. Liść jest stosunkowo łatwa z punktu widzenia programisty, kosztem nieefektywnego wykorzystania przestrzeni. Był to świadomy wybór dokonany przez projektantów i dlatego nie może być dyskwalifikowany jako słaby projekt.

Rozkazy o zmiennej długości

W dotychczasowych przykładach występowała jedna, ustalona długość rozkazów; przedyskutowane kompromisy dotyczyły tego właśnie kontekstu. Jednak projektanci mogą wybrać różne formaty rozkazów o różnych długościach. Taka taktyka ułatwia uzyskanie dużego repertuaru kodów operacji, o różnych długościach kodów. Adresowanie może być bardziej elastyczne przy różnych kombinacjach odniesień do rejestrów i do pamięci oraz przy różnych trybach adresowania. Rozkazy o zmiennej długości ułatwiają efektywne i zwarte wprowadzanie tych odmian.

Ceną, jaką płacimy za przyjęcie rozkazów o zmiennej długości jest wzrost złożoności procesora. Malejące ceny sprzętu, zastosowanie mikroprogramowania (przedyskutowane w części IV) oraz ogólnie lepsze rozumienie zasad projektowania procesorów przyczyniły się do tego, że tę niewielką cenę warto zapłacić.

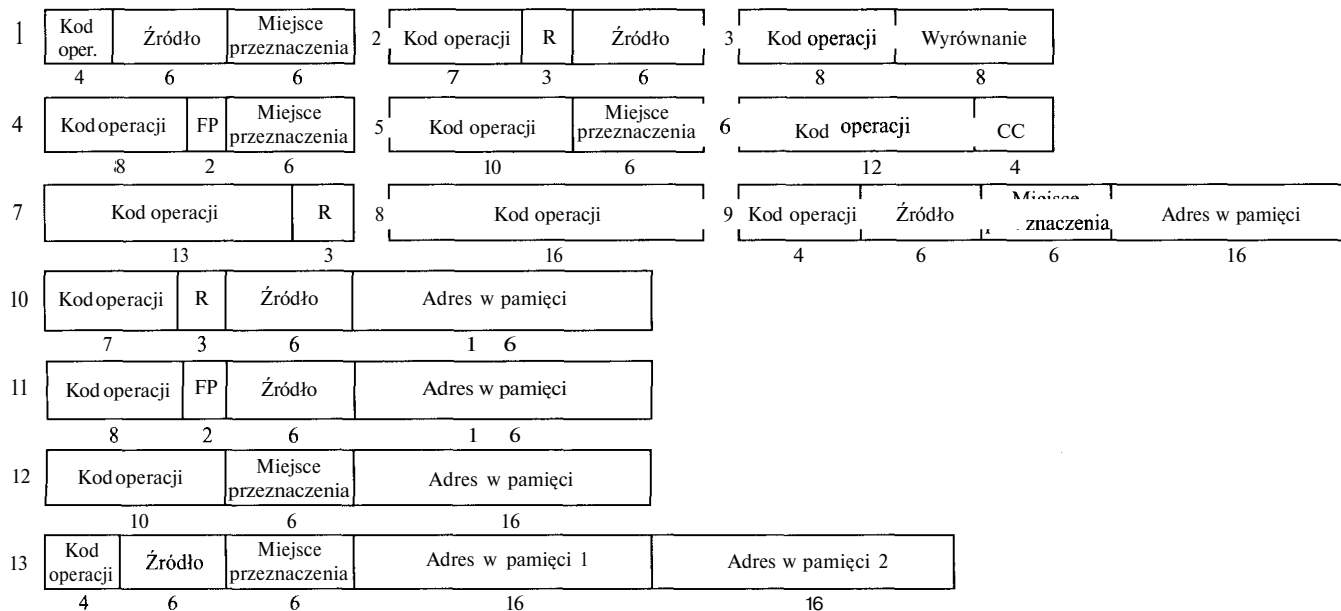
Zastosowanie rozkazów o zmiennej długości nie eliminuje żądania, aby wszystkie długości rozkazów były integralnie związane z długością słowa. Ponieważ procesor nie zna długości następnego rozkazu przewidzianego do pobrania, typową strategią jest pobieranie liczby bajtów lub słów równej przynajmniej najdłuższemu możliwemu rozkazowi. Oznacza to, że czasem pobiera się wiele rozkazów. Jak jednak zobaczymy w rozdz. 11, jest to w dowolnym przypadku strategia właściwa.

PDP-11

PDP-11 został zaprojektowany w celu zapewnienia skutecznej i elastycznej listy rozkazów w ramach ograniczeń minikomputera 16-bitowego [BELL70].

PDP-11 wykorzystuje zestaw ośmiu 16-bitowych rejestrów roboczych. Dwa z nich mają dodatkowe znaczenie: jeden jest używany jako wskaźnik stosu dla specjalnych operacji na stosie, a drugi pełni rolę licznika programu, który zawiera adres następnego rozkazu.

Na rysunku 10.6 są pokazane formaty rozkazu PDP-11. Zastosowano 13 różnych formatów, obejmujących rozkazy O, 1 i 2-adresowe. Długość kodu operacyjnego może się wahać od 4 do 16 bitów. Odniesienia do rejestrów mają długość 6 bitów. Trzy bity identyfikują rejestr a pozostałe 3 określają tryb adresowania.



Pola źródła i miejsca przeznaczenia zawierają po 3 bity określające tryb adresowania i 3-bitowy numer rejestru

FP jest jednym z rejestrów zmiennopozycyjnych 0, 1, 2 lub 3

R jest jednym z rejestrów roboczych

CC jest polem kodu warunkowego

RYSUNEK 10.6. Formaty rozkazu stosowane w PDP-11; liczby wskazują długość pól

wania. PDP-11 jest wyposażony w bogaty zestaw trybów adresowania. Zaletą łączenia trybu adresowania raczej z argumentem niż (jak to jest czasem czynione) z kodem operacji jest to, że dowolny tryb adresowania może być używany z dowolnym kodem operacji. Jak już wspomnieliśmy, ta niezależność została określona jako **ortogonalność**.

Rozkazy PDP-11 mają zwykle długość jednego słowa (16 bitów). W przypadku niektórych rozkazów jest dodany jeden lub dwa adresy pamięci, więc częścią repertuaru są rozkazy 32- i 48-bitowe. Zapewnia to dalsze zwiększenie elastyczności adresowania.

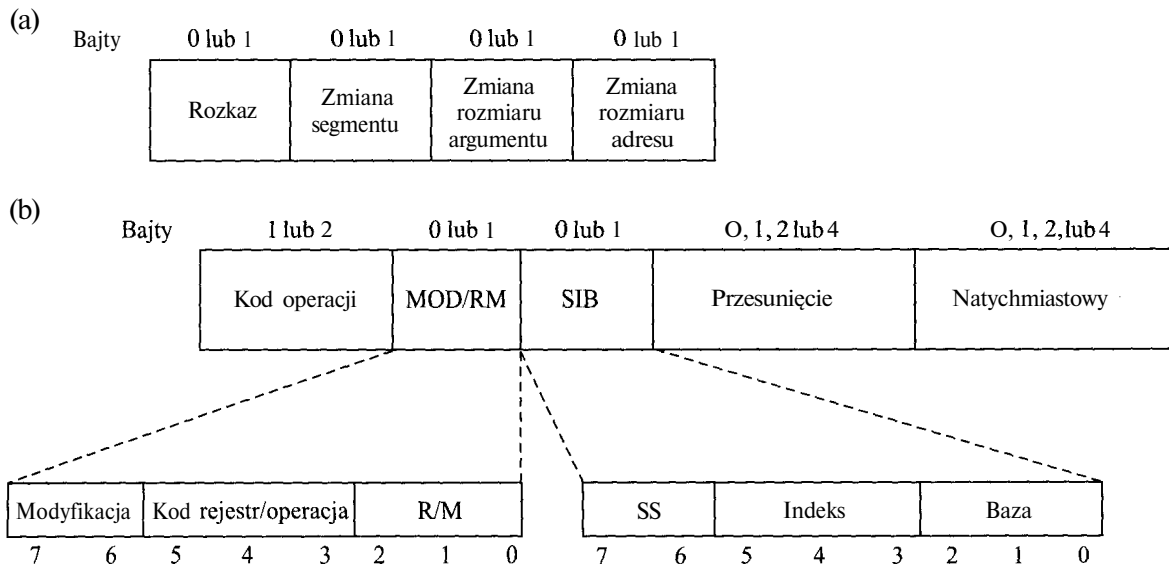
Lista rozkazów i możliwości adresowania PDP-11 są skomplikowane. Zwiększa to zarówno koszt sprzętu, jak i złożoność oprogramowania. Zaletą jest to, że mogą być opracowywane bardziej wydajne i zwarte programy.

Formaty rozkazów Pentium

Pentium został wyposażony w różnorodne formaty rozkazów. Spośród elementów opisanych poniżej, zawsze obecne jest tylko pole kodu operacji. Na rysunku 10.7 jest pokazany ogólny format rozkazu. Rozkazy są złożone z od 0 do 4 opcjonalnych przedrostków, z 1 lub 2-bajtowego kodu operacji, z opcjonalnego specyfikatora adresu składającego się z bajtu trybu r/m i bajtu Scale Index, z opcjonalnego przesunięcia oraz z opcjonalnego pola natychmiastowego.

Rozpatrzmy najpierw bajty przedrostka:

- ❑ *Przedrostki rozkazu.* Przedrostek rozkazu, jeśli jest obecny, zawiera przedrostek LOCK lub jeden z **przedrostków powtarzania** (ang. *repeat prefixes*). Przedrostek LOCK jest używany w celu zapewnienia wyłącznego użycia pamięci wspólnej w środowisku wieloprocesorowym. Przedrostki powtarzania określają powtarzalną operację na łańcuchu, która pozwala Pentium przetwarzać łańcuchy znacznie szybciej niż za pomocą zwykłych pętli programowych. Istnieje pięć różnych przedrostków powtarzania: REP, REPE, REPZ, REPNE i REPNZ. Gdy jest obecny bezwzględny przedrostek REP, operacja określona w rozkazie jest prowadzona powtarzalnie na kolejnych elementach łańcucha; liczba powtórzeń jest określona w rejestrze CX. Przedrostek warunkowy REP powoduje powtarzanie operacji, aż stan CX wyniesie zero lub do czasu spełnienia określonego warunku.
- a *Pominięcie segmentu.* Jawne ustalenie, którego rejestru segmentu powinien użyć rozkaz, niezależnie od wyboru dokonanego przez Pentium w odniesieniu do tego rozkazu.
- a *Rozmiar adresu.* Procesor może adresować pamięć, stosując adresy 16- lub 32-bitowe. Rozmiar adresu określa rozmiar przemieszczenia w rozkazach i rozmiar adresu względnego generowanego przy obliczaniu adresu efektywnego. Jeden z tych rozmiarów jest oznaczany jako pomijany, a przedrostek rozmiaru adresu powoduje przeskok między generowaniem adresów 16- a 32-bitowych.



RYСУNEK 10.7. Format rozkazu procesora Pentium: (a) przedrostek; (b) rozkaz

- **Rozmiar argumentu.** Podobnie jak wyżej, w rozkazie pomijana jest długość argumentu 16 lub 32 bity, a przedrostek argumentu powoduje przeskok między argumentami 32- a 16-bitowymi.

Sam rozkaz zawiera następujące pola:

- **Kod operacji (opcode).** Jest on 1 lub 2-bajtowy. Kod operacji może także zawierać bity, które precyzują, czy dane są bajtowe, czy pełnowymiarowe (16 lub 32 bity zależnie od kontekstu), bity określające kierunek transferu danych (do lub z pamięci) oraz bity wskazujące czy pole danych natychmiastowych musi być poszerzone o znak.
- **Tryb r/m (mod r/m).** Ten bajt oraz następny dostarczają informacji adresowej. Bajt „tryb r/m” określa, czy argument znajduje się w rejestrze, czy w pamięci. Jeśli znajduje się w pamięci, to pola wewnątrz bajta identyfikują przewidziany do użytku tryb adresowania. Bajt „tryb r/m” składa się z trzech pól: pole „tryb” (2 bity) łączy się z polem „r/m” tworząc 32 możliwe wartości: 8 rejestrów i 24 tryby adresowania; pole „rejestr/kod operacji” (3 bity) określa albo numer rejestru, albo daje 3 dodatkowe bity kodu operacji; pole „r/m” (3 bity) może wskazywać rejestr jako lokalizację argumentu lub może stanowić część kodu trybu adresowania w połączeniu z polem „tryb”.
- **Skalowanie-indeks-podstawa (SIB).** Jeden z kodów bajta „tryb r/m” oznacza włączenie bajta SIB w celu pełnego określenia trybu adresowania. Bajt SIB składa się z trzech pól: pole SS (2 bity) identyfikuje czynnik skalowania przy skalowanym indeksowaniu, pole „indeks” (3 bity) - rejestr indeksowy, a pole Base (3 bity) - rejestr podstawowy.
- a **Przesunięcie** (ang. *Displacement*). Gdy specyfikator trybu adresowania wskazuje, że jest używane przesunięcie, dodawane jest pole przesunięcia zawierające 8-, 16- lub 32-bitową liczbę całkowitą ze znakiem.
- a **Natychmiastowe** (ang. *immediate*). Zawiera wartość 8-, 16- lub 32-bitowego argumentu.

Pożyteczne może być teraz przeprowadzenie kilku porównań. W formacie Pentium tryb adresowania stanowi raczej część sekwencji kodu operacji niż argumentu. Ponieważ tylko jeden argument może nieść informację o trybie adresowania, w rozkazie może występować tylko jedno odniesienie do argumentu w pamięci. W przeciwieństwie do tego VAX przenosi informację o trybie adresowania z każdym argumentem, co pozwala na operacje pamięć-do-pamięci. Rozkazy Pentium są bardziej zwarte. Jeśli jednak jest wymagana operacja pamięć-do-pamięci, VAX może ją uruchomić za pomocą jednego rozkazu.

Format Pentium pozwala na użycie przy indeksowaniu adresów względnych nie ograniczających się do 1, 2 czy 4 bajtów. Chociaż zastosowanie długiego adresu względnego powoduje wydłużenie rozkazu, własność ta pozwala na użycie potrzebnej elastyczności. Jest ona na przykład użyteczna przy adresowaniu dużych tablic lub ramek stosów. W przeciwieństwie do tego, format rozkazu

IBM S/370 umożliwia adresowanie lokalne nie więcej niż 4 KB (12-bitowy adres względny), a przesunięcie musi być dodatnie. Gdy lokacja znajduje się poza zasięgiem tego adresu względnego, kompilator musi generować dodatkowy kod w celu uzyskania potrzebnego adresu. Problem ten jest szczególnie widoczny podczas operowania ramkami stosu, które mają zmienne lokalne zajmujące ponad 4 KB. Jak stwierdzono w [DEWA90], „generowanie kodów dla 370 jest tak bolesne, że istniały nawet kompilatory dla tych maszyn, które po prostu ograniczały rozmiar ramki stosu do 4 KB”.

Jak można zauważyć, dekodowanie listy rozkazów Pentium jest bardzo skomplikowane. Wynikło to częściowo z potrzeby wstecznej kompatybilności z maszyną 8086, a częściowo z dążenia części projektantów do zapewnienia piszącym kompilatory wszelkiej możliwej pomocy w tworzeniu efektywnych kodów. Jest sprawą do dyskusji, czy tak skomplikowana lista rozkazów może stanowić lepszy wybór w stosunku do znajdującej się na przeciwnym biegunie listy rozkazów RISC.

Formaty rozkazu PowerPC

Wszystkie rozkazy procesora PowerPC mają długość 32 bity i odpowiadają pewnemu regularnemu formatowi. Pierwsze 6 bitów rozkazu określa operację, która ma być wykonana. W pewnych przypadkach w innej części rozkazu znajduje się rozszerzenie kodu operacji, które określa szczególnie przypadek operacji. Na rysunku 10.8 bity kodu operacji są reprezentowane przez zacieniowane obszary każdego z formatów.

Zwróćmy uwagę na regularną strukturę formatów, która ułatwia pracę jednostkom wykonującym rozkazy. W przypadku wszystkich rozkazów arytmetycznych, logicznych oraz ładuj/zapisz po kodzie operacji następuje 25-bitowe odniesienie do rejestru, co pozwala na używanie 32 rejestrów roboczych.

Rozkazy rozgałęzienia zawierają bit powiązania (L), który wskazuje, że efektywny adres rozkazu następującego po rozkazie rozgałęzienia ma być umieszczony w rejestrze powiązania. Dwie postacie rozkazów zawierają także bit (A) wskazujący, czy tryb adresowania jest bezwzględny, czy zależny od PC (licznika programu). W przypadku warunkowych rozkazów rozgałęzienia pole bitowe CR określa bit, który ma być testowany w rejestrze warunku. Pole opcji określa warunki dokonania rozgałęzienia. Mogą być ustalone następujące warunki:

- Q rozgałęziaj zawsze;
- a rozgałęziaj, gdy wynik obliczeń jest różny od O i warunek jest fałszywy (nie spełniony);
- a rozgałęziaj, gdy wynik obliczeń jest różny od O i warunek jest prawdziwy (spełniony);
- rozgałęziaj, gdy wynik obliczeń jest równy od O i warunek jest fałszywy;
- a rozgałęziaj, gdy wynik obliczeń jest równy od O i warunek jest prawdziwy;
- Q rozgałęziaj, gdy wynik obliczeń jest różny od 0;

- rozgałęziaj, gdy wynik obliczeń jest równy 0;
- rozgałęziaj, gdy warunek jest fałszywy;
- rozgałęziaj, gdy warunek jest prawdziwy.

Większość rozkazów, których wynikiem są obliczenia (arytmetyczne, arytmetyczne zmiennopozycyjne, logiczne), zawiera bit wskazujący, czy wynik operacji powinien być zapisany w rejestrze warunku. Jak pokażemy, własność ta jest użyteczna do przetwarzania z przewidywaniem rozgałęzienia.

Rozkazy zmiennopozycyjne mają pola dla trzech rejestrów źródłowych. W wielu przypadkach są używane tylko dwa rejestry źródłowe. Kilka rozkazów obejmuje mnożenie dwóch rejestrów źródłowych, a następnie dodanie lub odjęcie trzeciego rejestru źródłowego. Włączono te złożone rozkazy, kierując się częstotliwością ich stosowania. Na przykład iloczyn wewnętrzny będący częścią wielu operacji macierzowych może być zaimplementowany za pomocą mnożenia-dodawania.

10.3

POLECANA LITERATURA

Pozycje literatury wymienione w rozdziale 9 są równie przydatne do tego rozdziału. Ponadto można sięgnąć do [FLYN85], gdzie znajduje się analiza problemów projektowania list rozkazów, szczególnie dotyczących formatu rozkazu.

FLYN85 Flynn M., Johnson J., Wakefield S.: „On instruction Sets and Their Formats”. *IEEE Trans. on Computers*, March 1985.

10.4

PROBLEMY DO ROZWIĄZANIA

- 10.1. Uzasadnij twierdzenie, że prawdopodobnie rozkaz 32-bitowy jest o wiele mniej niż 2-krotnie bardziej użyteczny niż rozkaz 16-bitowy.
- 10.2. Dane są następujące wartości pamięci i maszyna 1-adresowa z akumulatorem. Jakie wartości zostaną załadowane do akumulatora na podstawie następujących rozkazów?
 - a słowo 20 zawiera 40;
 - a słowo 30 zawiera 50;
 - a słowo 40 zawiera 60;
 - a słowo 50 zawiera 70.
 - (a) LOAD IMMEDIATE 20 (ładuj natychmiastowo 20)
 - (b) LOAD DIRECT 20 (ładuj bezpośrednio 20)
 - (c) LOAD INDIRECT 20 (ładuj pośrednio 20)
 - (d) LOAD IMMEDIATE 30 (ładuj natychmiastowo 30)

- (e) LOAD DIRECT 30 (ładuj bezpośrednio 30)
 - (f) LOAD INDIRECT 30 (ładuj pośrednio 30)*
- 10.3. Niech adres przechowywany w liczniku programu będzie oznaczony symbolem $X1$. Rozkaz przechowywany w $X1$ ma część adresową (odniesienie do argumentu) $X2$. Argument potrzebny do wykonania rozkazu jest przechowywany w słowie pamięci o adresie $X3$. Rejestr indeksu zawiera wartość $X4$. Jakie są wzajemne związki między tymi wielkościami, jeśli tryb adresowania jest:
- (a) bezpośredni,
 - (b) pośredni,
 - (c) względny wobec licznika PC,
 - (d) indeksowany?
- 10.4. Rozkaz rozgałęzienia sformułowany w trybie adresowania względnego wobec licznika PC jest przechowywany w pamięci pod adresem 620_{10} . Skok jest dokonywany do lokacji 530_{10} . Pole adresowe rozkazu ma długość 10 bitów. Jaka wartość binarna jest w rozkazie?
- 10.4. Ile razy procesor musi się odwoływać do pamięci, gdy pobiera i wykonuje rozkaz w trybie adresowania pośredniego, jeśli rozkaz jest:
- (a) obliczeniem wymagającym jednego argumentu,
 - (b) rozgałęzieniem?
- 10.6. IBM 370 nie przewiduje adresowania pośredniego. Załóżmy, że adres argumentu znajduje się w pamięci głównej. Jak możesz uzyskać dostęp do tego argumentu?
- 10.7. Dlaczego decyzja IBM przejścia od słowa 36-bitowego do 32-bitowego była bolesna i dla kogo?
- 10.8. W pracy [COOK82] autor proponuje, żeby tryby adresowania względnego wobec licznika PC zostały wyeliminowane na rzecz innych trybów, takich jak użycie stosu. Jaka jest wada tej propozycji?
- 10.9. Załóżmy, że lista rozkazów przewiduje używanie ustalonej, 16-bitowej długości rozkazu. Specyfikatory argumentu mają długość 6 bitów. W liście znajduje się K rozkazów 2-argumentowych i L 0-argumentowych. Ile maksymalnie może być rozkazów 1-argumentowych?
- 10.10. Zaprojektuj kod operacji o zmiennej długości umożliwiający zakodowanie poniższych rozkazów w postaci rozkazów 36-bitowych:
- 7 rozkazów z dwoma adresami 15-bitowymi i jednym 3-bitowym numerem rejestru;
 - 500 rozkazów z jednym adresem 15-bitowym i jednym 3-bitowym numerem rejestru;
 - 50 rozkazów bez adresów i rejestrów.

* Określenia towarzyszące rozkazowi LOAD (ładuj) dotyczą trybu adresowania (*przyp. tłum.*).

- 10.11. Rozważ wyniki problemu 9.4. Załóż, że M jest 16-bitowym adresem pamięci i że X , Y i Z są albo 16-bitowymi adresami, albo 4-bitowymi numerami rejestrów. Maszyna 1-adresowa używa akumulatora, natomiast maszyny 2- i 3-adresowe mają 16 rejestrów oraz rozkazy operujące na wszystkich kombinacjach lokacji pamięci i rejestrów. Załóż 8-bitowy kod operacji oraz długości rozkazu będące wielokrotnością 4 bitów. Ilu bitów potrzebuje każda z maszyn w celu obliczenia X ?
- 10.12. Czy istnieje możliwe do przyjęcia uzasadnienie stosowania rozkazów z dwoma kodami operacji?
- 10.13. Lista rozkazów procesora Pentium obejmuje następujący rozkaz:

`IMUL op1, op2, immediate`

Rozkaz ten powoduje pomnożenie wartości `op2`, która może być zawarta w rejestrze lub w pamięci, przez natychmiastową wartość argumentu, i umieszczenie wyniku w `op1`, który musi być rejestrem. W liście rozkazów nie występują inne 3-argumentowe rozkazy tego rodzaju. Jakie jest możliwe zastosowanie tego rozkazu? Wskazówka: rozważ indeksowanie.

Struktura i działanie jednostki centralnej

W tym rozdziale zajmiemy się tymi aspektami procesora, które nie zostały ujęte w części III. Stworzymy też podstawy do dyskusji na temat komputerów RISC oraz architektury superskalarnej w rozdz. 12 i 13.

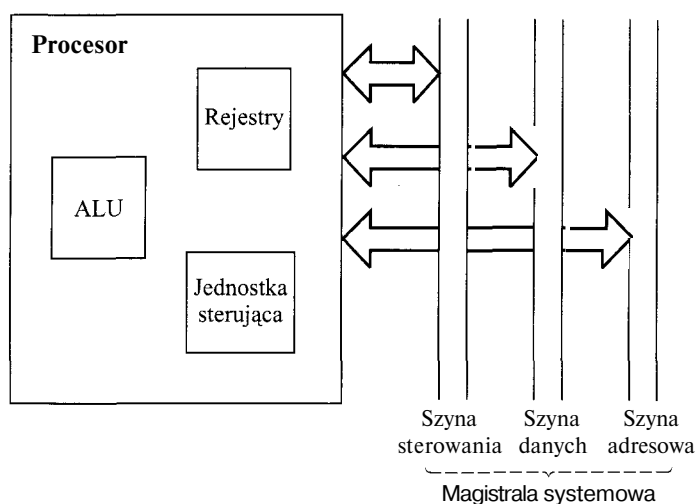
Rozdział rozpoczyna się od podsumowania organizacji procesora. Opiszemy rejestry, które stanowią wewnętrzną pamięć procesora. Następnie wrócimy do dyskusji (rozpoczętej w podrozdz. 3.2) na temat cyklu rozkazu. Omówimy cykl rozkazu oraz powszechnie stosowaną metodę przetwarzania potokowego. Na zakończenie przeanalizujemy pewne dodatkowe aspekty organizacji Pentium i PowerPC.

11.1

ORGANIZACJA PROCESORA

Aby zrozumieć organizację procesora, rozważmy stawiane mu wymagania, a więc zadania, które musi on realizować:

- a *Pobieranie rozkazów.* Procesor musi odczytywać rozkazy z pamięci.
- a *Interpretowanie rozkazów.* Rozkazy muszą być zdekodowane w celu określenia, jakie działania są wymagane.
- a *Pobieranie danych.* Wykonywanie rozkazów może wymagać odczytywania danych z pamięci lub z modułu wejścia-wyjścia.
- a *Przetwarzanie danych.* Wykonywanie rozkazów może wymagać przeprowadzenia na danych pewnych operacji arytmetycznych lub logicznych.
- Q *Zapisanie danych.* Wyniki operacji mogą wymagać zapisania danych w pamięci lub w module wejścia-wyjścia.

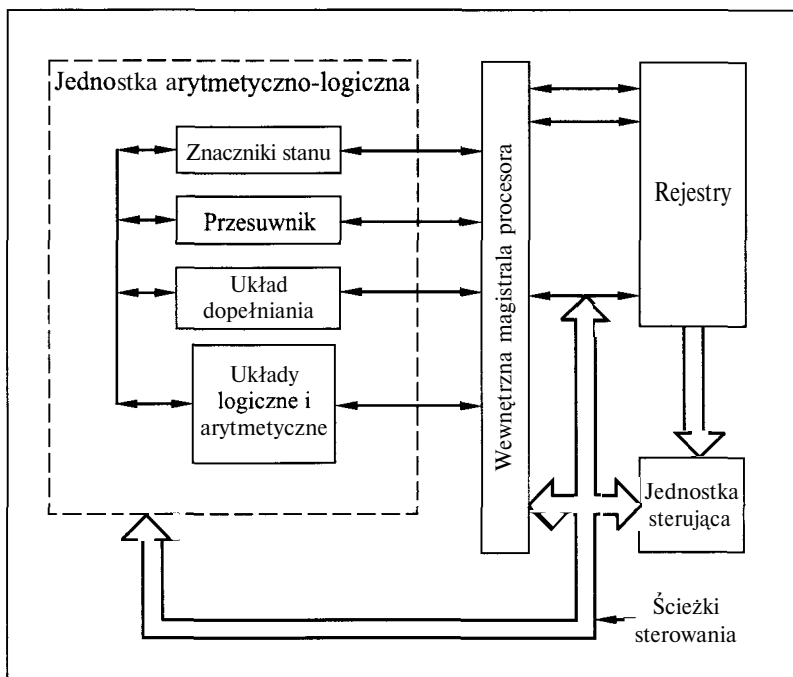


RYSUNEK 11.1. Procesor z magistralą systemową

Jest jasne, że aby procesor mógł wykonywać te zadania, musi mieć możliwość czasowego przechowywania danych. Procesor musi pamiętać lokalizację ostatniego rozkazu, dzięki czemu może wiedzieć, gdzie szukać następnego rozkazu. Wymaga czasowego przechowywania rozkazów i danych podczas wykonywania rozkazu. Innymi słowy, procesor potrzebuje małej pamięci wewnętrznej.

Na rysunku 11.1 jest pokazany uproszczony schemat procesora, wraz z jego połączeniami z resztą systemu poprzez magistralę systemową. Podobny interfejs jest potrzebny dla każdej ze struktur połączeń opisanej w rozdz. 3. Przypomnijmy sobie, że głównymi zespołami procesora są **jednostka arytmetyczno-logiczna (ALU)** i **jednostka sterująca (CU)**. ALU wykonuje rzeczywiste obliczenia lub przetwarza dane. Jednostka sterująca steruje ruchem danych i rozkazów do (oraz z) procesora i steruje pracą ALU. Na rysunku widać też minimalną pamięć wewnętrzną w postaci zespołu lokacji określanego mianem **rejestrów**.

Na rysunku 11.2 widać nieco bardziej szczegółowy schemat procesora. Są na nim pokazane ścieżki transferu danych i sterowania logicznego, włącznie z elementem określonym jako **wewnętrzna magistrala procesora**. Element ten jest potrzebny do przesyłania danych między różnymi rejestrami a ALU, ponieważ w rzeczywistości ALU operuje tylko na danych pochodzących z wewnętrznej pamięci procesora. Na rysunku są również pokazane typowe elementy podstawowe ALU. Zauważmy podobieństwo między wewnętrzną strukturą komputera jako całości a wewnętrzną strukturą procesora. W obu przypadkach występuje niewielki zbiór głównych elementów (komputer: procesor, wejście-wyjście, pamięć; procesor: jednostka sterująca, ALU, rejestry) połączonych ścieżkami danych.



RYSUNEK 11.2. Wewnętrzna struktura procesora

11.2

ORGANIZACJA REJESTRÓW

Jak przedyskutowaliśmy w rozdziale 4, system komputerowy zawiera hierarchię pamięci. Na wyższych poziomach hierarchii pamięć jest szybsza, mniejsza i droższa (w przeliczeniu na bit). Wewnątrz procesora występuje zestaw rejestrów, które działają jako poziom pamięci przewyższający w hierarchii pamięć główną i pamięć podręczną. Rejestry w procesorze można podzielić na dwie grupy:

- *Rejestry widzialne dla użytkownika.* Umożliwiają osobie programującej w języku maszynowym lub symbolicznym minimalizowanie odniesień do pamięci głównej drogą optymalizowania wykorzystania rejestrów.
- a *Rejestry sterowania i stanu.* Są one używane przez jednostkę sterującą do sterowania pracą procesora oraz przez uprzywilejowane programy systemu operacyjnego do sterowania wykonywaniem programów.

Nie ma ścisłego rozgraniczenia między tymi dwiema kategoriami. Na przykład, w pewnych maszynach licznik programu jest widzialny dla użytkownika; (np. VAX), ale w wielu tak nie jest. Jednak dla potrzeb dalszej dyskusji będziemy posługiwali się tymi kategoriami.

Rejestry widzialne dla użytkownika

Rejestr widzialny dla użytkownika jest rejestrem, do którego można się odnosić za pomocą języka maszynowego, którym posługuje się procesor. Praktycznie we wszystkich współczesnych projektach procesora przewidziano pewną liczbę rejestrów widzialnych dla użytkownika, w przeciwieństwie do dawniej stosowanego pojedynczego akumulatora. Możemy je scharakteryzować, posługując się następującym podziałem:

- ☐ ogólnego przeznaczenia, czyli robocze (*general purpose*),
- Q danych,
- a adresów,
- ☐ kodów warunkowych.

Rejestry ogólnego przeznaczenia (robocze) mogą być przypisane przez programistę wielu funkcjom. Czasem ich użycie w ramach listy rozkazów ma charakter ortogonalny w stosunku do operacji. Znaczy to, że dowolny rejestr roboczy może zawierać argument dowolnego kodu operacji. Jest to prawdziwe zastosowanie robocze (uniwersalne). Często jednak występują ograniczenia. Mogą na przykład występować rejestry dedykowane operacjom zmiennopozycyjnym.

W pewnych przypadkach rejestry ogólnego przeznaczenia mogą być używane do funkcji adresowania (np. pośredniego rejestrowego, przemieszczeniowego). W innych przypadkach ma miejsce częściowa lub pełna separacja rejestrów danych od rejestrów adresowych. **Rejestry danych** mogą być używane tylko do przechowywania danych i nie można ich używać do obliczania adresów argumentów. Same **rejestry adresowe** mogą być w pewnym stopniu uniwersalne lub mogą być przypisane określonym trybom adresowania. Do przykładów należą:

- a **Wskaźniki segmentu.** W maszynie z adresowaniem segmentowym (patrz podrozdz. 7.3) rejestr segmentu zachowuje adres podstawy segmentu. Może występować wiele rejestrów: na przykład jeden dla systemu operacyjnego i jeden dla bieżącego procesu.
- a **Rejestry indeksowe.** Są używane do adresowania indeksowego i mogą być indeksowane automatycznie.
- a **Wskaźnik stosu.** Jeśli występuje adresowanie stosowe widzialne dla użytkownika, to zwykle stos znajduje się w pamięci i używany jest dedykowany rejestr wskazujący wierzchołek stosu. Pozwala to na adresowanie domyślne; to znaczy, że rozkazy umieszczania na stosie, zdejmowania ze stosu i inne rozkazy związane ze stosiem nie muszą zawierać jawnego adresu argumentu.

Występuje tu kilka problemów projektowych. Ważnym z nich jest to, czy zastosować wyłącznie uniwersalne rejestry robocze, czy wyspecjalizowane. Zetknęliśmy się już z tym zagadnieniem w poprzednim rozdziale, ponieważ ma ono wpływ na projektowanie list rozkazów. W przypadku stosowania rejestrów wyspecjalizowanych z kodu operacji może domyślnie wynikać, do którego rodzaju rejestru odnosi się określony specyfikator argumentu. Specyfikator argu-

mentu musi tylko określić jeden z zespołu rejestrów wyspecjalizowanych a nie jeden ze wszystkich rejestrów, co pozwala na zaoszczędzenie bitów. Z drugiej strony specjalizacja ogranicza elastyczność programisty. Nie istnieje ostateczny i najlepszy rozwiązanie tego problemu. Jednak, jak już stwierdzono, istnieje tendencja do używania rejestrów wyspecjalizowanych.

Innym zagadnieniem projektowym jest liczba rejestrów (ogólnego przeznaczenia lub wyspecjalizowanych). Jak poprzednio, wpływa to na projekt listy rozkazów, ponieważ więcej rejestrów oznacza więcej bitów specyfikatora argumentu. Jak już wspomnieliśmy, optimum wydaje się leżeć między 8 a 32 rejestrami [LUND77]. Jednak w systemach RISC wystąpiło nowe podejście, oparte na stosowaniu setek rejestrów, co opisujemy w rozdz. 12.

Istnieje wreszcie problem długości rejestrów. Rejestry przeznaczone do przechowywania adresów muszą oczywiście mieć długość wystarczającą do przechowywania najdłuższego adresu. Rejestry danych powinny móc przechowywać większość rodzajów danych. W niektórych maszynach możliwe jest używanie sąsiednich rejestrów do przechowywania wartości o podwójnej długości.

W rejestrach należących do ostatniej kategorii, które są przynajmniej częściowo widzialne dla użytkownika, przechowuje się **kody warunkowe** (nazywane także **znacznikami stanu** lub **flagami**). Kody warunkowe są bitami ustalnymi sprzętowo przez procesor w wyniku operacji. Na przykład operacji arytmetycznej może dać wynik dodatni, ujemny, zerowy lub przepełnienie. Poza skierowaniem do rejestru lub do pamięci samego wyniku, ustalany jest również kod warunkowy. Kod ten może być następnie sprawdzany w ramach operacji rozgałęzienia warunkowego.

Bitowy kod warunkowy są zbierane w jednym lub w wielu rejestrach. Zwykle stanowią one część zawartości rejestru sterowania. Na ogół rozkazy maszyny nowo umożliwiają odczytywanie tych bitów za pomocą odniesienia domyślnego, jednak nie mogą one być zmieniane przez programistę.

W niektórych maszynach wywołanie podprogramu powoduje automatyczne zapisywanie zawartości wszystkich rejestrów widzialnych dla użytkownika, co umożliwia ich odtworzenie po powrocie z podprogramu. Zapisywanie i odtwarzanie jest wykonywane przez procesor w ramach wykonywania rozkazów wywołania i powrotu. Dzięki temu każdy podprogram może niezależnie używać rejestrów widzialnych dla użytkownika. W innych maszynach obowiązkiem programisty jest zapisanie zawartości odpowiednich rejestrów widzialnych dla użytkownika przed wywołaniem podprogramu, przez włączenie do programu niezbędnych rozkazów.

Rejestry sterowania i stanu

W procesorze występują różnorodne rejestry wykorzystywane do sterowania jego pracą. Większość z nich, w większości maszyn, jest niewidzialna dla użytkownika. Niektóre z nich mogą być widzialne dla rozkazów maszynowych wykonywanych w trybie sterowania lub przez system operacyjny.

Oczywiście różne maszyny mają różne organizacje rejestrów; jest też stosowana różna terminologia. Przedstawimy rozsądnie kompletną listę rodzajów rejestrów wraz z krótkim opisem.

Cztery rejestry mają zasadnicze znaczenie dla wykonywania rozkazów:

- a **Licznik programu (PC).** Zawiera adres rozkazu przewidzianego do pobrania.
- 3 **Rejestr rozkazu (IR).** Zawiera ostatnio pobrany rozkaz.
- 3 **Rejestr adresowy pamięci (MAR).** Zawiera adres lokacji w pamięci.
- 3 **Rejestr buforowy pamięci (MBR).** Zawiera słowo danych, które ma być zapisane do pamięci lub które zostało ostatnio odczytane z pamięci.

Licznik programu zawiera adres rozkazu. Zwykle jest on aktualizowany przez procesor po każdym pobraniu rozkazu, dzięki czemu zawsze wskazuje on następny rozkaz przewidziany do pobrania. Rozkaz rozgałęzienia lub pominięcia również modyfikuje zawartość licznika PC. Pobrany rozkaz jest ładowany do rejestru rozkazu, gdzie analizowane są kod operacji i specyfikator argumentu. Dane są wymieniane z pamięcią za pośrednictwem rejestrów MAR i MBR. W systemie magistralowym rejestr MAR jest połączony bezpośrednio z magistralą adresową, a MBR - z magistralą danych. Z kolei rejestry widzialne dla użytkownika wymieniają dane z rejestrem MBR.

Cztery wspomniane rejestry są używane do przenoszenia danych między procesorem a pamięcią. Wewnątrz procesora dane muszą być doprowadzone do ALU w celu ich przetwarzania. ALU musi mieć bezpośredni dostęp do rejestru MBR i rejestrów widzialnych dla użytkownika. Alternatywnie mogą występować dodatkowe rejestry buforujące na granicy z ALU; rejestry te służą jako rejestry wejściowe oraz wyjściowe dla ALU i wymieniają dane z rejestrem MBR i z rejestrami widzialnymi dla użytkownika.

We wszystkich projektach procesora jest rejestr lub zespół rejestrów, określanych często jako **słowo stanu programu** (ang. *program status word* - PSW). Rejestry te zawierają informację o stanie. W rejestrach PSW są zwykle przechowywane kody warunkowe i inne informacje o stanie. Występują tam zwykle następujące pola (znaczniki stanu, czyli flagi):

- 3 **Znak.** Zawiera bit znaku wyniku ostatniej operacji arytmetycznej.
- 3 **Zero.** Ustawiane, gdy wynik jest równy zeru.
- 3 **Przeniesienie.** Ustawiane, gdy wynikiem operacji jest przeniesienie (dodawanie) lub przeniesienie zanegowane (odejmowanie). Używane przy operacjach arytmetycznych obejmujących wiele słów.
- 3 **Równość.** Ustawiane, gdy wynikiem porównywania logicznego jest równość.
- 3 **Przepełnienie.** Używane do wskazywania przepełnienia arytmetycznego.
- 3 **Zezwolenie/blokowanie przerwania.** Używane do obsługi przerw.
- 3 **Nadzorca.** Wskazuje, czy procesor pracuje w trybie nadzorcy, czy w trybie użytkownika. Niektóre rozkazy uprzywilejowane mogą być wykonywane

tylko w trybie nadzorcy. Również pewne obszary pamięci mogą być dostępne tylko w trybie nadzorcy.

Istnieje wiele innych rejestrów związanych ze stanem i ze sterowaniem, które można odnaleźć w konkretnych projektach procesorów. Obok PSW może występować wskaźnik bloku pamięci zawierającego dodatkowe informacje o stanie (np. bloków sterowania procesem). W maszynach stosujących przerwania wektorowe może być przewidziany rejestr wektora przerwania. Jeśli do implementacji pewnych funkcji jest używany stos (np. do wywołania podprogramu), to potrzebny jest systemowy wskaźnik stosu. Z systemami pamięci wirtualnej współpracuje wskaźnik tablic stron. Rejestry mogą być wreszcie używane do sterowania operacjami wejścia-wyjścia.

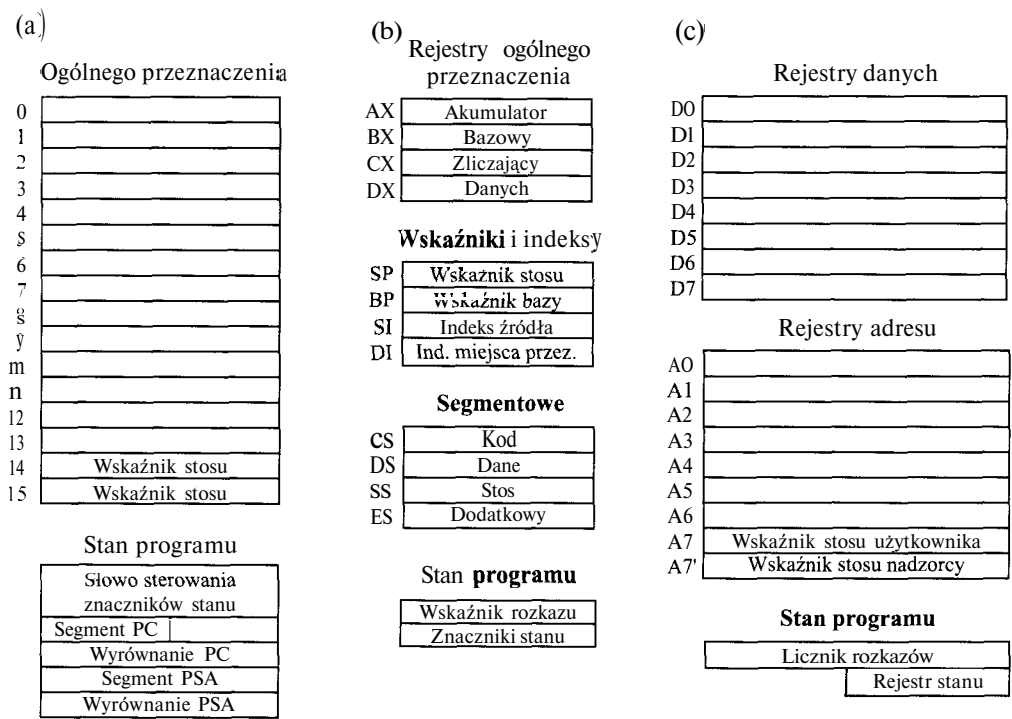
Wiele czynników wpływa na organizację rejestrów sterowania i stanu. Podstawowym zagadnieniem jest wspieranie systemu operacyjnego. Pewne rodzaje informacji sterowania mają szczególne znaczenie dla systemu operacyjnego. Jeśli projektant wie jak działa przewidziany do stosowania system operacyjny, to organizacja rejestrów może być do pewnego stopnia dostosowana do systemu operacyjnego.

Inną podstawową decyzją projektową jest podział informacji sterowania między rejestry a pamięć. Powszechną praktyką jest przeznaczanie pierwszych (o najniższych adresach) kilkuset lub kilku tysięcy słów pamięci do celów sterowania. Projektant musi zdecydować, ile informacji sterowania musi pozostać w rejestrach, a ile w pamięci. Zachodzi tu typowa wymiennosc między kosztem a szybkością.

Przykładowe organizacje rejestrów w mikroprocesorach

Pouczające jest przeanalizowanie i porównanie organizacji rejestrów w porównywalnych systemach. W tym punkcie rozpatrzmy trzy mikroprocesory 16-bitowe zaprojektowane prawie w tym samym czasie: Zilog Z8000 [PEUT79], Intel 8086 [MORS78, HEYW83] oraz Motorola MC68000 [STRI79]. Na rysunku 11.3 widać organizację rejestrów każdego z nich; czysto wewnętrzne rejestry, takie jak rejestry adresu pamięci, nie zostały pokazane.

W Z8000 użyto szesnastu 16-bitowych rejestrów ogólnego przeznaczenia (roboczych), które mogą być używane do przechowywania danych, adresów i do indeksowania. Projektanci uważali, że ważniejsze jest dysponowanie regularnym zespołem uniwersalnych rejestrów roboczych, niż oszczędzanie bitów rozkazu przez stosowanie rejestrów wyspecjalizowanych. Ponadto woleli oni pozostawić programiście przypisanie funkcji rejestrów, zakładając, że mogą istnieć różne potrzeby wynikające z różnych zastosowań. Rejestry te mogą być także używane do operacji 8- i 32-bitowych. Ustalono przestrzeń adresu segmentowego (7-bitowy numer segmentu, 16-bitowy adres względny). Do przechowywania jednego adresu są potrzebne 2 rejestry. Dwa spośród rejestrów są również



RYSUNEK 11.3. Organizacja rejestrów w mikroprocesorach: (a) Z8000; (b) Intel 8086; (c) MC 8000

używane jako domyślne wskaźniki stosu dla trybu systemowego i dla trybu normalnego.

Procesor Z8000 zawiera również 5 rejestrów związanych ze stanem programu. W dwóch rejestrach jest przechowywany stan licznika programu, a w dwóch - adres obszaru stanu programu w pamięci. W 16-bitowym rejestrze znaczników stanu są przechowywane różne bity stanu i sterowania.

W procesorze Intel 8086 przyjęto inne podejście do organizacji rejestrów. Każdy rejestr jest wyspecjalizowany, chociaż niektóre z nich mogą również służyć jako rejestry ogólnego przeznaczenia. Procesor 8086 zawiera cztery 16-bitowe rejestry danych, adresowalne na podstawie bajtowej lub 16-bitowej, oraz cztery 16-bitowe rejestry wskaźnikowe i indeksowe. W odniesieniu do niektórych rozkazów rejestry danych mogą służyć jako robocze. W odniesieniu do pozostałych rozkazów są one używane domyślnie. Na przykład rozkaz mnożenia zawsze korzysta z akumulatora. Cztery rejestry wskaźnikowe również w wielu operacjach są używane domyślnie; każdy z nich zawiera adres względny w segmencie. Istnieją również cztery 16-bitowe rejestry segmentu. Trzy z nich są używane w sposób dedykowany (domyślny) w celu wskazania segmentu bieżącego rozkazu (przydatne przy rozkazach rozgałęziania), segmentu zawierającego dane oraz segmentu zawiera-

jącego stos. Specjalizacja rejestrów umożliwia uzyskanie zwięzłego kodu kosztem elastyczności. Procesor 8086 ma także wskaźnik rozkazu i zespół 1-bitowych znaczników stanu i sterowania.

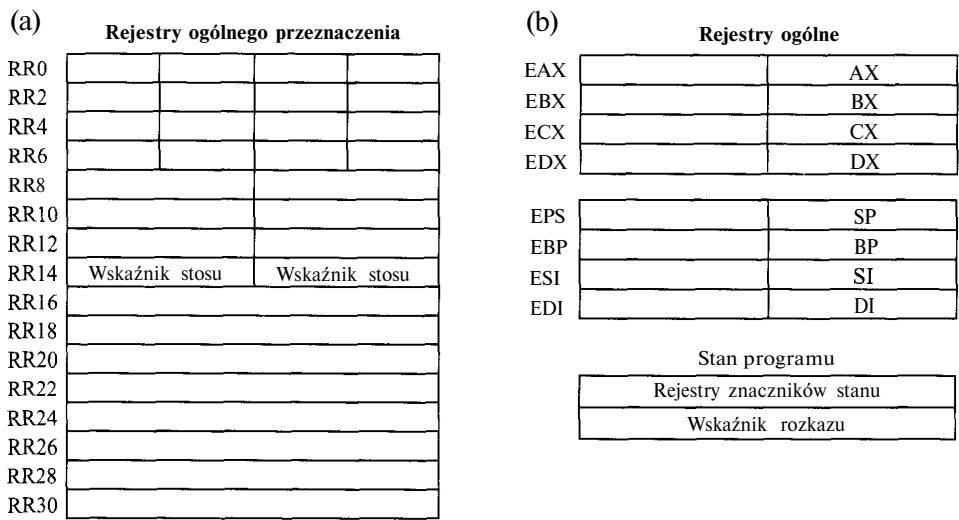
Organizacja rejestrów procesora Motorola MC68000 mieści się gdzieś pomiędzy koncepcjami projektowymi mikroprocesorów Zilog i Intel. Wśród rejestrów procesora MC68000 jest osiem 32-bitowych rejestrów danych i 9 rejestrów adresowych. Ośmiu rejestrów danych używa się głównie do manipulowania danymi. Mogą być one używane do adresowania tylko jako rejestry indeksowe. Wielkość rejestrów umożliwia przeprowadzanie operacji na danych 8-, 16- i 32-bitowych, określonych przez kod operacji. Rejestry adresowe zawierają adresy 32-bitowe (bez segmentowania); dwa z nich są również używane jako wskaźniki stosu, jeden przeznaczony dla użytkowników a drugi dla systemu operacyjnego, zależnie od bieżącego trybu wykonywania. Oba rejestry noszą numer 7, ponieważ tylko jeden z nich może być używany w określonej chwili. Procesor MC68000 zawiera także 32-bitowy licznik programu i 16-bitowy rejestr stanu.

Podobnie jak projektanci firmy Zilog, zespół Motoroli przyjął bardzo regularną listę rozkazów, bez rejestrów wyspecjalizowanych. Troska o sprawność kodowania doprowadziła do podziału rejestrów na dwie grupy funkcjonalne, co pozwoliło na zaoszczędzenie jednego bitu w każdym specyfikatorze rejestru. Wygląda to na rozsądny kompromis między uniwersalnością a zwartością programów.

Porównanie to powinno być jasne. Jak dotychczas nie istnieje ogólnie akceptowana filozofia dotycząca najlepszego sposobu organizacji rejestrów procesora [TOON81]. Podobnie jak w przypadku projektowania list rozkazów i w przypadku wielu innych problemów projektowania procesorów, jest to wciąż sprawa właściwego osądu i gustu.

Drugi pouczający problem dotyczący projektowania organizacji rejestrów jest zilustrowany na rys. 11.4. Widać na nim organizację rejestrów widzialnych dla użytkownika procesorów Zilog 80000 [PHIL85] i Intel 80386 [ELAY85], które są 32-bitowymi mikroprocesorami zaprojektowanymi odpowiednio jako rozszerzenia Z8000 i 8086¹. Oba te nowe procesory używają rejestrów 32-bitowych. Jednak w celu zapewnienia kompatybilności programów napisanych dla wcześniejszych maszyn, zachowują one oryginalną organizację rejestrów osadzoną w nowej organizacji. Napotykając to ograniczenie projektowe, architekci procesorów 32-bitowych dysponowali ograniczoną swobodą w projektowaniu organizacji rejestrów.

¹ Ponieważ w procesorze MC 68000 zastosowano już rejestry 32-bitowe, w MC 68020 [MACG84], stanowiącym pełne rozszerzenie 32-bitowe, użyto tej samej organizacji rejestrów.



RYSUNEK 11.4. Rozszerzenia organizacji rejestrów w mikroprocesorach 32-bitowych: (a) Z80000; (b) Intel 80386

11.3

CYKL ROZKAZU

W podrozdziale 3.2 opisaliśmy cykl rozkazu procesora. Rysunek 11.5 stanowi powtórzenie rys. 3.9. Przypomnijmy sobie składniki cyklu rozkazu:

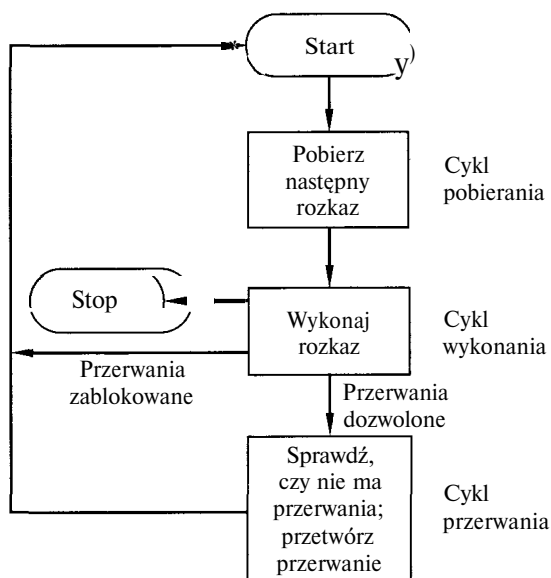
- ☐ *Pobranie.* Wczytanie następnego rozkazu z pamięci do procesora.
- a *Wykonanie.* Zinterpretowanie kodu operacji i wykonanie wskazanej operacji.
- ☐ *Przerwanie.* Jeśli dozwolone są przerwania i wystąpiło przerwanie, to następuje zachowanie bieżącego stanu procesu i obsłużenie przerwania.

Możemy teraz nieco głębiej przeanalizować cykl rozkazu. Najpierw musimy wprowadzić dodatkowy składnik, nazywany cyklem pośrednim.

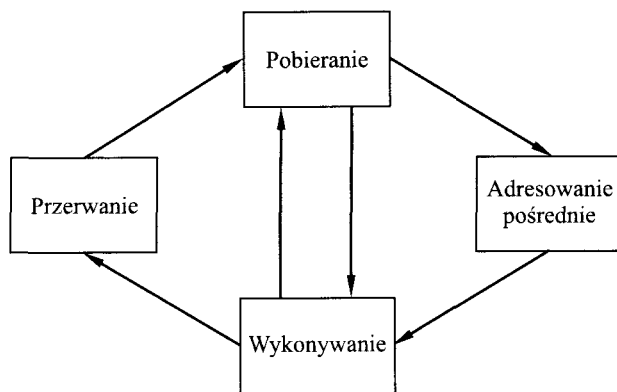
Cykl pośredni

Widzieliśmy w rozdziale 10, że wykonywanie rozkazu może obejmować powołanie na jeden lub więcej argumentów przechowywanych w pamięci, z których każdy wymaga dostępu do pamięci. Ponadto, jeśli jest stosowane adresowanie pośrednie, potrzebne są dodatkowe odniesienia do pamięci.

Możemy traktować pobieranie adresów pośrednich jako dodatkowy element cyklu rozkazu. Wynik widać na rys. 11.6. Działanie tego układu polega na prze-



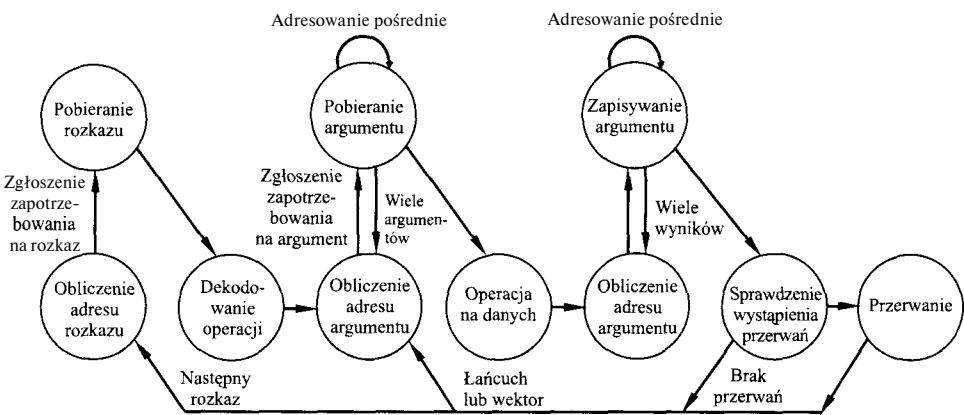
RYSUNEK 11.5. Cykl rozkazu w przypadku przerw



RYSUNEK 11.6. Cykl rozkazu

skakiwaniu między czynnością pobierania rozkazu a czynnością wykonywania rozkazu. Po pobraniu rozkaz jest badany w celu stwierdzenia, czy występuje w nim jakiekolwiek adresowanie pośrednie. Jeśli tak, to są pobierane wymagane argumenty za pomocą adresowania pośredniego. Po wykonaniu może być przetwarzane przerwanie, zanim zostanie pobrany następny rozkaz.

Innym sposobem widzenia tego procesu jest schemat pokazany na rys. 11.7, będący zrewidowaną wersją rys. 3.12. Ilustruje on dokładniej naturę cyklu rozkazu. Gdy rozkaz zostanie pobrany, muszą być zidentyfikowane jego specyfikatory argumentów. Następnie pobierany jest z pamięci każdy argument wejś-



RYSUNEK 11.7. Graf stanów cyklu rozkazu

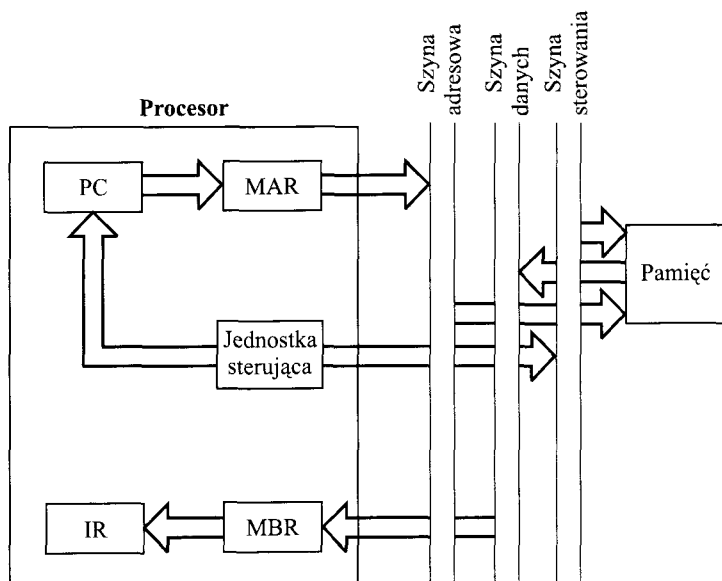
ciowy; proces ten może wymagać adresowania pośredniego. Argumenty przechowywane w rejestrach nie muszą być pobierane. Gdy operacja zostanie wykonana, może zajść potrzeba wykonania podobnego ciągu operacji w celu zapisania wyniku w pamięci głównej.

Przepływ danych

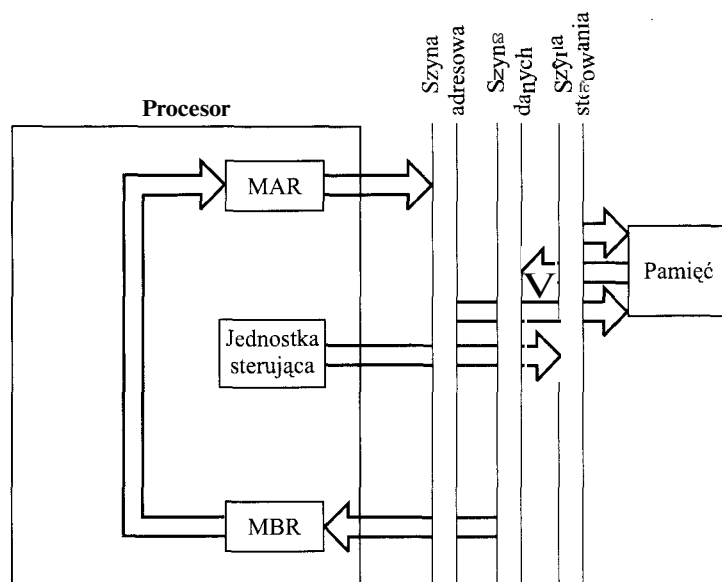
Dokładna sekwencja zdarzeń podczas cyklu rozkazu zależy od projektu procesora. Możemy jednak pokazać ogólnie, co musi się zdarzyć. Załóżmy, że procesor dysponuje rejestrem adresowym pamięci (MAR), rejestrem buforowym pamięci (MBR), licznikiem programu (PC) i rejestrem rozkazu (IR).

Podczas *cyklu pobierania* następuje odczytanie adresu z pamięci. Na rysunku 11.8 jest pokazany przepływ danych podczas tego cyklu. Licznik PC zawiera adres następnego rozkazu przewidzianego do pobrania. Adres ten jest przenoszony do rejestru MAR i umieszczany na szynie adresowej. Jednostka sterująca zgłasza zapotrzebowanie na odczyt z pamięci, a wynik jest umieszczany na szynie danych, kopiowany do MBR, a następnie przenoszony do IR. W tym samym czasie PC przyrasta o 1, co stanowi przygotowanie do pobrania następnego rozkazu.

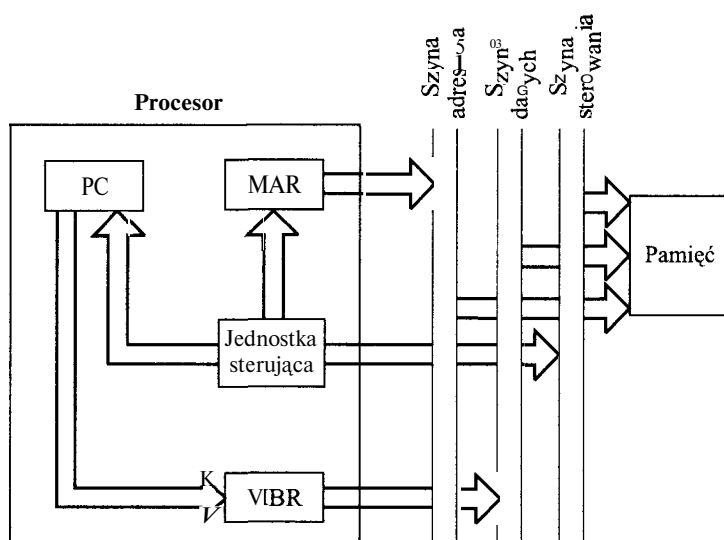
Gdy cykl pobierania jest zakończony, jednostka sterująca bada zawartość rejestru IR w celu stwierdzenia, czy zawiera on specyfikator argumentu wykorzystujący adresowanie pośrednie. Jeśli tak, to jest przeprowadzany *cykl pośredni*. Jak widać na rys. 11.9, jest to prosty cykl. Do rejestru MAR jest przenoszonych *N* najbardziej znaczących bitów rejestru MBR, zawierających odniesienie do adresu. Następnie jednostka sterująca zgłasza zapotrzebowanie na odczyt z pamięci w celu wprowadzenia pożądanego adresu argumentu do rejestru MBR.



RYSUNEK 11.8. Przepływ danych w cyklu pobierania



RYSUNEK 11.9. Przepływ danych w cyklu pośrednim



RYSUNEK 11.10. Przepływ danych w cyklu przerwania

Cykl pobierania i cykl pośredni są proste i przewidywalne. *Cykl rozkazu* przybiera wiele postaci, ponieważ postać ta zależy od tego, który z różnorodnych rozkazów maszynowych znajduje się w rejestrze IR. Cykl rozkazu może obejmować przesyłanie danych między rejestrami, odczytanie lub zapisanie w pamięci, operację wejścia-wyjścia oraz (lub) wywołanie ALU.

Podobnie jak cykl pobierania i cykl pośredni, *cykl przerwania* jest prosty i przewidywalny (rys. 11.10). Bieżąca zawartość licznika PC musi być zapisana, żeby procesor mógł wznowić normalną działalność po przerwaniu. Wobec tego zawartość licznika PC jest przenoszona do rejestru MBR w celu zapisania w pamięci. Zawartość specjalnej, zarezerwowanej do tego celu lokacji pamięci jest ładowana do MAR przez jednostkę sterującą. Może to być na przykład wskaźnik stosu. Do licznika PC jest ładowany adres procedury przerwania. W rezultacie rozpoczyna się następny cykl rozkazu przez pobranie odpowiedniego rozkazu.

11.4

POTOKOWE PRZETWARZANIE ROZKAZÓW

W miarę rozwoju systemów komputerowych możliwe stało się osiąganie większej wydajności poprzez wykorzystywanie udoskonaleń technologicznych, takich jak szybsze układy. Ponadto, wydajność może być zwiększana przez lepszą organizację procesora. Widzieliśmy już tego przykłady, takie jak stosowanie wielu rejestrów zamiast pojedynczego akumulatora oraz użycie pamięci podręcznej. Innym rozwiązaniem organizacyjnym, obecnie powszechnym, jest potokowe przetwarzanie rozkazów.

Strategia przetwarzania potokowego

Przetwarzanie potokowe rozkazów jest podobne do użycia linii montażowej w zakładzie produkcyjnym. Dzięki zorganizowaniu procesu produkcyjnego w formie linii montażowej, możliwa jest jednoczesna praca nad wyrobami w różnych stadiach produkcji. Proces ten nazwano przetwarzaniem potokowym ponieważ, tak jak w potoku, na jednym końcu są przyjmowane nowe elementy wejściowe, zanim jeszcze elementy poprzednio przyjęte ukażą się na wyjściu.

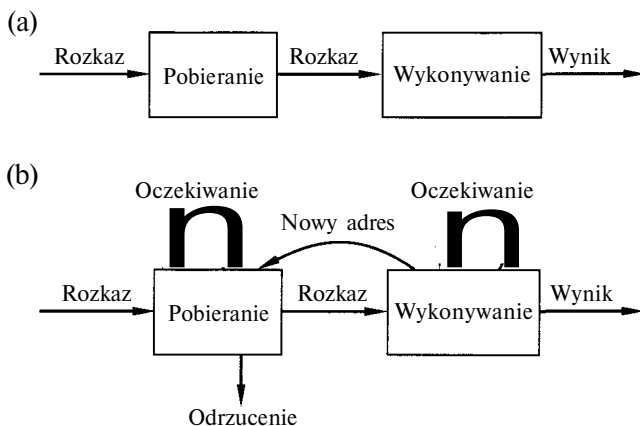
W celu zastosowania tej koncepcji do wykonywania rozkazów musimy założyć, że w rzeczywistości wykonywanie rozkazów ma wiele etapów. Na przykład na rys. 11.7 cykl rozkazu został podzielony na 10 kolejno realizowanych zadań. Jest oczywiste, że istnieją tu warunki umożliwiające zastosowanie przetwarzania potokowego.

Rozpatrzmy najpierw prosty podział przetwarzania rozkazu na dwa etapy: pobranie rozkazu i wykonanie rozkazu. Występują takie przedziały czasowe podczas wykonywania rozkazu, w których nie następują odniesienia do pamięci głównej. Przedziały te mogłyby być wykorzystane do pobrania następnego rozkazu równolegle z wykonywaniem bieżącego. Rozwiązanie to jest przedstawione na rys. 11.11a. Potok przebiega na 2 niezależnych etapach. Na pierwszym etapie następuje pobranie i zbuforowanie rozkazu. Gdy drugi etap jest wolny, następuje przekazanie do niego zbuforowanego rozkazu. Podczas gdy na drugim etapie ma miejsce wykonywanie rozkazu, na pierwszym etapie wykorzystuje się wolne z nieużywanych cykli pamięci w celu pobrania i zbuforowania następnego rozkazu. Nazywa się to **pobieraniem rozkazu z wyprzedzeniem** (ang. *instruction prefetch*) lub **pobieraniem na zakładkę** (ang. *fetchoverlap*).

Powinno być jasne, że ten proces przyspiesza wykonywanie rozkazów. Jeśli etapy pobierania i wykonywania zajmowałyby tyle samo czasu, czas cyklu rozkazu zmniejszyłby się do połowy. Jeśli jednak przyjrzymy się bliżej przetwarzaniu potokowemu (rys. 11.11b), to zobaczymy, że z dwóch powodów to podwojenie szybkości wykonywania nie jest prawdopodobne:

1. Czas wykonywania jest na ogół dłuższy niż czas pobierania. Wykonywanie obejmuje odczytywanie i przechowywanie argumentów oraz przeprowadzenie samej operacji. Wobec tego etap pobierania musi zawierać oczekiwanie przez pewien czas, zanim będzie mogło nastąpić opróżnienie bufora.
2. Rozkaz skoku warunkowego powoduje, że adres następnego rozkazu przewidzianego do pobrania jest nieznany. Wobec tego realizacja etapu pobierania może nastąpić dopiero po otrzymaniu adresu następnego rozkazu, który zostanie określony po zakończeniu etapu wykonywania. Następnie na etapie wykonywania następuje oczekiwanie na pobranie kolejnego rozkazu.

Strata czasu z tego drugiego powodu może być zmniejszona przez zgadywanie. Prosta reguła jest następująca. Gdy rozkaz rozgałęzienia warunkowego przechodzi z etapu pobierania do etapu wykonywania, na etapie pobierania na-



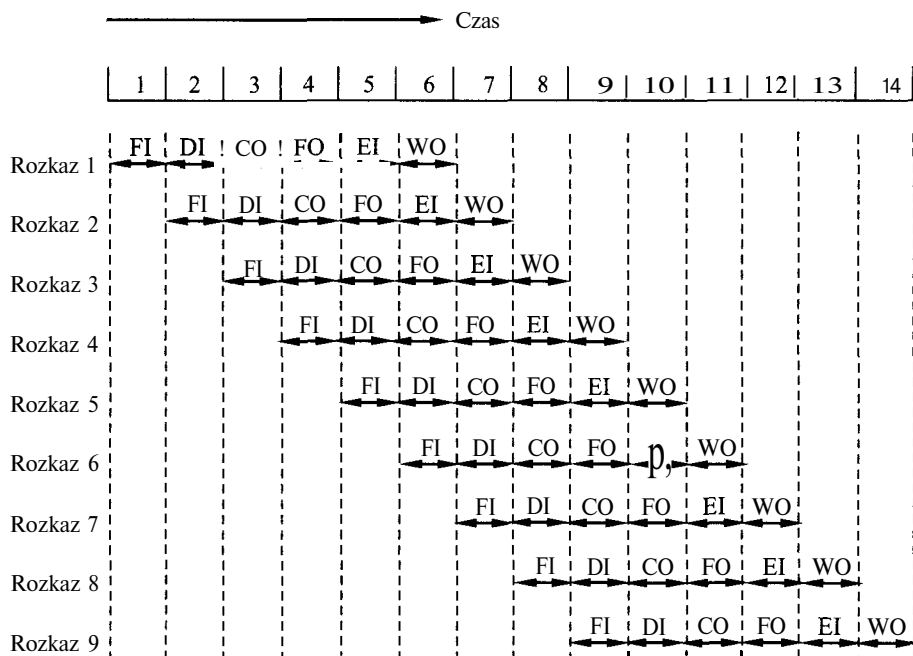
RYСУNEK 11.11. Dwuetapowy potok rozkazów: (a) schemat uproszczony; (b) schemat poszerzony

stępuje pobranie następnego rozkazu z pamięci po rozkazie rozgałęzienia. Wtedy, jeśli nie nastąpi rozgałęzienie, czas nie jest stracony. Gdy rozgałęzienie nastąpi, pobrany rozkaz musi być usunięty, a pobrany nowy.

Chociaż powyższe czynniki powodują zmniejszenia potencjalnej efektywności 2-etapowego potoku, pewne przyspieszenie jednak następuje. Aby uzyskać większe przyspieszenie, potok musi być przetwarzany na większej liczbie etapów. Rozważmy następującą dekompozycję przetwarzania rozkazu:

- *Pobranie rozkazu (FI)*. Wczytanie następnego spodziewanego rozkazu do bufora.
- *Dekodowanie rozkazu (DI)*. Określenie kodu operacji i specyfikatorów argumentu.
- *Obliczanie argumentów (CO)*. Obliczenie efektywnego adresu każdego argumentu źródłowego. Może to obejmować obliczanie adresu z przesunięciem, adresu rejestrowego pośredniego, adresu pośredniego lub innych form.
- a *Pobieranie argumentów (FO)*. Pobranie każdego argumentu z pamięci. Argumenty w rejestrach nie muszą być pobierane.
- a *Wykonanie rozkazu (EI)*. Przeprowadzenie wskazanej operacji i przechowanie wyniku, jeśli taki jest, w ustalonej lokacji docelowej.
- a *Zapisanie argumentu (WO)*. Zapisanie wyniku w pamięci.

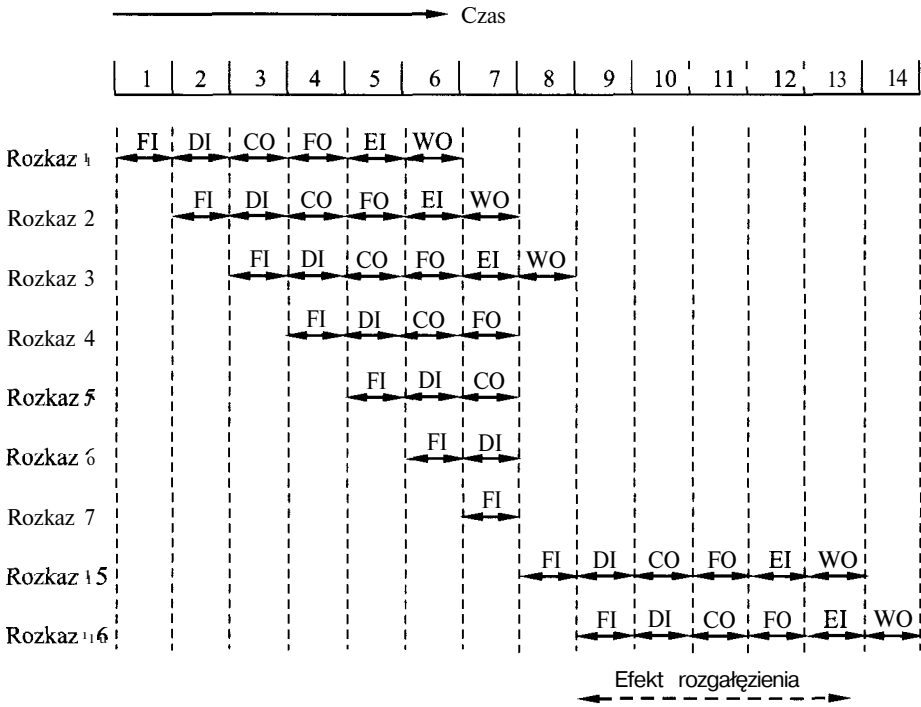
Przy takiej dekompozycji czas trwania różnych etapów może być zbliżony. Dla celów dyskusji przyjmijmy, że czasy te są równe. Przy takim założeniu, jak pokazano na rys. 11.12, stosując 6-etapowy potok zmniejsza się czas wykonywania 9 rozkazów z 54 jednostek czasu do 14.



RYСУNEK 11.12. Przebieg czasowy przetwarzania potoku rozkazów

Poczyńmy kilka komentarzy. Na rysunku przyjęliśmy, że każdy rozkaz przechodzi przez wszystkie 6 etapów potoku. Nie zawsze tak jest. Na przykład rozkaz ładowania nie wymaga etapu WO. Dla uproszczenia zakłada się jednak, że każdy rozkaz wymaga 6 etapów. Zakłada się również, że wszystkie etapy mogą być realizowane równolegle. W szczególności przyjmuje się, że nie występują konflikty dostępu do pamięci. Na przykład etapy FI, FO i WO zawierają odniesienia do pamięci. Na wykresie założono, że wszystkie te odniesienia mogą następować jednocześnie. W większości systemów pamięci nie jest to możliwe. Jednak pożądana wartość może się znajdować w pamięci podręcznej lub też etapy FO czy WO mogą nie być realizowane. Wobec tego w większości przypadków konflikty dostępu do pamięci nie spowodują spowolnienia potoku.

Poprawę wydajności ogranicza kilka innych czynników. Jeśli czasy trwania 6 etapów nie są równe, na różnych etapach wystąpi pewne oczekiwanie, podobne do wykazanego w przypadku potoku 2-etapowego. Inną trudność stanowi rozkaz rozgałęzienia warunkowego, który może unieważnić kilka pobranych rozkazów. Podobnym nieprzewidywalnym zdarzeniem jest przerwanie. Na rysunku 11.13 jest pokazany wpływ rozgałęzienia warunkowego przy zastosowaniu tego samego programu, co na rys. 11.12. Załóżmy, że rozkaz 3 jest rozkazem rozgałęzienia warunkowego do rozkazu 15. Aż do wykonania tego rozkazu nie ma możliwości przewidzenia, który rozkaz będzie wykonywany jako następny. W tym przykładzie w potoku jest po prostu ładowany następny w kolejności rozkaz (rozkaz 4) i przetwarzanie jest kontynuowane. Na rysunku 11.12 rozgałę-



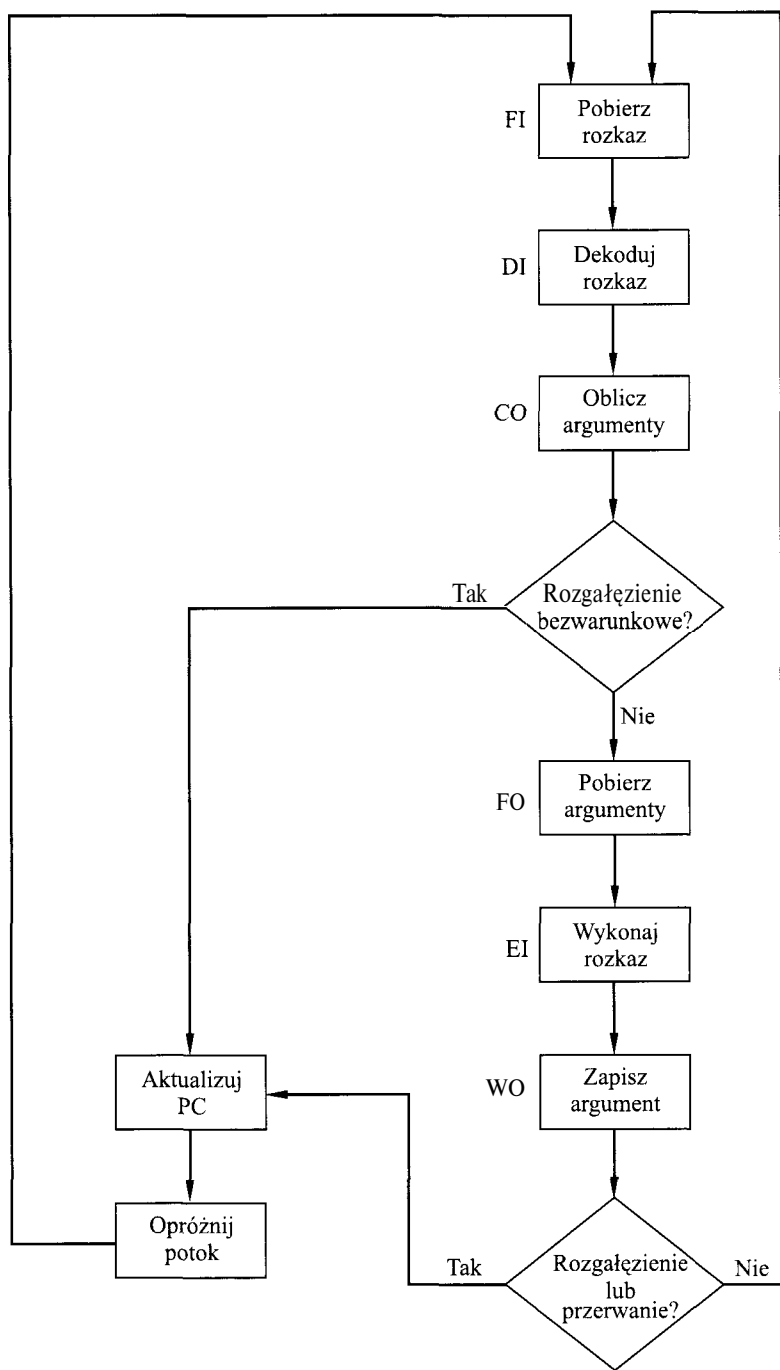
RYSunEK 11.13. Wpływ rozgałęzienia warunkowego na funkcjonowanie potoku rozkazów

zienie nie następuje i uzyskujemy pełną poprawę wydajności. Na rysunku 11.13 rozgałęzienie następuje, jednak nie jest to wiadome aż do końca 7 jednostki czasu. W tej chwili potok musi być oczyszczony z bezużytecznych rozkazów. Podczas 8 jednostki czasu, rozkaz 15 wchodzi do potoku. W czasie od jednostki 9 do 12 nie kończy się wykonywanie żadnego rozkazu; jest to strata wydajności wynikająca z tego, że nie mogliśmy przewidzieć rozgałęzienia. Na rysunku 11.14 jest pokazany schemat logiczny przetwarzania potokowego uwzględniający rozgałęzienia i przerwania.

Powstają też inne problemy, które nie miały miejsca przy potoku 2-etapowym. Etap CO może być uzależniony od zawartości rejestru, która mogłaby być zmieniona przez poprzedni rozkaz będący w dalszym ciągu w potoku. Mogą występować inne takie konflikty rejestrów lub pamięci. System musi zawierać rozwiązania logiczne zapobiegające konfliktom tego rodzaju.

Na podstawie dotychczasowej dyskusji mogłoby się wydawać, że im większa jest liczba etapów potoku, tym szybsze jest wykonywanie rozkazów. Niektórzy z projektantów IBM S/360 wskazali na dwa czynniki, które zaburzają ten pozornie prosty sposób zwiększania wydajności [ANDE67]. Pozostaje to słuszne do dzisiaj.

1. Na każdym etapie potoku występuje pewien narzut związany z przenoszeniem danych z bufora do bufora oraz z wykonywaniem różnych działań



RYSUNEK 11.14. Sześciostopowy potok rozkazów procesora

przygotowawczych. Ten narzut może wyraźnie wydłużyć całkowity czas wykonywania pojedynczego rozkazu. Jest to znaczące, gdy rozkazy sekwencyjne są logicznie zależne albo przez częste rozgałęzianie, albo przez uzależniony dostęp do pamięci.

2. Liczba układów logicznych wymaganych do zapobiegania konfliktom rejestrów i pamięci oraz do optymalizacji potoku wzrasta znacząco wraz z liczbą etapów. Może to prowadzić do sytuacji, w której układy logiczne sterujące przejściami między etapami są bardziej złożone niż same etapy podlegające sterowaniu.

Potokowe przetwarzanie rozkazów jest skuteczną metodą zwiększania wydajności, jednak wymaga starannego projektowania, aby osiągnąć optymalne wyniki przy rozsądnej złożoności.

Postępowanie z rozgałęzieniami

Jednym z głównych problemów przy projektowaniu potoków rozkazów jest zapewnienie stałego napływu rozkazów do początkowych etapów potoku. Jak widzieliśmy, zasadniczą przeszkodą jest rozkaz rozgałęzienia warunkowego. Aż do zakończenia wykonywania tego rozkazu nie jest możliwe stwierdzenie, czy rozgałęzienie nastąpi, czy nie.

Zastosowano wiele sposobów postępowania z rozkazami rozgałęzienia warunkowego:

- a zwielokrotnienie strumienia,
- pobieranie docelowego rozkazu z wyprzedzeniem,
- bufor pętli,
- Q przewidywanie rozgałęzienia,
- a opóźnione rozgałęzienie.

Zwielokrotnione strumienie

W przypadku prostego potoku występują straty spowodowane przez rozkaz rozgałęzienia, ponieważ konieczne jest wybranie jednego z dwóch możliwych rozkazów i wybór ten może się okazać niewłaściwy. Brutalnym rozwiązaniem jest powielenie początkowych części potoku i umożliwienie równoczesnego pobrania obu rozkazów za pomocą dwóch strumieni. Rozwiązanie to stwarza pewne problemy:

- a W przypadku zwielokrotnionego strumienia występują opóźnienia wynikające z rywalizacji o dostęp do rejestrów i pamięci.
- Następne rozkazy rozgałęzienia mogą wejść do potoku (którymkolwiek strumieniem), zanim zostanie przesądzone oryginalne rozgałęzienie. Każdy taki rozkaz wymaga dodatkowego strumienia.

Mimo powyższych wad strategia ta może poprawić wydajność. Przykładami maszyn z dwoma lub z większą ilością strumieni są IBM 370/168 oraz IBM 3033.

Pobieranie docelowego rozkazu z wyprzedzeniem

Gdy rozpoznawany jest rozkaz rozgałęzienia warunkowego, następuje wyprzedzające pobranie rozkazu docelowego razem z rozkazem następującym po rozgałęzieniu. Rozkaz docelowy jest następnie zachowywany, aż do czasu wykonania rozkazu rozgałęzienia. W momencie rozgałęzienia rozkaz docelowy jest już pobrany.

Rozwiązanie to zastosowano w IBM 360/91.

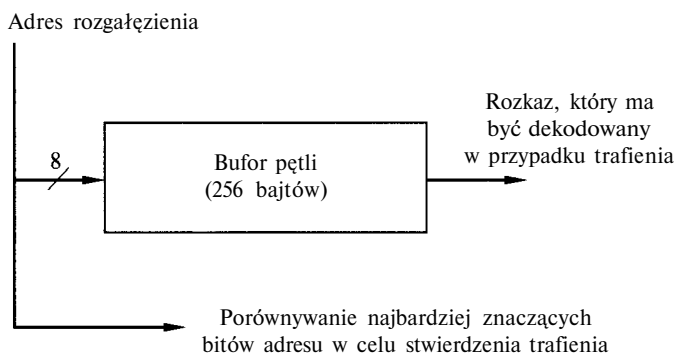
Bufor pętli

Bufor pętli jest małą, bardzo szybką pamięcią związaną z etapem pobierania rozkazów potoku. Zawiera on kolejno n ostatnio pobranych rozkazów. Jeśli ma nastąpić rozgałęzienie, sprawdza się najpierw, czy cel rozgałęzienia znajduje się wewnątrz bufora. Jeśli tak, to następny rozkaz jest pobierany z bufora. Stosowanie bufora pętli ma trzy zalety:

1. Dzięki pobieraniu z wyprzedzeniem bufor pętli zawiera pewne rozkazy następujące po kolei po adresie pobrania bieżącego rozkazu. Wobec tego kolejno pobierane rozkazy będą dostępne bez straty czasu wymaganej na dostęp do pamięci.
2. Jeśli następuje rozgałęzienie do rozkazu znajdującego się o kilka pozycji dalej od adresu rozkazu rozgałęzienia, to rozkaz docelowy będzie się już znajdował w buforze. Jest to użyteczne w przypadku stosunkowo powszechnych sekwencji IF-THEN (jeśli-to) oraz IF-THEN-ELSE (jeśli-to-w-przeciwnym-razie).
3. Taki sposób postępowania jest szczególnie przydatny przy operowaniu pętlami, stąd nazwa „bufor pętli”. Jeżeli bufor ten jest dostatecznie pojemny, aby zmieścić wszystkie rozkazy składające się na pętlę, to rozkazy te muszą być pobrane tylko raz, przy pierwszej iteracji. W przypadku następnej iteracji wszystkie potrzebne rozkazy będą się już znajdowały w buforze.

Bufor pętli jest w zasadzie podobny do dedykowanej rozkazom pamięci podręcznej. Różnica polega na tym, że bufor pętli zawiera tylko sekwencje rozkazów i jest o wiele mniejszy, a wobec tego tańszy.

Na rysunku 11.15 jest pokazany przykład bufora pętli. Zawiera on 256 adresowalnych bajtów, zatem do jego indeksowania używa się 8 najmniej znaczących bitów. Pozostałe najbardziej znaczące bity są sprawdzane w celu stwierdzenia, czy cel rozgałęzienia leży wewnątrz wycinka przechwyconego przez bufor.



RYSUNEK 11.15. Bufor pętli

Wśród maszyn stosujących bufor pętli są niektóre maszyny CDC (Star 100, 6600, 7600) oraz CRAY-1. Wspecjalizowana postać bufora pętli jest zaimplementowana w procesorze 68010 firmy Motorola i służy ona do wykonywania 3-rozkazowej pętli obejmującej rozkaz Dbcc (*decrement and branch on condition* - dekrementacja i rozgałęzienie warunkowe - patrz problem 11.8). Bufor ma pojemność 3 słów, a procesor powtarza wykonywanie tych rozkazów aż do spełnienia warunku pętli.

Przewidywanie rozgałęzień

Zastosowano wiele metod przewidywania, czy nastąpi rozgałęzienie. Wśród nich często używane są następujące:

- ❑ przewidywanie zawsze następującego rozgałęzienia;
- a przewidywanie nigdy nie następującego rozgałęzienia;
- a przewidywanie za pomocą kodu operacji;
- a przełącznik nastąpiło/nie nastąpiło;
- a tablica historii rozgałęzień.

Pierwsze trzy rozwiązania są statyczne: nie zależą one od historii poprzedzającej rozkaz rozgałęzienia warunkowego. Ostatnie dwa są dynamiczne: zależą od historii.

Pierwsze dwa rozwiązania są najprostsze. Albo zakłada się zawsze, że rozgałęzienie nie nastąpi i kontynuuje się kolejne pobieranie rozkazów. Albo zakłada się zawsze, że rozgałęzienie nastąpi pobiera się rozkaz z celu rozgałęzienia. Maszyny 68020 i VAX 11/780 stosują przewidywanie nigdy nie następującego rozgałęzienia. VAX 11/780 umożliwia też minimalizowanie efektu złej decyzji. Jeśli pobranie rozkazu po rozgałęzieniu spowoduje błąd strony lub naruszenie ochrony, procesor wstrzymuje wstępne pobieranie, aż do upewnienia się, że rozkaz ten powinien być pobrany.

Badania nad zachowaniem się programów wykazały, że rozgałęzienia warunkowe zabierają ponad 50% czasu [LILJ88], a wobec tego systematyczne po-

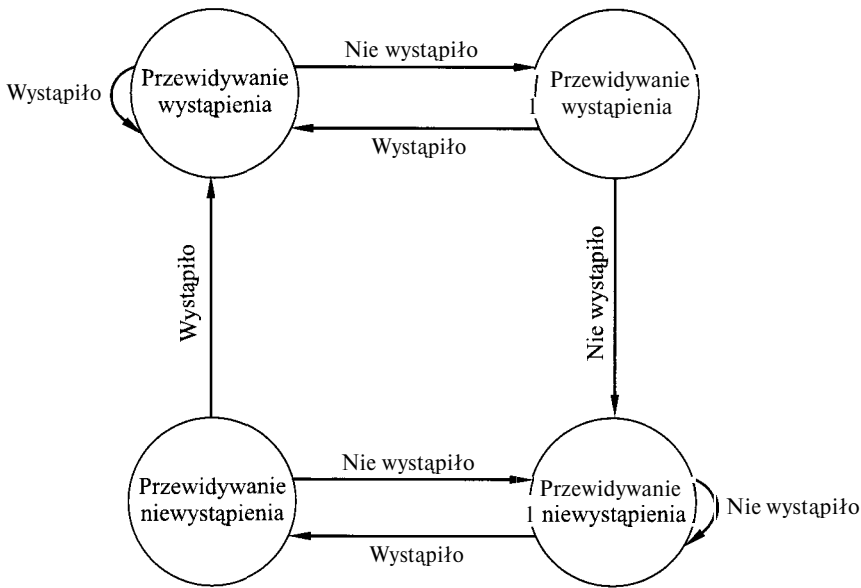
bieranie wstępne docelowego rozkazu rozgałęzienia powinno zapewnić większą wydajność, niż systematyczne pobieranie wstępne ze kolejnych, następujących po sobie rozkazów. Jednak w przypadku maszyn realizujących stronicowanie pobieranie wstępne docelowego rozkazu rozgałęzienia powoduje większe prawdopodobieństwo błędu strony, niż kolejne pobieranie wstępne następnego rozkazu. Powinna więc być wzięta pod uwagę wynikająca stąd strata wydajności. Może być zastosowany mechanizm zapobiegawczy umożliwiający zmniejszenie tej straty.

Ostatnie z rozwiązań statycznych polega na podejmowaniu decyzji na podstawie kodu operacji rozkazu rozgałęzienia. Procesor zakłada, że rozgałęzienie nastąpi w przypadku pewnych kodów operacji, a nie nastąpi w przypadku pozostałych. Zgodnie z [LILJ88], przy tej strategii prawdopodobieństwo sukcesu przekracza 75%.

W strategiach dynamicznych próbuje się poprawić dokładność przewidywania, rejestrując historię rozkazów rozgałęzienia warunkowego w programie. Z każdym rozkazem rozgałęzienia warunkowego można na przykład powiązać jeden lub dwa bity odzwierciedlające najnowszą historię rozkazu. Bity te są określane jako przełącznik nastąpiło/nie nastąpiło, który ukierunkowuje procesor na podjęcie określonej decyzji po ponownym natrafieniu tego rozkazu. Zwykle te bity historii nie są związane z rozkazem w pamięci głównej. Są one raczej przechowywane w tymczasowej, bardzo szybkiej pamięci. Jedną z możliwości jest związanie tych bitów z każdym rozkazem rozgałęzienia warunkowego, który znajduje się w pamięci podręcznej. Gdy rozkaz ten jest zastępowany w pamięci, jego historia jest tracona. Inną możliwością jest utrzymywanie małej tablicy ostatnio wykonywanych rozkazów rozgałęzienia z jednym lub większą liczbą bitów w każdym zapisie. Procesor mógłby sięgać do tej tablicy skojarzeniowo, jak do pamięci podręcznej, lub używając najniższych bitów adresu rozkazu rozgałęzienia.

Wszystko, co można zapisać za pomocą jednego bitu, to wystąpienie lub niewystąpienie rozgałęzienia podczas ostatniego wykonywania rozkazu. Ograniczenie wynikające ze stosowania pojedynczego bitu ujawnia się w przypadku rozkazu rozgałęzienia warunkowego, które następuje prawie zawsze, np. w przypadku rozkazu pętli. Przy zastosowaniu jednego bitu historii, błędne przewidywanie wystąpi dwukrotnie przy każdym użyciu pętli: raz przy wejściu do pętli i raz przy jej opuszczeniu.

Jeśli stosowane są dwa bity, mogą one być użyte do zarejestrowania wyników ostatnich dwóch przypadków wykonania stowarzyszonego rozkazu lub do zapisania stanu w pewien inny sposób. Na rysunku 11.16 jest pokazane typowe rozwiązanie (inne możliwości są opisane w problemie 11.6). Proces decyzyjny może być reprezentowany za pomocą maszyny o skończonej liczbie (czterech) stanów. Jeśli ostatnie dwa rozgałęzienia danego rozkazu nastąpiły w ten sam sposób, to przewiduje się, że należy wybrać tę samą ścieżkę. Jeśli to przewidywanie okazało się błędne, pozostaje ono bez zmian, aż do ponownego napotka-



RYSUNEK 11.16. Graf stanów przewidywania rozgałęzień

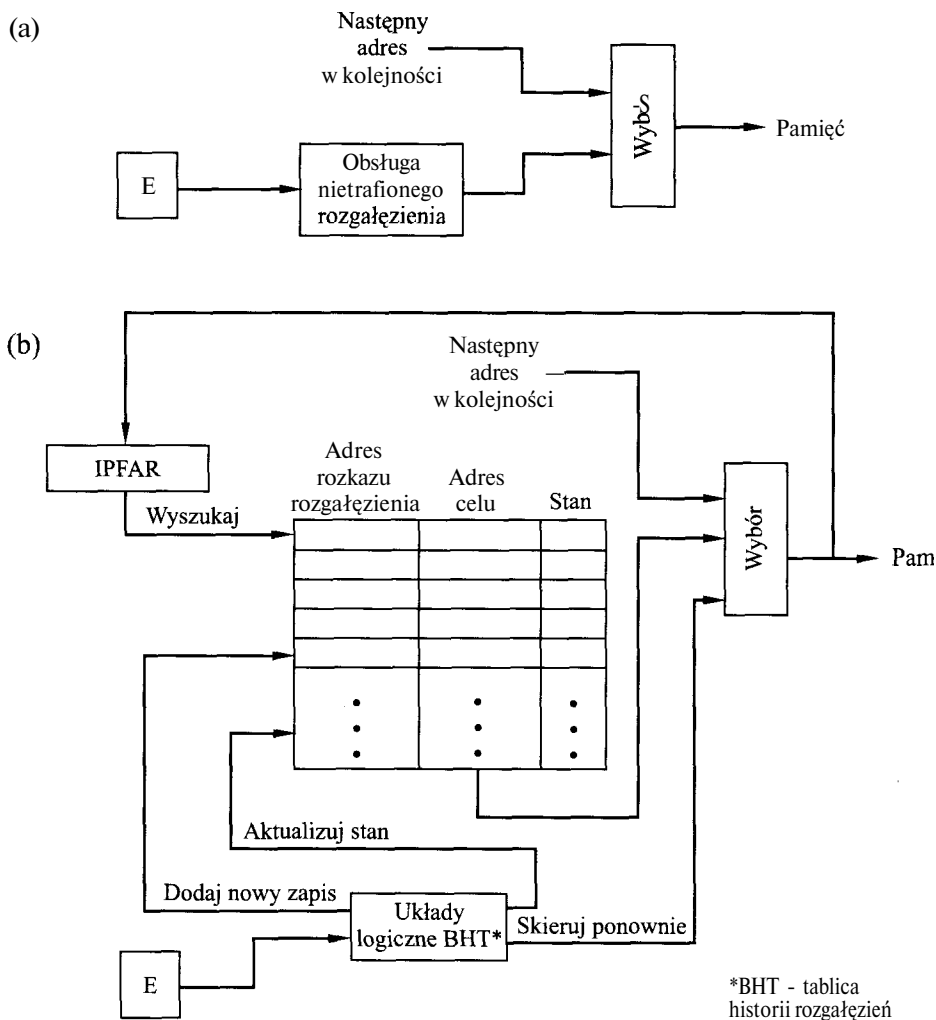
nia tego rozkazu. Jeśli przewidywanie okazało się ponownie błędne, to w wyniku następnego przewidywania zostanie wybrana ścieżka przeciwna. Wobec tego algorytm ten wymaga dwóch kolejnych błędnych przewidywań do zmiany decyzji. Jeśli rozgałęzienie nastąpi w nietypowy sposób tylko raz, na przykład w przypadku pętli, przewidywanie będzie błędne tylko raz.

Przykładem systemu stosującego rozwiązanie z przełącznikiem nastąpiło/nie nastąpiło jest IBM 3090/400.

Opisane właśnie stosowanie bitów historii ma jedną wadę: jeśli zostaje podjęta decyzja, że nastąpi rozgałęzienie, rozkaz docelowy nie może być pobrany, zanim jego adres, który jest argumentem w rozkazie rozgałęzienia warunkowego, nie zostanie zdekodowany. Można by osiągnąć większą efektywność, jeśli pobieranie rozkazu mogłoby być zapoczątkowane tuż po podjęciu decyzji rozgałęzienia. Do tego celu trzeba zachowywać więcej informacji w miejscu nazywanym buforem docelowym rozgałęzienia lub tablicą historii rozgałęzień.

Tablica historii rozgałęzień jest małą pamięcią podręczną powiązaną z etapem pobierania rozkazu w potoku. Każdy zapis w tablicy składa się z trzech elementów: adresu rozkazu rozgałęzienia, pewnej liczby bitów historii, wskazujących na używanie tego rozkazu, oraz informacji o rozkazie docelowym. W większości propozycji i wdrożeń to trzecie pole zawiera adres rozkazu docelowego. Inną możliwością jest to, żeby trzecie pole zawierało sam rozkaz docelowy. Wymienność jest oczywista: przechowywanie adresu prowadzi do mniejszej tablicy, jednak równocześnie do wydłużenia czasu pobierania rozkazu w porównaniu z przechowywaniem rozkazu docelowego.

Na rysunku 11.17 widać odmiennosć tego rozwiązania w porównań z przewidywaniem nigdy nie następującego rozgałęzienia. W ramach tej ostatniej strategii pobierany jest zawsze następny adres w kolejności. Jeśli nastąpi rozgałęzienie, to wykrywają to pewne układy logiczne procesora i przekazują strukturę, że następny rozkaz powinien być pobrany spod adresu docelowego (obok polecenia oczyszczenia potoku). Tablica historii rozgałęzień jest traktowana jak pamięć podręczna. Każde wstępne pobranie powoduje zajrzenie do bazy historii rozgałęzień. Jeśli nie znaleziono zgodności, do pobierania jest używany następny adres w kolejności. Jeśli stwierdzono zgodność, przewidywanie jest oparte na stanie rozkazu: do układów logicznych wyboru jest kierowany albo następny adres w kolejności, albo docelowy adres rozgałęzienia.



RYSunEK 11.17. Postępowanie z rozgałęzieniami: (a) strategia oparta na przewidywaniu, że rozgałęzienie nigdy nie występuje; (b) strategia oparta na tablicy historii rozgałęzień

Gdy rozkaz rozgałęzienia zostanie wykonany, z etapu wykonywania jest kierowany sygnał o wyniku do tablicy historii rozgałęzień. Stan rozkazu jest aktualizowany w celu odzwierciedlenia poprawności lub błędności przewidywania. Jeśli przewidywanie jest błędne, układy logiczne wyboru są ponownie kierowane na poprawny adres następnego pobrania. Gdy napotkany został rozkaz rozgałęzienia warunkowego nie znajdujący się w tablicy, jest on do niej wprowadzany, a jeden z istniejących zapisów jest usuwany za pomocą jednego z algorytmów wymiany zawartości pamięci podręcznej opisanych w rozdziale 4.

Jedynym przykładem wdrożenia tablicy historii rozgałęzień jest mikroprocesor AMD29000 firmy Advanced Micro Devices.

Opóźnione rozgałęzienie

Możliwe jest poprawienie wydajności potoku za pomocą automatycznej zmiany porządku rozkazu wewnątrz programu, tak żeby rozkazy rozgałęzienia występowały później. To intrygujące rozwiązanie jest przeanalizowane w rozdz. 12.

Przetwarzanie potokowe w procesorze Intel 80486

W procesorze 80486 wdrożono 5-etapowy potok o następujących etapach:

- a *Pobieranie*. Rozkazy są pobierane z pamięci podręcznej lub zewnętrznej i umieszczane w jednym z dwóch 16-bajtowych buforów wstępnego pobierania. Celem etapu pobierania jest napełnienie tych buforów nowymi danymi natychmiast po wykorzystaniu starych danych przez dekodery rozkazu. Ponieważ rozkazy mają zmienną długość (od 1 do 11 bajtów, nie licząc przedrostków), stan etapu pobierania w stosunku do innych etapów potoku zmienia się od rozkazu do rozkazu. Przeciętnie jest pobieranych około 5 rozkazów przy ładowaniu każdych 16 bajtów [CRAW90]. Etap pobierania przebiega niezależnie od innych etapów, co ma na celu utrzymywanie pełnych buforów wstępnego pobierania.
- *Etap dekodowania 1 (D1)*. Wszystkie kody operacji oraz informacje o trybie adresowania są dekodowane podczas etapu D1. Wymagane informacje, również dotyczące długości rozkazu, są zawarte w co najwyżej 3 pierwszych bajtach rozkazu. Wobec tego właśnie 3 bajty są przenoszone z buforów wstępnego pobierania w czasie etapu D1. Dekoder D1 może następnie sprawić, że na etapie D2 nastąpi przechwycenie pozostałej części rozkazu (przesunięcia i danych natychmiastowych), która nie była angażowana w dekodowanie prowadzone podczas etapu D1.
- *Etap dekodowania 2 (D2)*. Podczas etapu D2 jest poszerzany każdy kod operacji i są tworzone sygnały sterujące ALU. Zachodzi także sterowanie obliczaniem bardziej złożonych trybów adresowania.
- *Wykonywanie (EX)*. Etap ten obejmuje operacje ALU, dostęp do pamięci podręcznej i aktualizację rejestru.

Gdy rozkaz rozgałęzienia zostanie wykonany, z etapu wykonywania jest kierowany sygnał o wyniku do tablicy historii rozgałęzień. Stan rozkazu jest aktualizowany w celu odzwierciedlenia poprawności lub błędności przewidywania. Jeśli przewidywanie jest błędne, układy logiczne wyboru są ponownie kierowane na poprawny adres następnego pobrania. Gdy napotkany został *rozkaz* rozgałęzienia warunkowego nie znajdujący się w tablicy, jest on do niej wprowadzany, a jeden z istniejących zapisów jest usuwany za pomocą jednego z algorytmów wymiany zawartości pamięci podręcznej opisanych w rozdziale 4.

Jedynym przykładem wdrożenia tablicy historii rozgałęzień jest mikroprocesor AMD29000 firmy Advanced Micro Devices.

Opóźnione rozgałęzienie

Możliwe jest poprawienie wydajności potoku za pomocą automatycznej zmiany porządku rozkazu wewnątrz programu, tak żeby rozkazy rozgałęzienia występowały później. To intrygujące rozwiązanie jest przeanalizowane w rozdz. 12.

Przetwarzanie potokowe w procesorze Intel 80486

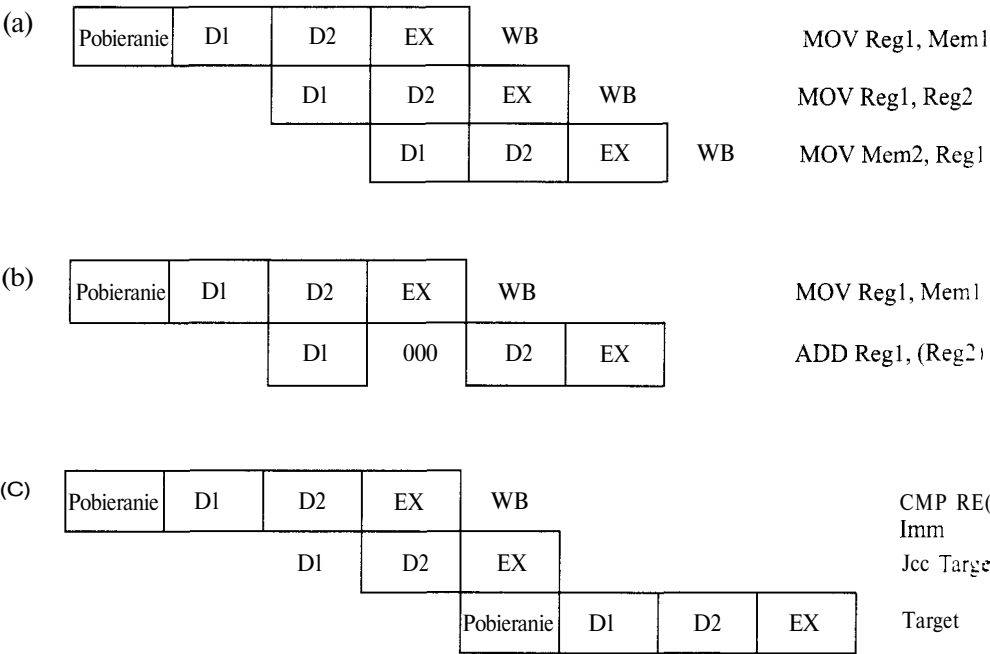
W procesorze 80486 wdrożono 5-etapowy potok o następujących etapach:

- a *Pobieranie*. Rozkazy są pobierane z pamięci podręcznej lub zewnętrznej i umieszczane w jednym z dwóch 16-bajtowych buforów wstępnego pobierania. Celem etapu pobierania jest napełnienie tych buforów nowymi danymi natychmiast po wykorzystaniu starych danych przez dekodery rozkazu. Ponieważ rozkazy mają zmienną długość (od 1 do 11 bajtów, nie licząc przedrostków), stan etapu pobierania w stosunku do innych etapów potoku zmienia się od rozkazu do rozkazu. Przeciętnie jest pobieranych około 5 rozkazów przy ładowaniu każdego 16 bajtów [CRAW90]. Etap pobierania przebiega niezależnie od innych etapów, co ma na celu utrzymywanie pełnych buforów wstępnego pobierania.
- *Etap dekodowania 1 (D1)*. Wszystkie kody operacji oraz informacje o trybie adresowania są dekodowane podczas etapu D1. Wymagane informacje, również dotyczące długości rozkazu, są zawarte w co najwyżej 3 pierwszych bajtach rozkazu. Wobec tego właśnie 3 bajty są przenoszone z buforów wstępnego pobierania w czasie etapu D1. Dekoder D1 może następnie sprawić, że na etapie D2 nastąpi przechwycenie pozostałej części rozkazu (przesunięcia i danych natychmiastowych), która nie była angażowana w dekodowanie prowadzone podczas etapu D1.
- a *Etap dekodowania 2 (D2)*. Podczas etapu D2 jest poszerzany każdy kod operacji i są tworzone sygnały sterujące ALU. Zachodzi także sterowanie obliczaniem bardziej złożonych trybów adresowania.
- a *Wykonywanie (EX)*. Etap ten obejmuje operacje ALU, dostęp do pamięci podręcznej i aktualizację rejestru.

- ❑ *Zapis opóźniony (WB).* Na tym etapie, jeśli jest potrzebny, następuje aktualizacja rejestrów i znaczników stanu zmienionych na poprzednim etapie (wykonywania). Jeśli bieżący rozkaz aktualizuje pamięć, obliczona wartość jest kierowana jednocześnie do pamięci podręcznej i do buforów zapisu i interfejsu magistrali.

Dzięki zastosowaniu dwóch etapów dekodowania potok może utrzymywać przepustowość zbliżoną do jednego rozkazu na cykl zegara. Złożone rozkazy oraz rozkazy rozgałęzienia warunkowego mogą zmniejszyć tę przepustowość.

Na rysunku 11.18 są pokazane przykłady działania potoku. W części a) widać, że dostęp do pamięci nie powoduje opóźnień potoku. Jak jednak wynika z części b), może wystąpić opóźnienie w przypadku wartości używanych do obliczania adresu pamięci. To znaczy, jeśli wartość jest ładowana z pamięci do rejestru, a rejestr ten jest następnie używany jako rejestr podstawowy w następnym rozkazie, to procesor będzie czekał przez jeden cykl. W tym przykładzie procesor sięga do pamięci na etapie EX pierwszego rozkazu i przechowuje pobraną wartość w rejestrze podczas cyklu WB. Jednak następny rozkaz potrzebuje tego rejestru na swoim etapie D2. Gdy etap D2 zrównuje się z etapem WB poprzedniego rozkazu, obojętna ścieżka sygnału pozwala na to, że na etapie D2 będzie miał miejsce dostęp do tych samych danych, co używane podczas zapisu na etapie WB, co pozwala na zaoszczędzenie jednego etapu potoku.



RYSUNEK 11.18. Przykłady potoku rozkazów procesora 80486: (a) brak opóźnienia związanego z ładowaniem danych; (b) opóźnienie związane z ładowaniem wskaźnika; (c) przebieg czasowy rozkazu rozgałęzienia

Na rysunku 11.18c jest pokazane taktowanie rozkazu rozgałęzienia przy założeniu, że następuje rozgałęzienie. Rozkaz porównania aktualizuje kody warunkowe na etapie WB, zaś ścieżki obejściowe udostępniają je jednocześnie na etapie EX rozkazu skoku. Równolegle, podczas etapu EX rozkazu skoku, procesor realizuje spekulatywny cykl pobierania adresu docelowego skoku. Jeśli procesor stwierdzi fałszywość warunku rozgałęzienia, porzuca wstępne pobieranie i kontynuuje pracę z następnym rozkazem sekwencyjnym (już pobranym i zdekodowanym).

11.5

PROCESOR PENTIUM

Ogólny widok organizacji procesora Pentium jest pokazany na rys. 4.24. W tym podrozdziale przeanalizujemy pewne rozwiązania szczegółowe.

Organizacja rejestrów

Organizacja rejestrów obejmuje następujące rodzaje rejestrów (tab. 11.1):

- a **Ogólnego przeznaczenia (robocze).** Występuje osiem 32-bitowych rejestrów ogólnego przeznaczenia (patrz rys. 11.4b). Mogą one być używane przez wszystkie rodzaje rozkazów Pentium; mogą także służyć do przechowywania argumentów do obliczania adresu. Ponadto niektóre z tych rejestrów mogą służyć do celów specjalnych. Na przykład rozkazy łańcuchowe używają zawartości rejestrów ECX, ESI i EDI jako argumentów, bez konieczności jawnego odnoszenia się do tych rejestrów w rozkazie. Dzięki temu wiele rozkazów można zakodować zwięźlej.
- **Segmentowe.** Sześć 16-bitowych rejestrów segmentowych zawiera selektory segmentu służące jako indeksy w tablicach segmentów, co omówiliśmy w rozdz. 7. Rejestr segmentu kodu (CS) odnosi się do segmentu zawierającego wykonywany rozkaz. Rejestr segmentu stosu (SS) odnosi się do segmentu zawierającego stos widzialny dla użytkownika. Pozostałe rejestry segmentowe (DS, ES, FS, GS) umożliwiają użytkownikowi równoczesne odniesienie do czterech oddzielnych segmentów danych.
- **Znaczniki stanu.** Rejestr EFLAGS zawiera kody warunkowe i różne bity trybu.
- **Wskaźnik rozkazu.** Zawiera adres bieżącego rozkazu.

Są też rejestry specjalnie przeznaczone do współpracy z jednostką zmiennopozycyjną:

- a **Numeryczne.** Każdy rejestr zawiera 80-bitową liczbę zmiennopozycyjną o zwiększonej dokładności. Istnieje 8 rejestrów, które działają jako stos,

przy czym w liście rozkazów występują rozkazy umieszczania i zdejmowania.

- **Sterowania.** 16-bitowy rejestr sterowania zawiera bity sterujące pracą jednostki zmiennopozycyjnej, łącznie z zaokrągleniem; bity określające dokładność pojedynczą, podwójną i rozszerzoną oraz bity zezwolenia i blokowania różnych warunków wyjątkowych.
- a **Stanu.** 16-bitowy rejestr stanu zawiera bity odzwierciedlające stan bieżący jednostki zmiennopozycyjnej, włącznie z 3-bitowym wskaźnikiem wierzchołka stosu; kody warunkowe informujące o wyniku ostatniej operacji oraz znaczniki stanu wyjątku.
- a **Słowo wyróżników.** Rejestr 16-bitowy zawiera 2-bitowy wyróżnik dla każdego numerycznego rejestru zmiennopozycyjnego, który wskazuje rodzaj zawartości odpowiedniego rejestru. Cztery możliwe wartości obejmują: ważny, zerowy, specjalny (NaN, nieskończoność, zdenormalizowany) i pusty. Wyróżniki te umożliwiają programom sprawdzanie zawartości rejestru numerycznego bez skomplikowanego dekodowania aktualnych danych w rejestrze.

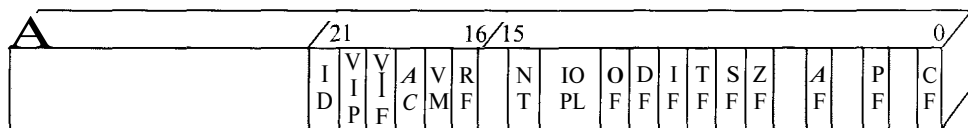
TABELA 11.1. Rejestry procesora Pentium

<i>(a) Jednostka całkowitoliczbowa</i>			
Rodzaj	Liczba	Długość (bitów)	Przeznaczenie
Robocze	8	32	robocze rejestry użytkownika
Segmentowe	6	16	zawierają selektory segmentu
Znacznik stanu	1	32	bity stanu i sterowania
Wskaźnik rozkazu	1	32	wskaźnik rozkazu
<i>(b) Jednostka zmiennopozycyjna</i>			
Rodzaj	Liczba	Długość (bitów)	Przeznaczenie
Numeryczne	8	80	przechowywanie liczb zmiennopozycyjnych
Sterowania	1	16	bity sterowania
Stanu	1	16	bity stanu
Słowo wyróżników	1	16	określa zawartość rejestrów numerycznych
Wskaźnik rozkazu	1	48	wskazuje rozkaz przerwany przez wyjątek
Wskaźnik danych	1	48	wskazuje argument przerwany przez wyjątek

Zastosowanie większości z powyższych rejestrów jest łatwe do zrozumienia. Opiszemy krótko kilka z nich.

Rejestr EFLAGS

Rejestr EFLAGS (rys. 11.19) pokazuje stan procesora i pomaga sterować jego pracą. Obejmuje sześć kodów stanu zdefiniowanych w tabeli 9.8 (przeniesienie, parzystość, pomocniczy, zero, znak, przepełnienie), które informują o wynikach operacji na liczbach całkowitych. Ponadto występują tu bity, które można określić mianem kontrolnych; należą do nich:



ID	–	znacznik stanu identyfikacji	DF	–	znacznik stanu kierunku
VIP	–	zawieszenie wirtualnego przerwania	IF	–	znacznik stanu zezwolenia przerwania
VIF	–	znacznik wirtualnego przerwania	TF	–	znacznik stanu pułapki
AC	–	sprawdzenie wyrównania	ZF	–	znacznik stanu zera
VM	–	tryb wirtualnego 8086	AF	–	pomocniczy znacznik stanu przeniesienia
RF	–	znacznik stanu	PF	–	znacznik stanu parzystości
NT	–	znacznik stanu zadania zagnieżdżonego	CF	–	znacznik stanu przeniesienia
IOPL	–	poziom uprzywilejowania wejścia-wyjścia			
OF	–	znacznik stanu przepełnienia			

RYSUNEK 11.19. Rejestr EFLAGS procesora Pentium

- ❑ **Znacznik pułapki (TF).** Gdy jest ustawiony, powoduje przerwanie po wykonaniu każdego rozkazu. Jest używany przy usuwaniu błędów z programów.
- a **Znacznik zezwolenia przerwania (IF).** Gdy jest ustawiony, procesor rozpoznaje przerwanie zewnętrzne.
- a **Znacznik kierunku (DF).** Wskazuje, czy rozkazy przetwarzania łańcuchowego inkrementują, czy dekrementują stan półrejestrów 16-bitowych SI i DI (w przypadku operacji 16-bitowych) lub rejestrów 32-bitowych ESI i EDI (w przypadku operacji 32-bitowych).
- Q **Znacznik uprzywilejowania wejścia-wyjścia (IOPL).** Gdy jest ustawiony, powoduje, że procesor generuje sygnał wyjątku w stosunku do wszystkich dostępu do urządzeń wejścia-wyjścia podczas funkcjonowania w trybie chronionym.
- a **Znacznik wznowienia (RF).** Umożliwia programiście zablokowanie wyjątków związanych z usuwaniem błędów, co pozwala na wznowienie rozkazu bez natychmiastowego powodowania następnego wyjątku.
- a **Kontrola wyrównania (AC).** Jest wzbudzany, jeśli słowo lub podwójne słowo jest adresowane w nieodpowiednich granicach.
- Q **Znacznik identyfikacji (ID).** Bit ten może być ustawiony i wyzerowany, jeśli procesor może wykonywać rozkaz CPUID. Rozkaz ten dostarcza informacji o sprzedawcy, rodzinie maszyn i o modelu.

Ponadto występują 4 bity związane z trybem operacji. Znacznik stanu zadania zagnieżdżonego (NT) wskazuje, że bieżące zadanie jest zagnieżdżone wewnątrz innego zadania w trybie chronionym. Bit trybu wirtualnego (VM) umożliwia programiście zezwolenie lub zablokowanie trybu wirtualnego 8086, w którym procesor pracuje jak 8086. Znacznik stanu przerwania wirtualnego (VIF) oraz znacznik stanu zawieszenia przerwania wirtualnego (VIP) są używane w środowisku wielozadaniowym.

Rejestry sterowania

Pentium wykorzystuje cztery 32-bitowe rejestry sterowania (rejestr CR1 jest nieużywany) w celu sterowania różnymi aspektami pracy procesora (rys. 11.20). Rejestr CRO zawiera znaczniki stanu sterowania systemowego, które sterują trybami lub wskazują stany odnoszące się raczej do procesora niż do wykonywania indywidualnego zadania. Znaczniki te są następujące:

- Q Zezwolenie ochrony (PE).** Zezwala lub blokuje tryb pracy chronionej.
- a Monitorowanie koprocatora (MP).** Istotny tylko przy realizacji programów z dawniejszych maszyn Pentium; odnosi się do obecności koprocatora arytmetycznego.
- a Emulacja (EM).** Ustawiany, gdy procesor nie ma jednostki zmiennopozycyjnej. Powoduje przerwanie, gdy czyniona jest próba wykonania rozkazów zmiennopozycyjnych.
- a Przełączone zadanie (TS).** Wskazuje, że procesor przełączył zadania.
- ☐ **Rodzaj rozszerzenia (ET).** Nie używany w Pentium; wskazuje wspieranie koprocatora arytmetycznego we wcześniejszych maszynach.

31											11											6	5	4	3	2	1	0																																		
																						M	C	E	P	S	E	D	E	T	S	D	P	V	I	M	E	CR4																								
Baza katalogu stron																						P	C	D	P	W	T										CR3																									
Liniowy adres błędu strony																																																														CR2
																																																														CR1
P	C	N									A	M	W	P									N	E	T	S	E	M	P	E							CR0																									
31	30	29									18		16									5	4	3	2	1	0																																			

MCE - zezwolenie sprawdzenia komputera
 PSE - rozszerzenia rozmiaru strony
 DE - rozszerzenia uruchamiania (debugging)
 TSD - blokada znaku czasowego
 PVI - przerwanie wirtualne w trybie chronionym
 VME - rozszerzenia trybu wirtualnego 8086
 PCD - blokada pamięci podręcznej na poziomie strony
 PWT - przejrzystość zapisów na poziomie strony
 PG - stronicowanie
 CD - blokada pamięci podręcznej

NW - niestosowanie zapisu jednoczesnego
 AM - maska wyrównania
 WP - ochrona zapisu
 NE - błąd numeryczny
 ET - rodzaj rozszerzenia
 TS - przełączenie zadania
 EM - emulacja
 MP - koprocator kontrolny
 PE - zezwolenie ochrony

Przetwarzanie przerwań

Przetwarzanie przerwań wewnątrz procesora jest udogodnieniem przewidzianym do wspierania systemu operacyjnego. Umożliwia ono zawieszenie programu użytkowego w celu obsłużenia różnych warunków przerwań oraz późniejsze wznowienie programu.

Przerwania i wyjątki

Dwie klasy zdarzeń skłaniają Pentium do zawieszenia wykonywania bieżącego strumienia rozkazów i do reakcji na te zdarzenia: przerwania i wyjątki. W obu przypadkach procesor zachowuje kontekst bieżącego procesu i przechodzi do wcześniej określonej procedury związanej z obsługą warunku. **Przerwanie** jest generowane przez sygnał pochodzenia sprzętowego i może nastąpić w dowolnym czasie podczas realizacji programu. **Wyjątek** jest generowany przez oprogramowanie i jest spowodowany wykonywaniem rozkazu. Istnieją dwa źródła przerwań i dwa źródła wyjątków:

1. Przerwania:

- Q Maskowane.** Odebrane poprzez końcówkę INTR procesora. Procesor nie rozpoznaje przerwania maskowanego, chyba że jest ustawiony znacznik zezwolenia przerwania (IF).
- a Niemaskowane.** Odebrane poprzez końcówkę NMI procesora. Rozpoznaniu takich przerwań nie można zapobiec.

2. Wyjątki:

- a Wykryte przez procesor.** Są wynikiem napotkania błędu przez procesor usiłujący wykonać rozkaz.
- ☐ **Programowane.** Istnieją rozkazy generujące wyjątki (INT0, INT3, INT i BOUND).

Tablica wektorów przerwań

Przetwarzanie przerwań w Pentium opiera się na używaniu tablicy wektorów przerwań. Każdemu rodzajowi przerwania jest przypisany numer, służący do indeksowania w tablicy wektorów przerwań. Tablica ta zawiera 256 32-bitowych wektorów przerwań, które są adresami (segment i wyrównanie) procedur obsługi przerwania dla określonego numeru przerwania. W tabeli 11.2 jest podane przypisanie numerów w tablicy wektorów przerwań; zapisy zacieniowane dotyczą przerwań, a niezacieniowane – wyjątków. Przerwanie sprzętowe NMI jest rodzaju 2. Przerwaniom sprzętowym INTR przypisano numery od 32 do 255. Gdy generowane jest przerwanie INTR, na magistrali musi mu towarzyszyć numer wektora przerwania odnoszący się do tego przerwania. Pozostałe numery wektorów dotyczą wyjątków.

TABELA 11.2. Tabela wektorów wyjątków i przerwań procesora Pentium

Numer wektora	Opis
0	Błąd dzielenia; przepełnienie dzielenia lub dzielenie przez zero.
1	Wyjątek związany z usuwaniem błędów; obejmuje różne błędy i pułapki związane z usuwaniem błędów (<i>debugging</i>).
2	Przerwanie za pośrednictwem końcówki NMI; sygnał na końcówce NMI.
3	Punkt przerywania; spowodowany przez rozkaz INT 3, będący 1-bajtowym rozkazem używanym przy uruchamianiu.
4	Przepełnienie wykryte przez INTO; następuje, gdy procesor wykonuje INTO przy ustawionym znaczniku stanu OF.
5	Przekroczenie zakresu BOUND; rozkaz BOUND powoduje porównanie zawartości rejestru z granicami przechowywanymi w pamięci i wygenerowanie przerwania, jeśli zawartość rejestru wykracza poza granice.
6	Nieokreślony kod operacji.
7	Urządzenie nieosiągalne; próba użycia rozkazu ESC lub WAIT nie powiodła się z powodu braku urządzenia zewnętrznego.
8	Podwójny błąd; podczas tego samego rozkazu następują dwa przerwania, które nie mogą być obsługiwane szeregowo.
9	Zarezerwowany.
10	Nieważny segment stanu zadania; segment opisujący żądane zadanie nie został zainicjowany lub jest nieważny.
11	Segment nieobecny; wymagany segment nie jest obecny.
12	Błąd stosu; przekroczenie wartości granicznej segmentu stosu lub segment stosu nie jest obecny.
13	Ogólna ochrona; naruszenie ochrony, które nie powoduje innych wyjątków (np. zapisywanie w segmencie przeznaczonym tylko do odczytu).
14	Błąd strony.
15	Zarezerwowany.
16	Błąd zmiennopozycyjny; generowany przez rozkaz arytmetyki zmiennopozycyjnej.
17	Sprawdzenie wyrównania; dostęp do słowa przechowywanego pod nieparzystym adresem bajtowym lub do podwójnego słowa, którego adres nie jest wielokrotnością 4.
18	Sprawdzenie komputera; zależny od modelu.
19+31	Zarezerwowane.
32+255	Wektory przerwań użytkownika; dostarczane, gdy aktywowany jest sygnał INTR.

Niezacienione - wyjątki, zacienione - przerwania.

Jeśli nie rozstrzygnięte jest więcej niż jedno przerwanie lub wyjątek, procesor obsługuje je w przewidywalnej kolejności. Położenie numeru wektora wewnątrz tablicy nie odzwierciedla priorytetu. Priorytety wyjątków i przerwań są zorganizowane w postaci pięciu klas. Oto one, w kolejności malejącego priorytetu:

Q Klasa 1. Pułapka w poprzednim rozkazie (wektor nr 1).

a Klasa 2. Przerwania zewnętrzne (2, 32+255).

Q Klasa 3. Błędy pobierania następnego rozkazu (3, 14).

- a **Klasa 4.** Błędy dekodowania następnego rozkazu (6, 7).
- a **Klasa 5.** Błędy wykonywania rozkazu (O, 4, 5, 8, 10÷14, 16, 17).

Obsługa przerwania

Podobnie jak przy przeniesieniu wykonywania za pomocą rozkazu CALL, przy przeniesieniu do procedury obsługi przerwania jest wykorzystywany stos systemowy w celu zachowania stanu procesora. Gdy następuje przerwanie i jest ono rozpoznane przez procesor, ma miejsce następująca sekwencja zdarzeń:

1. Jeśli transfer powoduje zmianę poziomu uprzywilejowania, to rejestr bieżącego segmentu stosu i rejestr bieżącego rozszerzonego wskaźnika stosu (ESP) są umieszczane na stosie.
2. Bieżąca wartość rejestru EFLAGS jest umieszczana na stosie.
3. Znaczniki stanu przerwania (IF) i pułapki (TF) są zerowane. Blokuje to przerwanie INTR i pułapkę lub postępowanie w trybie krokowym.
4. Bieżący wskaźnik segmentu kodu (CS) i bieżący wskaźnik rozkazu (IP lub EIP) są umieszczane na stosie.
5. Jeśli przerwaniu towarzyszy kod błędu, to kod błędu jest umieszczany na stosie.
6. Zawartość wektora przerwania jest pobierana i ładowana do rejestrów CS i IP lub EIP. Wykonywanie rozkazów przez procesor jest kontynuowane począwszy od procedury obsługi przerwania.

W celu wznowienia pracy po przerwaniu w ramach procedury obsługi przerwania wykonywany jest rozkaz IRET. Powoduje to odtworzenie wszystkich wartości umieszczonych na stosie; wykonywanie programu jest wznowiane począwszy od punktu przerwania.

11.6

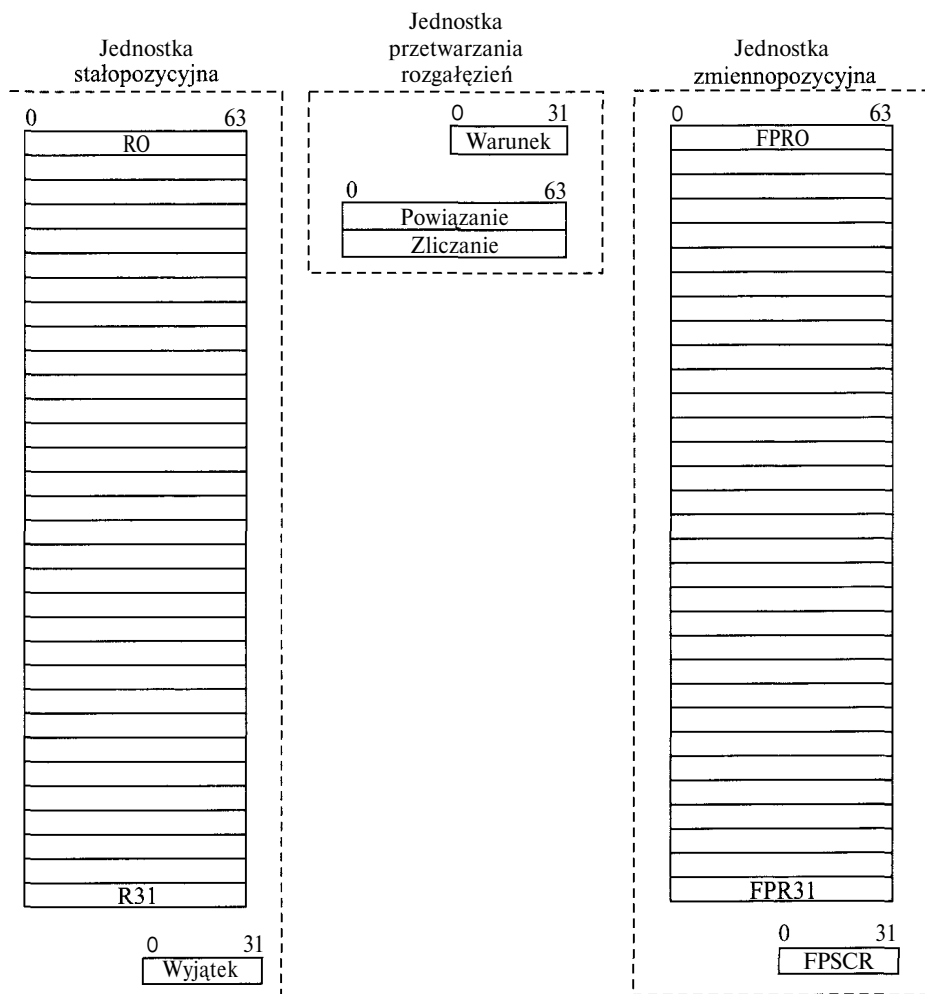
PROCESOR POWERPC

Organizacja procesora PowerPC była przedstawiona na rys. 4.26. W tym podrozdziale przeanalizujemy niektóre rozwiązania szczegółowe dotyczące wersji 64-bitowej.

Organizacja rejestrów

Na rysunku 11.21 są pokazane rejestry PowerPC widzialne dla użytkownika. Jednostka stałopozycyjna zawiera następujące rejestry:

- a **Ogólnego przeznaczenia.** Procesor zawiera 32 64-bitowe rejestry ogólnego przeznaczenia (robocze). Mogą one być używane do ładowania, zapisu



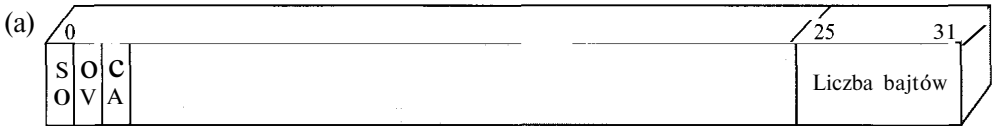
RYSUNEK 11.21. Rejestry procesora PowerPC widzialne dla użytkownika

i manipulowania argumentami oraz jako rejestry adresowania pośredniego. Rejestr O jest traktowany nieco odmiennie. W przypadku operacji ładowania i zapisu oraz niektórych rozkazów dodawania rejestr O jest traktowany jako stale wyzerowany, niezależnie od swojej rzeczywistej zawartości.

- Q Rejestr wyjątku (XER).** Zawiera 3 bity zgłaszania wyjątków w operacjach arytmetycznych na liczbach całkowitych. Zawiera również pole zliczania bajtów używane jako argument w przypadku niektórych rozkazów dotyczących łańcuchów (rys. 11.22a).

Jednostka zmiennopozycyjna zawiera dodatkowe rejestry widzialne dla użytkownika:

- Q Ogólnego przeznaczenia (robocze).** Są to 32 64-bitowe rejestry robocze używane do wszystkich operacji zmiennopozycyjnych.

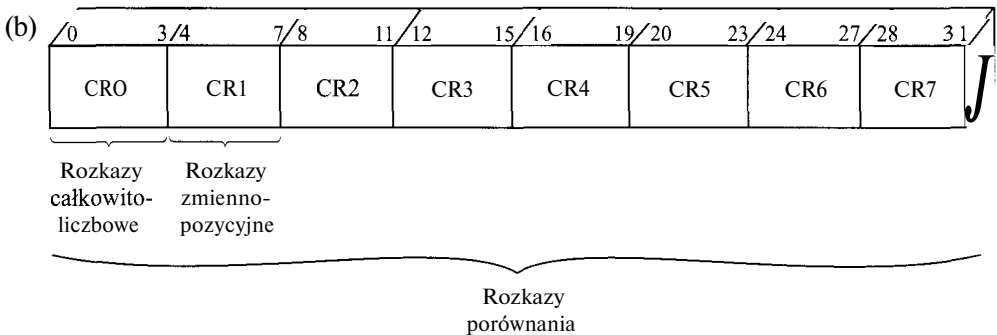


SO - przepełnienie sumaryczne: ustawiany na 1 w celu wskazania przepełnienia występującego podczas wykonywania rozkazu; pozostaje równy 1, aż do wyzerowania programowego

OV - przepełnienie: ustawiany na 1 w celu wskazania przepełnienia podczas wykonywania rozkazu; zerowany przez następny rozkaz, jeśli nie ma przepełnienia

CA - przeniesienie: ustawiany na 1 w celu wskazania przeniesienia na zewnątrz bitu O podczas wykonywania rozkazu

Liczba bajtów - określa liczbę bajtów, które mają być przeniesione przez rozkaz ładowania/zapisu łańcucha



RYSUNEK 11.22. Formaty rejestrów procesora PowerPC: (a) stałopozycyjny rejestr wyjątku (XER); (b) rejestr warunku

- a Zmiennopozycyjny rejestr stanu i sterowania (FPSCR).** Ten 32-bitowy rejestr zawiera bity sterujące pracą jednostki zmiennopozycyjnej oraz bity rejestrujące stan wynikający z operacji zmiennopozycyjnych (tab. 11.3).

Jednostka przetwarzania rozgałęzień zawiera następujące rejestry widzialne dla użytkownika:

- a Rejestr warunku.** Zawiera 8 4-bitowych pól kodu warunkowego (rys. 11.22b).
- a Rejestr powiązania.** Rejestr powiązania może być używany w operacjach rozgałęzienia warunkowego do adresowania pośredniego adresu docelowego. Może być również używany do operacji wywołania i powrotu. Jeżeli jest ustawiony bit LK w rozkazie rozgałęzienia warunkowego, to adres następujący po rozkazie rozgałęzienia jest umieszczany w rejestrze powiązania i może być użyty przy późniejszym powrocie.
- a Rejestr zliczania.** Rejestr zliczania może być używany do sterowania pętlą, co wyjaśniliśmy w rozdz. 9; jego stan jest zmniejszany za każdym razem, gdy jest on sprawdzany w ramach rozkazu rozgałęzienia warunkowego. Innym zastosowaniem tego rejestru jest pośrednie adresowanie adresu docelowego w operacjach rozgałęzienia.

TABELA 11.3. Zmiennopozycyjny rejestr stanu i sterowania procesora PowerPC

Bit	Definicja
0	Podsumowanie wyjątku. Ustawiany, gdy następuje jakikolwiek wyjątek; pozostaje ustawiony, aż do wyzerowania przez program.
1	Podsumowanie dozwolonego wyjątku. Ustawiany, gdy następuje jakikolwiek dozwolony wyjątek.
2	Podsumowanie wyjątku nieważnej operacji. Ustawiany, jeśli nastąpił wyjątek spowodowany przez nieważną operację.
3	Wyjątek przepełnienia. Wielkość wyniku przekracza tę, która może być reprezentowana.
4	Wyjątek niedomiaru. Wynik jest zbyt mały, aby mógł być znormalizowany.
5	Wyjątek dzielenia przez zero. Dzielnik jest zerem, a dzielna jest skończona i niezerowa.
6	Wyjątek niedokładności. Zaokrąglony wynik różni się od wyniku pośredniego lub następuje przepełnienie przy zablokowanym wyjątku przepełnienia.
7÷12	Wyjątek nieważnej operacji: 7 - sygnalizujące NaN; 8 - $(\infty - \infty)$; 9 - $(\infty + \infty)$; 10 - $(0 + 0)$; 11 - $(\infty \times 0)$; 12 - porównanie obejmujące NaN.
13	Zaokrąglenie ułamka. Zaokrąglenie wyniku inkrementowało ułamek.
14	Niedokładność ułamka. Zaokrąglenie wyniku zmieniło ułamek lub wystąpiło przepełnienie przy zablokowanym wyjątku przepełnienia.
15÷19	Znaczniki stanu wyniku. 5-bitowy kod określa: mniejszy niż, większy niż, równy, nie uporządkowany, ciche NaN, $+\infty$, $\pm$ znormalizowany, $\pm$ zdenormalizowany, ± 0 .
20	Zarezerwowany.
21÷23	Wyjątek nieważnej operacji. 21 - zapotrzebowanie programowe; 22 - pierwiastek kwadratowy z liczby ujemnej; 23 - konwersja liczby całkowitej obejmująca wielką liczbę, nieskończoność lub NaN.
24	Zezwolenie wyjątku nieważnej operacji.
25	Zezwolenie wyjątku przepełnienia.
26	Zezwolenie wyjątku niedomiaru.
27	Zezwolenie wyjątku dzielenia przez zero.
28	Zezwolenie wyjątku niedokładności.
29	Tryb niezgodny z IEEE.
30÷31	Sterowanie zaokrągleniem: 2-bitowy kod określa: do najbliższej, w stronę zera, w stronę $+\infty$, w stronę $-\infty$.

Niezacienione - bity stanu; zacienione - bity sterowania.

Pola rejestru warunku mają wiele zastosowań. Pierwsze cztery bity (CRO) są ustawiane w przypadku wszystkich operacji arytmetycznych na liczbach całkowitych, dla których jest ustawiony bit Rc. Jak wynika z tabeli 11.4, pole to wskazuje, czy wynik operacji jest dodatni, ujemny czy zerowy. Czwarty bit jest kopią bitu łącznego przepełnienia z rejestru XER. Następne pole (CR1) jest ustawiane w przypadku wszystkich zmiennopozycyjnych operacji arytmetycznych, dla których jest ustawiony bit Rc. W tym przypadku 4 bity są ustawiane jako równe pierwszym 4 bitom FPSCR (tab. 11.3). Wreszcie 8 pól warunku (CRO÷CR7) może być używanych w przypadku rozkazu porównania; zawartość pola jest określana w samym rozkazie. W odniesieniu do stało- i zmiennopozy-

cyjnych rozkazów porównania, pierwsze 3 bity wyznaczonego pola warunku określają, czy pierwszy argument jest mniejszy niż, większy od lub równy drugiemu argumentowi. Czwarty bit jest bitem łącznego przepełnienia przy porównywaniu stałopozycyjnym, a wskaźnikiem nieuporządkowania przy porównywaniu zmiennopozycyjnym.

TABELA 11.4. Interpretacja bitów w rejestrze warunkowym

Pozycja bitu	CRO (rozkaz całkowitoliczbowy przy $Rc = 1$)	CR1 (rozkaz zmiennopozycyjny przy $Rc = 1$)	CRi (stałopozycyjny rozkaz porównania)	CRi (zmiennopozycyjny rozkaz porównania)
i	wynik < 0	podsumowanie wyjątku	$op1 < op2$	$op1 < op2$
$i + 1$	wynik > 0	podsumowanie dozwolonego wyjątku	$op1 > op2$	$op1 > op2$
$i + 2$	wynik $= 0$	podsumowanie wyjątku nieważnej operacji	$op1 = op2$	$op1 = op2$
$i + 3$	przepełnienie podsumowania	wyjątek przepełnienia	przepełnienie podsumowania	nieuporządkowany (jednym z argumentów jest NaN)

Przetwarzanie przerwań

Jak każdy procesor, PowerPC umożliwia przerwanie przez procesor wykonywania bieżącego programu w celu zajęcia się warunkiem wyjątkowym.

Rodzaje przerwań

Przerwania w PowerPC są podzielone na wywołane przez pewien warunek lub zdarzenie systemowe oraz na spowodowane przez wykonywanie rozkazu. W tabeli 11.5 są wymienione przerwy rozpoznawane przez PowerPC.

Większość przerwań wymienionych w tabeli jest łatwa do zrozumienia. Kilka wymaga dodatkowego komentarza. Systemowe przerwanie ponownego inicjowania pracy zdarza się przy włączaniu zasilania oraz po naciśnięciu przycisku ponownego inicjowania pracy (ang. *reset*). Powoduje ono powtórne zainicjowanie (ang. *reboot*) systemu. Maszynowe przerwanie kontrolne dotyczy pewnych anomalii, takich jak błąd parzystości pamięci podręcznej lub odniesienie do nieistniejącej lokacji pamięci. Może ono spowodować przejście systemu do tzw. stanu zatrzymania kontrolnego (ang. *check-stop state*). W tym stanie zawieszono są operacje procesora i jest zamrożona zawartość rejestrów aż do powtórnego zainicjowania systemu. Przerwanie wspomaganie zmiennopozycyjnego pozwala procesorowi na wywołanie procedur programowych w celu zakończenia operacji, które nie mogą być przeprowadzone przez jednostkę zmiennopozycyjną, w tym operacji na liczbach zdenormalizowanych, lub operacji określanych przez niewdrożone kody operacji zmiennopozycyjnych.

TABELA 11.5. Tabela przerwai PowerPC

Punkt wejścia	Rodzaj przerwania	Opis
00000h	zarezerwowane	
00100h	resetowanie systemu	potwierdzenie twardych lub miękkich sygnałów wejściowych resetowania przez zewnętrzne układy logiczne
00200h	sprawdzenie komputera	potwierdzenie TEA# procesorowi, gdy dozwolone jest rozpoznawanie operacji sprawdzenia komputera
00300h	przechowywanie danych	przykłady: błąd strony danych; naruszenie praw dostępu w czasie ładowania/zapisu
00400h	przechowywanie rozkazów	błąd strony programu; próba pobrania rozkazu z segmentu wejścia-wyjścia; naruszenie praw dostępu
00500h	zewnętrzne	potwierdzenie sygnału wejściowego zewnętrznego przerwania procesora przez zewnętrzne układy logiczne, gdy rozpoznanie zewnętrznego przerwania jest dozwolone
00600h	wyrównanie	nieudana próba dostępu do pamięci spowodowana przez niewyrównany argument
00700h	programowe	przerwanie zmiennopozycyjne; użytkownik próbuje wykonać uprzywilejowany rozkaz; wykonywany jest rozkaz pułapki przy spełnionym określonym warunku; nielegalny rozkaz
00800h	przetwarzanie zmiennopozycyjne nieosiągalne	próba wykonania rozkazu zmiennopozycyjnego przy zablokowanej jednostce zmiennopozycyjnej
00900h	rejestr odejmujący	wyczerpanie możliwości rejestru odejmującego (<i>decrementer register</i>), gdy dozwolone jest rozpoznawanie zewnętrznego przerwania
00A00h	zarezerwowane	
00B00h	zarezerwowane	
00C00h	wywołanie systemowe	wykonywanie rozkazu wywołania systemowego
00D00h	śledzenie	jednoetapowe lub rozgałęzieniowe przerwanie śledzenia
00E00h	wspomaganie zmiennopozycyjne	próba wykonania stosunkowo rzadkiej, złożonej operacji zmiennopozycyjnej (np. operacji na liczbie zdenormalizowanej)
00E10h+ +00FFFh	zarezerwowane	
01000h+ +02FFFh	zarezerwowane (specyficzne wdrożenie)	

Niezacienione - przerwania spowodowane przez wykonywanie rozkazu, zacienione - przerwania nie spowodowane przez wykonywanie rozkazu.

Rejestr stanu maszyny (MSR)

Podstawowe znaczenie przy przerwaniu programu ma zdolność do odtworzenia stanu procesora w czasie zaistnienia przerwania. Dotyczy to nie tylko zawartości różnych rejestrów, ale także różnych warunków sterowania związanych z reali-

TABELA 11.6. Rejestr stanu procesora PowerPC

Bit	Definicja
0	Procesor jest w trybie 32-bitowym/64-bitowym
1+44	Zarezerwowane
45	Zarządzanie mocą dozwolone/zablokowane
46	Zależne od wdrożenia
47	Określa , czy programy obsługi przerwań funkcjonują w trybie najpierw najmłodszy bajt, czy najpierw najstarszy bajt
48	Przerwanie zewnętrzne dozwolone/zablokowane
49	Stan uprzywilejowany/nieuprzywilejowany
50	Jednostka zmiennopozycyjna osiągalna/nieosiągalna
51	Przerwania kontroli komputera dozwolone/zablokowane
52	Tryb O wyjątku zmiennopozycyjnego
53	Siedzenie jednoetapowe dozwolone/zablokowane
54	Siedzenie rozgałęzieniowe dozwolone/zablokowane
55	Tryb I wyjątku zmiennopozycyjnego
56	Zarezerwowane
57	Najbardziej znaczącą częścią adresu wyjątku jest 000h/FFFh
58	Translacja adresu rozkazu włączona/wyłączona
59	Translacja adresu danych włączona/wyłączona
60+61	Zarezerwowane
62	Przerwanie jest przywracalne/nieprzywracalne
63	Procesor jest w trybie najpierw najstarszy bajt/najpierw najmłodszy bajt

Niezacienione - kopiowane do SRR1, zacienione - nie kopiowane do SRR1.

zają programu. Warunki te są zwykle sumowane w MSR (tab. 11.6). Kilka bitów tego rejestru wymaga dodatkowego komentarza.

Gdy jest ustawiony bit trybu uprzywilejowania (bit 49), procesor pracuje na poziomie uprzywilejowania użytkownika. Dostępny jest tylko pewien podzbiór listy rozkazów. Gdy bit ten jest wyzerowany, procesor pracuje na poziomie uprzywilejowania nadzorcy. Umożliwia to dostęp do wszystkich rozkazów oraz do niektórych rejestrów systemowych (takich jak MSR) niedostępnych z poziomu użytkownika.

Wartości dwóch bitów wyjątku zmiennopozycyjnego (bity 52 i 55) określają rodzaje przerwań, które mogą być generowane przez jednostkę zmiennopozycyjną. Ich interpretacja jest następująca:

FE0	FE1	Przerwania, które zostaną rozpoznane
0	0	żadne
0	1	nieprecyzyjne, nieprzywracalne
1	0	nieprecyzyjne, przywracalne
1	1	precyzyjne

Gdy jest ustawiony bit śledzenia krokowego (bit 53), procesor wraca do śledzącego programu obsługi przerwań po udanym wykonaniu każdego rozkazu. Natomiast gdy jest ustawiony bit śledzenia rozgałęzień (bit 54), procesor wraca do śledzącego programu obsługi przerwań śledzącego po udanym wykonaniu każdego rozkazu rozgałęzienia, niezależnie od tego, czy rozgałęzienie nastąpiło.

Bit translacji adresu rozkazu (bit 58) oraz bit translacji adresu danych (bit 59) określają, czy jest używane rzeczywiste adresowanie, czy też jednostka zarządzania pamięcią prowadzi translację adresu.

Obsługa przerwań

Gdy następuje przerwanie i jest ono rozpoznane przez procesor, ma miejsce następująca sekwencja zdarzeń:

1. Procesor umieszcza adres następnego rozkazu przewidzianego do wykonania w rejestrze zachowania/odtworzenia O (SRRO). Jest to adres właśnie wykonywanego rozkazu, jeśli przerwanie zostało spowodowane przez nieudaną próbę wykonania tego rozkazu; w przeciwnym razie jest to adres następnego rozkazu przewidzianego do wykonania po rozkazie bieżącym.
2. Procesor kopiuje informację o stanie z rejestru MSR do rejestru zachowania/odtworzenia I (SRR1). Kopiowane są bity, które w tabeli 11.6 nie są zacieniowane. Do pozostałych bitów SRR1 jest ładowana informacja specyficzna dla rodzaju przerwania.
3. W rejestrze MSR jest ustawiana wartość określona sprzętowo, specyficzna dla rodzaju przerwania. Dla wszystkich rodzajów przerwań translacja adresu jest wyłączona, a przerwania zewnętrzne są zablokowane.
4. Następnie procesor przekazuje sterowanie odpowiedniemu programowi obsługi przerwań. Adresy programów obsługi przerwań są przechowywane w tablicy przerwań (tab. 11.5). Adres podstawowy tej tablicy jest wyznaczany przez bit 57 w rejestrze MSR.

Aby wrócić po przerwaniu, w ramach procedury obsługi przerwań jest wykonywany rozkaz rfi. Powoduje on, że bity zachowane w rejestrze SRR1 są odtwarzane w rejestrze MSR. Wykonywanie jest wznawiane począwszy od lokacji przechowywanej w SRRO.

11.7

POLECANA LITERATURA

Prace [HENN91] i [HAYE88] zawierają szczegółowy opis przetwarzania potokowego. W [SOHI90] można znaleźć doskonałe, szczegółowe omówienie problemów projektowania sprzętu związanych z potokowym przetwarzaniem rozkazów. W [CRAG92] jest szczegółowe studium przewidywania rozgałęzień w potokach rozkazów. W [DUBE91] i [LILJ88] są przeanalizowane różne stra-

tęgie przewidywania rozgałęzień, które mogą być używane w celu zwiększania wydajności potokowego przetwarzania rozkazów. W [KAEL91] są przeanalizowane utrudnienia w przewidywaniu rozgałęzień powodowane przez rozkazy, których adres docelowy jest zmienny. Potokowe przetwarzanie rozkazów w procesorze Intel 80486 jest opisane w [TABA91].

W [BREY95] dobrze opisano przetwarzanie przerwań w Pentium, a w [SHAN94b] - w PowerPC.

BREY95 Brey B.: *The Intel 32-bit Microprocessors: 80386, 80486 i Pentium*. Englewood Cliffs, Prentice Hall, 1995.

CRA92 Cragon H.: *Branch Strategy Taxonomy and Performance Models*. Los Alamitos, IEEE Computer Society Press, 1992.

DUBE91 Dubey P., Flynn M.: „Branch Strategies: Modeling and Optimization”. *IEEE Trans. on Computers*, October 1991.

HAYE88 Hayes J.: *Computer Architecture and Organization*. Wyd. II. New York, McGraw-Hill, 1998.

HENN91 Hennessy J., Jouppi N.: „Computer Technology and Architecture: an Evolving Interaction”. *Computer*, September 1991.

KAEL91 Kaeli D., Emma P.: „Branch History Table Prediction of Moving Target Branches Due to Subroutine Returns”. *Proceeding, 18th Annual International Symposium on Computer Architecture*, May 1991.

LILJ88 Lilja D.: „Reducing the Branch Penalty in Pipelined Processors”. *Computer*, July 1988.

SHAN94b Shanley T.: *PowerPC 601 System Architecture*. Richardson, Mindshare Press, 1994.

SOHI90 Sohi G.: „Instruction Issue Logic for High-Performance Interruptable, Multiple Functional Unit, Pipelined Computers”. *IEEE Trans. on Computers*, March 1990.

TABA91 Tabak D.: *Advanced Microprocessors*. New York, McGraw-Hill, 1991.

11.8

PROBLEMY DO ROZWIĄZANIA

11.1. (a) Jeśli ostatnią operacją przeprowadzoną w komputerze o słowie 8-bitowym było dodawanie, w którym dwoma argumentami były 2 i 3, to jaka jest wartość następujących znaczników stanu (flag)?

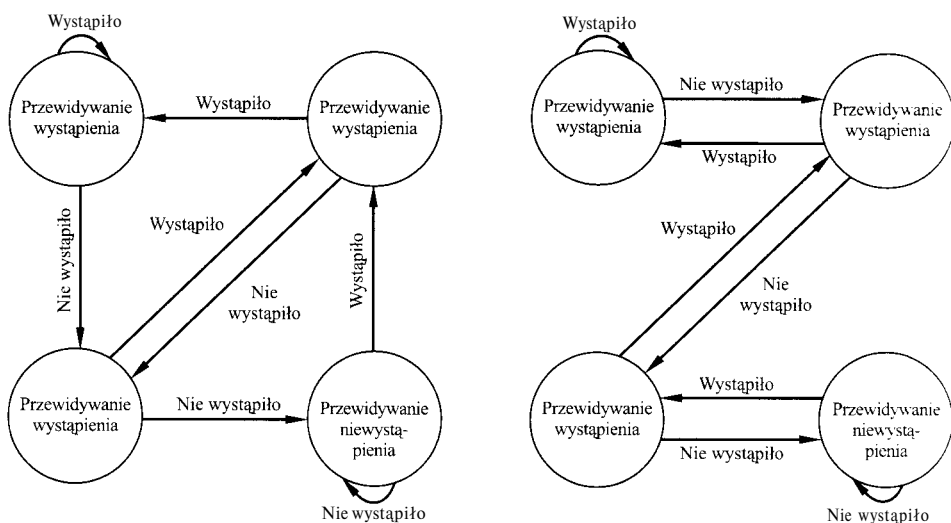
- a przeniesienie,
- a zero,
- a przepełnienie,
- a znak,
- a parzystość,
- a półprzeniesienie.

(b) Jakie byłyby te wartości, jeśli argumentami byłyby -1 (uzupełnienie dwóch) i +1.

11.2. Rozważmy diagram czasowy na rys. 11.12. Załóżmy, że mamy do czynienia z potokiem dwuetapowym (pobieranie, wykonanie). Narysuj nowy

diagram w celu wykazania, ile jednostek czasu potrzeba do wykonania 4 rozkazów.

- 11.3. Jednym z ograniczeń rozwiązania wielostrumieniowego obsługi rozgałęzień w potoku jest to, że napotykane są dodatkowe rozgałęzienia, zanim pierwsze rozgałęzienie jest wykonane do końca. Zasygnalizuj dwa dodatkowe ograniczenia lub niedogodności.
- 11.4. Jakie własności powinny być przewidziane w projekcie kodów operacyjnych nowego systemu w celu umożliwienia przyszłego wdrożenia oddzielnych układów równoległych z przetwarzaniem potokowym rozkazów zmiennopozycyjnych, stałopozycyjnych oraz dotyczących łańcuchów bitowych i bajtowych?
- 11.5. Rozważ wykresy stanów z rys. 11.23.
- Opisz funkcjonowanie każdego z nich.
 - Porównaj je z wykresem stanów odnoszącym się do przewidywania rozgałęzień w podrozdz. 11.4. Przedyskutuj mocne i słabe strony każdego z tych trzech rozwiązań przewidywania rozgałęzień.



RYSUNEK 11.23. Graf stanów przetwarzania rozgałęzień

- 11.6. Procesory Motorola 680x0 zawierają rozkaz dekrementuj i rozgałęziaj zależnie od warunku (ang. *Decrement and Branch According to Condition*), który ma następującą postać:

Dbcc Dn, displacement

gdzie cc jest jednym z warunków testowalnych, Dn jest rejestrem roboczym, a przesunięcie (*displacement*) określa adres docelowy w stosunku do adresu bieżącego. Rozkaz ten może być zdefiniowany następująco:

```

if (cc = False)
then begin
    Dn := (Dn) - 1;
    if Dn  $\neq$  -1 then PC := (PC) + displacement end
else PC := (PC) + 2;

```

Gdy wykonywany jest ten rozkaz, najpierw jest sprawdzany warunek w celu stwierdzenia, czy warunek zakończenia pętli jest spełniony. Jeśli tak, to nie jest przeprowadzana żadna operacja, natomiast jest wykonywany następny rozkaz w kolejności. Jeśli warunek jest fałszywy, to zawartość ustalonego rejestru danych jest zmniejszana i sprawdzana, czy nie jest mniejsza od zera. Jeśli jest mniejsza od zera, pętla jest kończona i jest wykonywany następny rozkaz w kolejności. W przeciwnym razie program rozgałęzia się do ustalonej lokacji.

Rozważmy teraz następujący fragment programu w języku asemblerowym:

```

AGAIN      CPMPL      (A0)+,(A1)+
           DBNE        D1,AGAIN
           NOP

```

Porównywane są dwa łańcuchy adresowane za pomocą AO i A1; wskaźniki łańcuchów są zmniejszane przy każdym odniesieniu. D1 początkowo zawiera wartość określającą liczbę długich słów (4 bajty), które mają być porównane.

- (a) Początkowe zawartości rejestrów są: AO = \$00004000, A1 = \$00005000 i D1 = \$000000FF (znak \$ oznacza notację szesnastkową). Pamięć między \$4000 a \$6000 jest załadowana słowami \$AAAA. Jeśli powyższy program jest uruchomiony, określ, ile razy jest wykonywana pętla DBNE oraz podaj zawartości trzech rejestrów, gdy osiągnięty jest rozkaz NOP.
- (b) Powtórz (a), jednak tym razem załóż, że pamięć między \$4000 a \$4FEE jest załadowana słowami \$AAA.

11.7. Narysuj ponownie rys. 11.18c zakładając, że rozgałęzienie warunkowe nie następuje.

Komputery o zredukowanej liście rozkazów

Od czasu powstania komputera z przechowywanym programem około roku 1950, niewiele było znaczących, prawdziwych innowacji w dziedzinie organizacji i architektury komputerów. Następujące innowacje, chociaż nie tworzące kompletnej listy, stanowią główne osiągnięcia od narodzin komputera:

- ❑ *Koncepcja rodziny.* Wprowadzona przez IBM wraz z systemem 360 w roku 1964, powtórzona wkrótce potem przez DEC wraz z PDP8. Koncepcja rodziny zakłada oddzielenie architektury maszyny od jej implementacji. Oferowany jest zbiór komputerów różniących się stosunkiem ceny do wydajności, prezentujących użytkownikowi tę samą architekturę. Różnice ceny i wydajności wynikają z różnych wdrożeń tej samej architektury.
- a *Mikroprogramowana jednostka sterująca.* Zasugerowana przez Wilkesa w roku 1951 i wprowadzona przez IBM w linii 360 w roku 1964. Mikroprogramowanie ułatwia zadanie zaprojektowania i implementacji jednostki sterującej i wspiera koncepcję rodziny.
- ❑ *Pamięć podręczna.* Po raz pierwszy wprowadzona na rynek w modelu 85 IBM S/360 w roku 1968. Dołączenie tego elementu do hierarchii pamięci w istotny sposób poprawia wydajność.
- a *Przetwarzanie potokowe.* Sposób wprowadzenia równoległości do w zasadzie sekwencyjnej natury programów złożonych z rozkazów maszynowych. Przykładami są przetwarzanie potokowe rozkazów i przetwarzanie wektorowe.
- a *Wieloprocesorowość.* Kategoria ta obejmuje wiele różnych organizacji i celów.

Do tej listy musimy teraz dodać jedną z najbardziej interesujących i (potencjalnie) najważniejszych innowacji: architekturę komputerów o zredukowanej li-

ście rozkazów (RISC). Architektura RISC stanowi drastyczne odchylenie od historycznej tendencji rozwoju architektury procesorów; jest też wyzwaniem wobec konwencjonalnej mądrości wyrażanej w czynach i w słowach przez większość specjalistów od architektury komputerów. Analiza architektury RISC skłania do zastanowienia się nad najważniejszymi problemami organizacji i architektury komputerów.

Chociaż systemy RISC były definiowane i projektowane na różne sposoby przez różne grupy, kluczowymi elementami występującymi w większości projektów (choć nie we wszystkich) są:

- ograniczony i prosty zbiór rozkazów;
- a duża liczba rejestrów roboczych lub zastosowanie kompilatorów do optymalizacji wykorzystania rejestrów;
- a akcent na optymalizację potoku rozkazów.

W tabeli 12.1 znajduje się porównanie kilku systemów RISC oraz nie-RISC.

TABELA 12.1. Własności niektórych procesorów CISC, RISC i superskalarnych

Własność	Komputery o złożonej liście rozkazów (CISC)			Komputery o zredukowanej liście rozkazów (RISC)		Superskalarne	
	IBM 370/168	VAX 11/780	Intel 80486	Motorola 88000	MIPS R4000	IBM system RS 6000	Intel 80960
Rok powstania	1973	1978	1989	1988	1991	1990	1989
Liczba rozkazów	208	303	235	51	94	184	62
Rozmiar rozkazu [B]	2-6	2÷57	1÷11	4	32	4	4, 8
Tryby adresowania	4	22	11	3	1	2	11
Liczba rejestrów roboczych	16	16	8	32	32	32	23-256
Rozmiar pamięci sterującej [Kbit]	420	480	246	–	–	–	–
Rozmiar pamięci podręcznej [KB]	64	64	8	16	128	32-64	0,5

Rozpocniemy ten rozdział od krótkiego przeglądu pewnych wyników prac nad listami rozkazów, następnie przeanalizujemy każdy z wyżej wymienionych trzech elementów. W dalszym ciągu opiszemy dwa najlepiej udokumentowane projekty RISC.

12.1

WŁASNOŚCI WYKONYWANIA ROZKAZÓW

Jednym z najbardziej widocznych elementów ewolucji związanej z komputerami są języki programowania. W miarę spadku kosztu sprzętu wzrósł względny koszt oprogramowania. A ciągły brak programistów spowodował bezwzględny

wzrost kosztów oprogramowania. W wyniku tego głównym kosztem w cyklu życia systemu jest oprogramowanie, a nie sprzęt. Poza kosztem, a także niewygoda, istnieje element zawodności: jest powszechne w odniesieniu do programów zarówno systemowych, jak i użytkowych, że ciągle, po latach wykorzystywania, ujawniane są nowe błędy.

Odpowiedzią badaczy i przemysłu było opracowywanie coraz potężniejszych i złożonych języków programowania wysokiego poziomu (porównajmy języki FORTRAN i Ada). Języki wysokiego poziomu umożliwiają programiście bardziej zwarte wyrażanie algorytmów, przejmują część troski o szczegóły i często w naturalny sposób wspierają użycie programowania strukturalnego.

Niestety, rozwiązanie to spowodowało powstanie innego problemu nazwanego luką semantyczną. Chodzi o różnicę między operacjami wyrażonymi w języku wysokiego poziomu a realizowanymi w architekturze komputera. Do symptomów tej luki należą nieefektywność wykonywania, nadmierne rozmiary programów maszynowych oraz złożoność kompilatorów. Odpowiedzią projektantów było opracowanie architektur umożliwiających zmniejszenie tej luki, które oferowały rozbudowane listy rozkazów, dziesiątki trybów adresowania wdrożone sprzętowo i różne dyrektywy języków wysokiego poziomu. Przykładem tej ostatniej możliwości jest rozkaz maszynowy CASE w systemie VAX. Tak złożone listy rozkazów zaprojektowano w celu:

- ułatwienia zadania programistom tworzącym kompilatory;
- a poprawienia efektywności wykonywania, ponieważ złożone sekwencje operacji mogą być wdrożone w postaci mikrorozkazu;
- a wspierania nawet bardziej złożonych i wyrafinowanych języków wysokiego poziomu.

Równocześnie, przez lata prowadzono badania w celu określenia właściwości i wzorów wykonywania rozkazów maszynowych generowanych przez programy pisane w językach wysokiego poziomu. Wyniki tych badań zainspirowały niektórych badaczy do rozważenia całkiem nowego rozwiązania: mianowicie do opracowania architektury, która wspierałaby raczej prostsze niż bardziej złożone języki wysokiego poziomu.

Tak więc, aby móc podążać tokiem rozumowania zwolenników architektury RISC, rozpoczniemy od krótkiego przeglądu właściwości wykonywania rozkazów. Interesującymi aspektami obliczania są:

- *Przeprowadzane operacje.* Określają one działania prowadzone przez procesor i jego współpracę z pamięcią.
- *Używane argumenty.* Rodzaje argumentów i częstość ich używania determinują organizację pamięci służącej do ich przechowywania oraz tryby adresowania.
- Q *Szeregowanie rozkazów.* Określa ono organizację sterowania i przetwarzania potokowego.

W pozostałej części tego podrozdziału podsumujemy wyniki badań nad programami w językach wysokiego poziomu. Wszystkie wyniki są oparte na pomiarach dynamicznych. Oznacza to, że pomiary były gromadzone poprzez wykonywanie programu i liczenie częstości pojawiania się pewnej własności. W przeciwieństwie do tego pomiary statyczne polegają na obliczeniach prowadzonych na tekście źródłowym programu. Nie dają one użytecznej informacji na temat wydajności, ponieważ nie są ważne zależnie od częstości wykonywania każdej instrukcji.

Operacje

Przeprowadzono wiele badań w celu przeanalizowania zachowywania się programów w językach wysokiego poziomu. W tabeli 4.8 zawarto główne wyniki niektórych badań. Badania te dotyczyły pewnego zbioru języków i zastosowań i wykazały dobrą zgodność. Dominowały instrukcje przypisania, co sugeruje ważność prostego przemieszczania danych. Wystąpiła też przewaga instrukcji warunkowych (IF, LOOP). Instrukcje te są wdrażane w języku maszynowym za pomocą rozkazów porównywania i rozgałęzienia. Sugeruje to ważność mechanizmu sterowania szeregowaniem rozkazów.

Wyniki te są bardzo pouczające dla projektanta list rozkazów maszynowych. wskazują bowiem, jakie rodzaje instrukcji występują najczęściej i wobec tego powinny być optymalnie realizowane. Jednakże wyniki te nie ujawniają, które instrukcje zajmują najwięcej czasu przy wykonywaniu typowego programu. Przy danym skompilowanym programie w języku maszynowym chodzi o to, które instrukcje w języku źródłowym powodują wykonywanie większości rozkazów w języku maszynowym.

W celu zbadania tego podstawowego zjawiska skompilowano programy Pattersona [PATT82a] na maszynach VAX, PDP-11 i Motorola 68000, aby wyznaczyć średnią liczbę rozkazów maszynowych i odnieść do pamięci przypadającą na określony rodzaj instrukcji. Mnożąc częstość występowania każdego rodzaju instrukcji przez te średnie, otrzymano tabelę 12.2. Kolumny 2 i 3 zawierają zastępcze miary rzeczywistego czasu wykonywania różnych rodzajów instrukcji. Wyniki sugerują, że najbardziej czasochłonnymi operacjami w typowych programach w językach wysokiego poziomu są procedury wywołania i powrotu.

Znaczenie tabeli 12.2 powinno być dla czytelnika jasne. Pokazuje ona względne znaczenie różnych rodzajów instrukcji w języku wysokiego poziomu, gdy język ten jest skompilowany dla typowej, współczesnej architektury listy rozkazów. W przypadku odniesienia do innej architektury wyniki zapewne byłyby inne. Jednak badania te doprowadziły do rezultatów, które są reprezentatywne dla współczesnych architektur komputerów o złożonej liście rozkazów (CISC). Mogą więc stanowić przewodnik dla tych, którzy szukają efektywniejszych dróg wspierania języków wysokiego poziomu.

TABELA 12.2. Wazona względna częstotliwość operacji w językach wysokiego poziomu [PATT82a]

	Występowanie dynamiczne		Wazona według rozkazów maszynowych		Wazona według odniesień do pamięci	
	Pascal	C	Pascal	C	Pascal	C
ASSIGN	45	38	13	13	14	15
LOOP	5	3	42	32	33	26
CALL	15	12	31	33	44	45
IF	29	43	11	21	7	13
GOTO	—	3	—	—	—	—
W innych przypadkach	6	1	3	1	2	1

Argumenty

O wiele mniej prac poświęcono występowaniu różnych rodzajów argumentów, mimo ważności tego problemu. Występuje tu kilka znaczących aspektów.

Wspomniane już badania Pattersona [PATT82a] obejmowały również dynamiczną częstość występowania klas zmiennych (tabela 12.3). Wyniki, zgodne między programami w Pascalu i C, wykazały, że większość odniesień dotyczyła prostych zmiennych skalarnych. Ponad 80% tych skalarów to zmienne lokalne (w stosunku do procedury). Ponadto odniesienia do tablic i struktur wymagały uprzedniego odniesienia do ich indeksów lub wskaźników, które znów są na ogół lokalnymi skalarami. Występuje wobec tego przewaga odniesień do skalarów, a te z kolei są wysoce zlokalizowane.

TABELA 12.3. Dynamiczny udział procentowy argumentów

	Pascal	C	Średnio
Stałe całkowite	16	23	20
Zmienneskalarne	58	53	55
Tablice/struktury	26	24	25

Badania Pattersona dotyczyły też dynamicznego zachowywania się programów w językach wysokiego poziomu, niezależnie od tła w postaci architektury. Jak stwierdzono poprzednio, aby głębiej przeanalizować zachowywanie się programów, konieczne jest oparcie się na rzeczywistych architekturach. W jednym z badań [LUND77] przeanalizowano dynamicznie rozkazy DEC-10 i stwierdzono, że- przeciętnie każdy rozkaz odnosi się do 0,5 argumentu w pamięci i 1,4 w rejestrach. O podobnych wynikach poinformowano w [HUCK83] w odniesieniu do programów w językach C, Pascal i FORTRAN na S/370, PDP-11 i VAX. Oczywiście liczby te zależą silnie zarówno od architektury, jak i od kompilatora, jednak ilustrują częstość dostępu do argumentów.

Te ostatnie badania sugerują ważność takiej architektury, która jest podporządkowana szybkiemu dostępowi do argumentów, ponieważ ta właśnie opera-

cja jest tak często wykonywana. Z badań Pattersona wynika, że pierwszym kandydatem do optymalizacji jest mechanizm przechowywania i osiągnięcia lokalnych zmiennych skalarnych.

Wywołania procedur

Wiemy już, że wywołania procedur i powroty są ważnym aspektem programów w językach wysokiego poziomu. Udowodniono (tabela 12.2), że są one najbardziej czasochłonnymi operacjami w skompilowanych programach w językach wysokiego poziomu. Powinno więc być korzystne rozważenie sposobów efektywnego wdrażania tych operacji. Dwa aspekty są znaczące: liczba parametrów i zmiennych, które obejmuje dana procedura, oraz głębokość zagnieżdżenia.

W badaniach Tanenbauma [TANE78] zostało stwierdzone, że 98% dynamicznie wywoływanych procedur używa mniej niż 6 argumentów, zaś 92% z nich używa mniej niż 6 lokalnych zmiennych skalarnych. O podobnych wynikach doniósł zespół Berkeley RISC [KATE83], co widać w tabeli 12.4. Wyniki te wskazują, że liczba potrzebnych słów przypadająca na jedno wywołanie procedury nie jest wielka. Badania wspomniane wcześniej wykazały, że znaczną część odniesień do argumentów stanowią odniesienia do lokalnych zmiennych skalarnych. Wykazały one również, że odniesienia te są w rzeczywistości ograniczone do stosunkowo niewielu zmiennych.

TABELA 12.4. Argumenty procedur i lokalne zmienne skalarne

Udział wykonywanych wywołań podprogramów z:	Kompilator, interpreter i składopis	Małe programy nienumeryczne
> 3 argumentami	0*7%	0*5%
> 5 argumentami	0+3%	0%
> 8 słowami argumentów i skalarów lokalnych	1*20%	0*6%
> 12 słowami argumentów i skalarów lokalnych	1*6%	0*3%

Ten sam zespół z Berkeley analizował również wzór wywołań procedur i powrotów w programach w językach wysokiego poziomu. Stwierdził on, że rzadkością są długie, nieprzerwane sekwencje wywołań procedur, po których następują odpowiednie sekwencje powrotów. Program pozostawał raczej ograniczony do wąskiego zakresu głębokości wywoływania procedur. Widać to na rys. 4.31, który był omówiony w rozdziale 4. Wyniki te stanowią potwierdzenie konkluzji, że odniesienia do argumentów są wysoce zlokalizowane.

Wnioski

Wiele zespołów zapoznawało się z takimi wynikami, jak przedstawione powyżej, i dochodziło do wniosku, że dążenie do dostosowania architektury listy rozkazów do języka wysokiego poziomu nie jest najbardziej efektywną strategią

projektowania. Najkorzystniejszym wsparciem języka wysokiego poziomu jest optymalizacja najbardziej czasochłonnych składników typowych programów w tych językach.

Uogólniając wyniki prac wielu badaczy możemy stwierdzić, że architektury RISC można w całości scharakteryzować za pomocą trzech elementów. Po pierwsze, w architekturze RISC występuje duża liczba rejestrów. Ma to na celu zoptymalizowanie odnoszenia się do argumentów. Z przedstawionych wyżej wyników badań można wywnioskować, że na jedną instrukcję w języku wysokiego poziomu przypada kilka odniesień i że występuje znaczny udział instrukcji przenoszenia (przypisania). W połączeniu z lokalnością i z dominacją odniesień skalarnych sugeruje to, że wydajność może być poprawiona przez zredukowanie odniesień do pamięci kosztem większej liczby odniesień do rejestrów. Ze względu na lokalność tych odniesień rozszerzony zestaw rejestrów wydaje się być praktyczny.

Po drugie, w architekturze RISC należy poświęcić wiele uwagi projektowaniu potoków rozkazów. Ze względu na wysoki udział rozkazów rozgałęzienia warunkowego oraz wywołania procedury, prosty potok rozkazów nie będzie efektywny. Uwidacznia się to poprzez wysoki udział rozkazów, które są wstępnie pobierane i nigdy nie wykonywane.

Po trzecie wreszcie, wskazana jest tu uproszczona (zredukowana) lista rozkazów. Punkt ten nie jest tak oczywisty, jak pozostałe, jednak stanie się jaśniejszy podczas dalszej dyskusji. Zobaczymy ponadto, że żądanie wdrożenia całego procesora w postaci jednego mikroukładu sugeruje przyjęcie zredukowanej listy rozkazów.

12.2

UŻYCIĘ DUŻEJ TABLICY REJESTRÓW

Z rezultatów badań podsumowanych w podrozdz. 12.1 wynika, że pożądanym jest szybki dostęp do argumentów. Widzieliśmy, że w programach w językach wysokiego poziomu występuje duży odsetek instrukcji przypisania i że wiele z nich ma prostą postać $A = B$. Na jedną instrukcję w języku wysokiego poziomu przypada też znaczna liczba odniesień do argumentów. Jeśli połączymy te stwierdzenia z faktem, że większość z tych odniesień dotyczy lokalnych skalarów, wynika stąd sugestia, aby polegać na przechowywaniu w rejestrach.

Przyczyną tego, że wskazane jest przechowywanie w rejestrach jest to, że są one najszybszymi przyrządami do przechowywania, szybszymi niż pamięć główna i pamięć podręczna. Tablica rejestrów jest fizycznie mała. Znajduje się na ogół w tym samym mikroukładzie co ALU i jednostka sterująca oraz wymaga znacznie krótszych adresów niż pamięć główna i podręczna. Potrzebna jest więc strategia, która pozwoli na to, żeby najczęściej powoływane argumenty były przechowywane w rejestrach i żeby operacje rejestr-pamięć były zminimalizowane.

Możliwe są dwa podstawowe rozwiązania: jedno oparte na oprogramowaniu, drugie na sprzęcie. W rozwiązaniu programowym polega się na maksymalizacji użycia rejestrów przez kompilator. Kompilator może dążyć do przydzielenia rejestrów tym zmiennym, które w określonym czasie będą najczęściej używane. Rozwiązanie to wymaga użycia wyrafinowanych algorytmów analizy programów. Podejście sprzętowe polega po prostu na użyciu większej liczby rejestrów, dzięki czemu więcej zmiennych można przetrzymywać w rejestrach przez dłuższe okresy.

W tym podrozdziale przedyskutujemy rozwiązanie sprzętowe. Zostało ono zapoczątkowane przez grupę RISC z Berkeley [PATT82a] i było zastosowane w pierwszym komercyjnym wyrobie o architekturze RISC, jakim był procesor Pyramid [RAGA83].

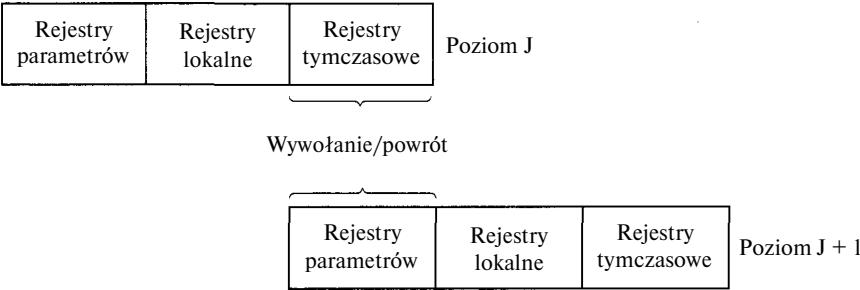
Okna rejestrów

Użycie dużego zestawu rejestrów przede wszystkim powinno ograniczyć potrzebę sięgania do pamięci. Projektowanie powinno polegać na zorganizowaniu rejestrów w taki sposób, żeby cel ten został osiągnięty.

Ponieważ większość odniesień do pamięci dotyczy lokalnych skalarów, oczywistym rozwiązaniem jest przechowywanie ich w rejestrach, z jednoczesnym ewentualnym pozostawieniem kilku rejestrów na zmienne globalne. Problem polega na tym, że pojęcie *lokalności* zmienia się wraz z każdym wywołaniem procedury i powrotem, a więc z operacjami, które występują często. Przy każdym wywołaniu zmienne lokalne muszą być zachowane poprzez przeniesienie z rejestrów do pamięci, dzięki czemu rejestry mogą być użyte ponownie przez wywołany program. Ponadto muszą być przekazane parametry. Przy powrocie zmienne programu macierzystego muszą być odtworzone (załadowane z powrotem do rejestrów), a wyniki muszą być przekazane z powrotem do programu macierzystego.

Rozwiązanie jest oparte na dwóch innych rezultatach badań, które omówiliśmy w podrozdz. 12.1. Po pierwsze, typowa procedura wykorzystuje tylko kilka przekazanych parametrów i zmiennych lokalnych. Po drugie, głębokość aktywacji procedury ulega wahaniom w stosunkowo wąskim zakresie (rys. 4.31). W celu wykorzystania tych własności używa się wielu małych zestawów rejestrów, z których każdy jest przypisany do innej procedury. Wywołanie procedury powoduje automatyczne przełączenie procesora do użycia innego okna rejestrów o ustalonym rozmiarze, zamiast zapisywania zawartości rejestrów w pamięci. Okna sąsiednich procedur nakładają się, co umożliwia przekazywanie parametrów.

Koncepcja ta jest zilustrowana na rys. 12.1. W dowolnej chwili widzialne i adresowalne jest tylko jedno okno rejestrów, tak jak gdyby istniał tylko jeden zestaw rejestrów (o adresach np. od 0 do $N - 1$). Okno jest podzielone na trzy obszary o stałych rozmiarach. W rejestrach parametrów przechowuje się para-



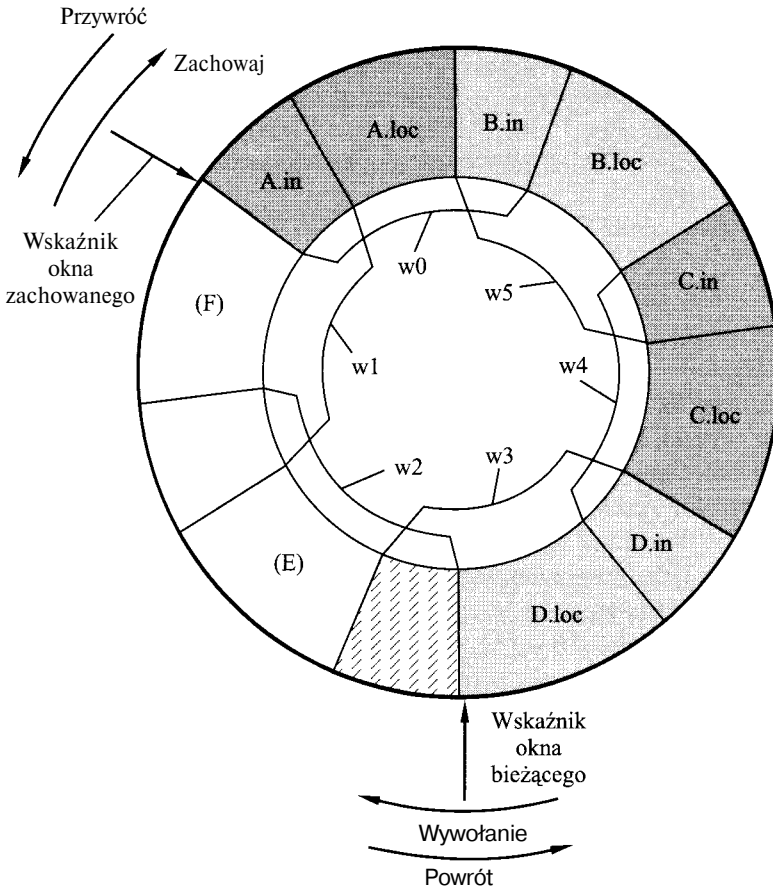
RYSUNEK 12.1. Nakładające się okna rejestrów

metry przekazane przez procedurę, która spowodowała wywołanie bieżącej procedury, oraz wyniki, które mają być przekazane z powrotem. Rejestry lokalne są używane do przechowywania zmiennych lokalnych przypisanych przez kompilator. Rejestry tymczasowe służą do wymiany parametrów i wyników z następnym niższym poziomem (z procedurą wywołaną przez procedurę bieżącą). Rejestry tymczasowe jednego poziomu są fizycznie tymi samymi rejestrami, co rejestry parametrów następnego niższego poziomu. To nakładanie się umożliwia przekazywanie parametrów bez rzeczywistego przenoszenia danych.

Wydawałoby się, że w celu sprostania dowolnym możliwym układom wywołań i powrotów liczba okien rejestrów powinna być nieograniczona. W rzeczywistości okna rejestrów mogą być używane do przechowywania argumentów kilku ostatnio wywoływanych procedur. Argumenty starszych procedur muszą być zapisywane w pamięci oraz odtwarzane później, gdy maleje głębokość zagnieżdżenia. Wobec tego rzeczywista organizacja tablicy rejestrów ma charakter cyklicznego bufora złożonego z nakładających się częściowo okien.

Organizację tę widać na rys. 12.2, na którym jest pokazany cykliczny bufor złożony z 6 okien. Bufor jest napełniony do poziomu 4 (A wywołała B; B wywołała C; C wywołała D), przy czym aktywna jest procedura D. Wskaźnik bieżącego okna (CWP) wskazuje okno aktualnie aktywnej procedury. Odniesienia do rejestrów ze strony rozkazów maszynowych są przesuwane za pomocą tego wskaźnika w celu określenia rzeczywistego rejestru fizycznego. Wskaźnik zachowanego okna (SWP) określa okno, którego zawartość została ostatnio przekazana do pamięci. Jeśli procedura D wywoła teraz procedurę E, to argumenty procedury E są już umieszczone w rejestrach tymczasowych D (nakładanie się w3 i w2), a stan CWP jest zwiększany o jedno okno.

Jeśli następnie procedura E wywoła procedurę F, to wywołanie to nie może być zrealizowane przy aktualnym stanie bufora. Jest tak, ponieważ okno F nakłada się z oknem A. Procedura F, przygotowując się do wywołania, rozpoczęłaby ładowanie swoich rejestrów tymczasowych, wymieniłaby tym samym zawartość rejestrów parametrów A (A.In). Wobec tego, gdy stan CWP jest zwiększany (modulo 6) i staje się równy SWP, następuje przerwanie i zachowywana jest zawartość okna A. Zachowane muszą być tylko pierwsze dwie części (A.In



RYSUNEK 12.2. Cykliczno-buforowa organizacja nakładających się okien

i A.loc). Następnie zwiększany jest stan SWP, a procedura F postępuje. Podobne przerwanie może nastąpić przy powrocie. Jeśli na przykład po wzbudzeniu procedury F procedura B wraca do A, stan CWP jest zmniejszany i staje się równy SWP. Powoduje to przerwanie, którego wynikiem jest odtworzenie okna procedury A.

Na podstawie powyższego opisu można zauważyć, że w tablicy rejestrów o N oknach można przechowywać tylko $N - 1$ wywołań procedur. Wartość N nie musi być duża. Jak wspomnieliśmy wcześniej, w jednym z badań [TAMI83] stwierdzono, że w przypadku 8 okien zachowywanie lub odtwarzanie jest wymagane tylko dla 1% wywołań i powrotów. W komputerach RISC Berkeley użyto 8 okien po 16 rejestrów każde. W komputerze Pyramid zastosowano 16 okien po 32 rejestry każde.

Zmienne globalne

Opisany powyżej układ okien stanowi efektywną organizację służącą do przechowywania zmiennych skalarnych w rejestrach lokalnych. Układ ten nie odpowiada jednak potrzebom przechowywania zmiennych globalnych wykorzystywanych przez więcej niż jedną procedurę (np. zmienne COMMON w FORTRAN-ie). Dwie opcje nasuwają się same. Po pierwsze, zmienne zadeklarowane w języku wysokiego poziomu jako globalne mogą być przypisane przez kompilator lokacjom w pamięci i wszystkie rozkazy maszynowe odnoszące się do tych zmiennych będą używały argumentów zawartych w pamięci. Jest to proste zarówno z punktu widzenia sprzętu, jak i oprogramowania (kompilatora). Jednak w przypadku często wywoływanych zmiennych globalnych układ ten jest nieefektywny.

Alternatywą jest wbudowanie do CPU zestawu rejestrów globalnych. Ich liczba powinna być ustalona i powinny one być dostępne dla wszystkich procedur. W celu uproszczenia formatu rozkazów może być użyty zunifikowany schemat numerowania. Na przykład odniesienia do rejestrów 0 do 7 mogą dotyczyć jednoznacznych rejestrów globalnych, a odniesienia do rejestrów 8 do 31 powinny być przemieszczane w celu określenia fizycznych rejestrów w bieżącym oknie. Oznacza to wzrost obciążenia sprzętowego w celu dostosowania się do podziału w adresowaniu rejestrów. Ponadto, kompilator musi zdecydować, które zmienne globalne powinny być przypisane rejestrom.

Duża tablica rejestrów a pamięć podręczna

Tablica rejestrów zorganizowana w postaci okien działa jako mały, szybki bufor służący do przechowywania wszystkich zmiennych, które wykazują wysokie prawdopodobieństwo częstego wykorzystywania. Z tego punktu widzenia tablica rejestrów działa podobnie do pamięci podręcznej. Powstaje zatem kwestia, czy nie byłoby prostsze i lepsze użycie pamięci podręcznej oraz niewielkiego, tradycyjnego zestawu rejestrów.

W tabeli 12.5 jest zamieszczone porównanie obu rozwiązań. W tablicy rejestrów opartej na oknach są przechowywane wszystkie skalarne zmienne lokalne (z wyjątkiem rzadkich przypadków przepełnienia okna) ostatnich $N - 1$ wywołań procedur. W pamięci podręcznej jest przechowywany pewien zbiór ostatnio używanych zmiennych skalarnych. Tablica rejestrów powinna umożliwiać oszczędzanie czasu, ponieważ są w niej przechowywane wszystkie lokalne zmienne skalarne. Jednak pamięć podręczna może ułatwić lepsze wykorzystanie przestrzeni, ponieważ umożliwia dynamiczne reagowanie na rozwój sytuacji. Ponadto, pamięci podręczne na ogół traktują wszystkie odniesienia do pamięci podobnie, włączając w to rozkazy i inne rodzaje danych. Zatem oszczędności w tych innych obszarach są możliwe w przypadku pamięci podręcznej, a nie występują w przypadku tablicy rejestrów.

TABELA 12.5. Własności organizacji opartych na dużych tablicach rejestrów oraz na pamięciach podręcznych

Duża tablica rejestrów	Pamięć podręczna
Wszystkie skalary lokalne	Ostatnio używane skalary lokalne
Pojedyncze zmienne	Bloki pamięci
Zmienne globalne przypisane przez kompilator	Ostatnio używane zmienne globalne
Zachowywanie/odtworzenie oparte na głębokości zagnieżdżenia procedury	Zachowywanie/odtworzenie oparte na algorytmie wymiany pamięci podręcznej
Adresowanie rejestrów	Adresowanie pamięci

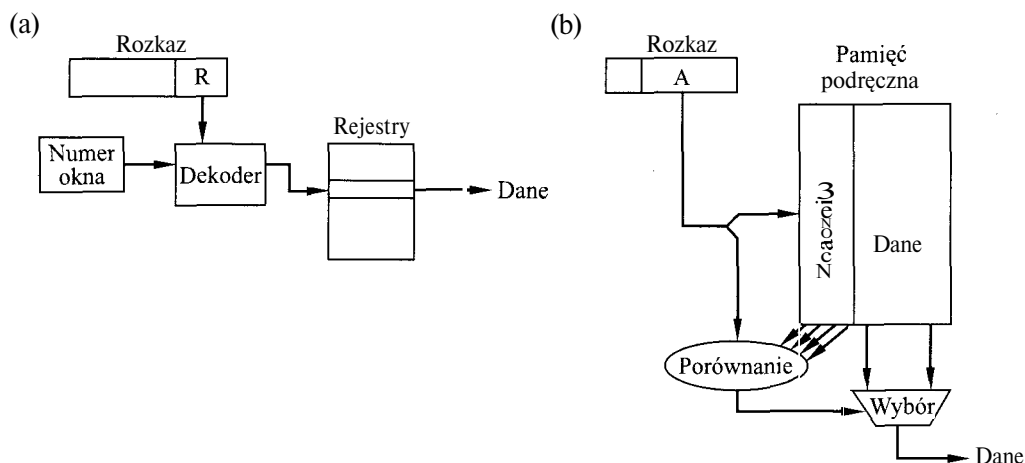
Tablica rejestrów może nieefektywnie wykorzystywać przestrzeń, ponieważ nie wszystkie procedury będą potrzebowały pełnej przestrzeni przypisanych im okien. Z drugiej strony pamięć podręczna charakteryzuje się nieefektywnością innego rodzaju: dane są wczytywane do pamięci podręcznej blokami. Podczas gdy tablica rejestrów zawiera tylko używane zmienne, do pamięci podręcznej są wczytywane bloki danych, z których wiele nie będzie używanych.

W pamięci podręcznej mogą być przechowywane zarówno zmienne lokalne, jak i globalne. Zwykle znajduje się w niej wiele skalarów globalnych, jednak niewiele z nich jest często używanych [KATE83]. Pamięć podręczna dynamicznie wykryje te zmienne i przechowuje. Jeśli tablica rejestrów oparta na oknach jest uzupełniona rejestrami globalnymi, ona również może przechowywać pewne skalary globalne. Kompilatorowi jest jednak trudno określić, które zmienne globalne będą intensywnie używane.

W przypadku tablicy rejestrów ruch danych między rejestrami a pamięcią jest zdeterminowany przez głębokość zagnieżdżenia procedury. Ponieważ głębokość ta zwykle waha się w wąskim zakresie, użycie pamięci jest niezbyt częste. Większość pamięci podręcznych stanowią pamięci sekcyjno-skojarzeniowe o małym rozmiarze sekcji. Wynika stąd niebezpieczeństwo zapisu kasującego często używane zmienne.

Na podstawie dotychczasowej dyskusji nie można w łatwy sposób zdecydować, czy wybrać dużą tablicę rejestrów opartą na oknach, czy pamięć podręczną. Istnieje jednak własność, która skłania ku wybraniu rozwiązania opartego na rejestrach i na podstawie której można stwierdzić, że system oparty na pamięci podręcznej będzie zauważalnie wolniejszy. Rozróżnienie to wynika z nakładów na adresowanie towarzyszących obu rozwiązaniom.

Różnicę tę widać na rys. 12.3. W celu odniesienia do lokalnego skalara w tablicy rejestrów opartej na oknach jest używany „wirtualny” numer rejestru oraz numer okna. Mogą one być przepuszczone przez stosunkowo prosty dekodery w celu wybrania jednego z rejestrów fizycznych. W celu odniesienia do lokacji w pamięci podręcznej musi być wygenerowany adres pamięci o pełnej szerokości. Złożoność tej operacji zależy od trybu adresowania. W sekcyjno-skojarzeniowej pamięci podręcznej część adresu jest używana do odczytania liczby słów i znaczników równej rozmiarowi sekcji. Inna część adresu jest porównywana ze wskaźnikami i wybierane jest jedno z odczytanych słów. Powin-



RYСУNEK 12.3. Odniesienie do skłara lokalnego: (a) tablica rejestrów oparta na oknach; (b) pamięć podręczna

no być jasne, że jeśli nawet pamięć podręczna będzie tak samo szybka jak tablica rejestrów, to czas dostępu będzie znacznie dłuższy. Wobec tego, z punktu widzenia wydajności, tablica rejestrów oparta na oknach jest lepsza do przechowywania lokalnych skłarów. Dalsza poprawa wydajności może być uzyskana przez dodanie pamięci podręcznej przeznaczonej wyłącznie do przechowywania rozkazów.

12.3

OPTIMALIZACJA REJESTRÓW ZA POMOCĄ KOMPILATORA

Załóżmy teraz, że w docelowej maszynie RISC jest dostępna jedynie niewielka liczba rejestrów (np. $16 \div 32$). W tym przypadku za optymalne wykorzystanie rejestrów odpowiada kompilator. Program napisany w języku wysokiego poziomu nie wykazuje oczywiście jawnych odniesień do rejestrów, raczej odwołuje się do potrzebnych wielkości symbolicznie. Celem kompilatora jest maksymalnie długie przetrzymywanie argumentów w rejestrach zamiast w pamięci oraz minimalizacja operacji ładowania i zapisu.

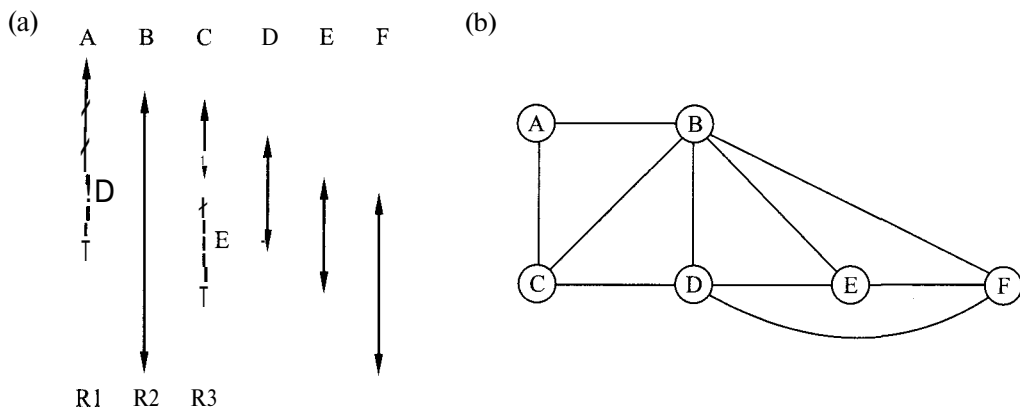
Na ogół stosuje się następujące rozwiązanie. Każda wielkość programu mająca pozostawać w rejestrach jest przypisana do rejestru symbolicznego lub wirtualnego. Następnie kompilator odwzorowuje nieograniczoną liczbę rejestrów symbolicznych na ustalonej liczbie rejestrów rzeczywistych. Rejestry symboliczne, których zastosowania nie nakładają się, mogą być przypisane do tego samego rejestru. Jeśli w określonej części programu występuje więcej wielkości niż rejestrów rzeczywistych, to niektóre z tych wielkości mogą być przypisane do lokacji w pamięci. Aby tymczasowo umieścić te wielkości w rejestrach w celu przeprowadzenia operacji, używa się rozkazów ładowania i zapisu.

Istotą zadania optymalizacji jest zdecydowanie, które wielkości mają być przypisane rejstrów w określonym punkcie programu. Metoda najczęściej stosowana w kompilatorach RISC nazywa się **kolorowaniem grafów** (ang. *graph coloring*) i zapożyczono ją z topologii [CHAI82, CHOW86, COUT86, CHOW90].

Problem kolorowania grafów wygląda następująco. Dany jest graf złożony z węzłów i krawędzi. Należy przypisać węzłom kolory w taki sposób, żeby sąsiednie węzły miały różne kolory i żeby liczba różnych kolorów była minimalna. Problem ten został zaadaptowany do tworzenia kompilatorów. Po pierwsze, analizuje się program w celu zbudowania grafu (wykresu) interferencji rejstrów. Węzłami grafu są rejestry symboliczne. Jeśli dwa rejestry symboliczne są aktywne w tym samym fragmencie programu, łączy się je krawędzią odzwierciedlającą interferencję. Następnie próbuje się pokolorować graf za pomocą n kolorów, gdzie n jest liczbą rejstrów. Jeśli proces ten nie w pełni się udaje, to węzły, które nie mogą być pokolorowane, muszą być umieszczone w pamięci. Gdy będą one potrzebne, miejsce dla nich jest tworzone za pomocą rozkazów ładowania i zapisu.

Na rysunku 12.4 widać prosty przykład takiego procesu. Załóżmy, że program o 6 rejestrach symbolicznych ma być skompilowany do 3 rejstrów rzeczywistych. Na rysunku 12.4a jest zilustrowana sekwencja czasowa aktywacji każdego rejestru symbolicznego, a w części b - graf interferencji rejstrów. Pokazano możliwość pokolorowania za pomocą trzech kolorów. Jeden z rejstrów symbolicznych, F, pozostał niepokolorowany i musi być używany za pomocą operacji ładowania i zapisu.

Na ogół istnieje pewna wymiennność między używaniem dużego zestawu rejstrów a optymalizacją rejstrów za pomocą kompilatora. Na przykład w pracy [BRAD91a] opisano badanie, w którym modelowano architekturę RISC o własnościach zbliżonych do procesorów Motorola 88000 i MIPS R2000.



RYСУNEK 12.4. Podejście oparte na kolorowaniu grafów: (a) sekwencja czasowa aktywnego używania rejstrów symbolicznych; (b) graf interferencji rejstrów

W badaniu zmieniano liczbę rejestrów od 16 do 128 i rozpatrywano zarówno używanie ich jako rejestrów ogólnego przeznaczenia, jak i podzielenie rejestrów na zmiennopozycyjne i liczb całkowitych. Badanie wykazało, że nawet przy zrównoważonej optymalizacji rejestrów korzyść z używania większej liczby rejestrów niż 64 jest niewielka. Przy rozsądnie wyrafinowanych metodach optymalizacji liczby rejestrów, już powyżej 32 rejestrów występuje marginalna poprawa wydajności. Stwierdzono również, że przy małej liczbie rejestrów (np. 16) maszyna o wspólnych rejestrach funkcjonuje szybciej niż maszyna o podzielonych rejestrach. Podobne wnioski można wyciągnąć z pracy [HUGU91], w której są opisane badania poświęcone bardziej optymalizacji użycia małej liczby rejestrów, niż porównywaniu użycia dużych zestawów rejestrów w połączeniu z optymalizacją.

12.4

ARCHITEKTURA O ZREDUKOWANEJ LIŚCIE ROZKAZÓW

W tym podrozdziale rozpatrzymy niektóre własności ogólne i sposoby stosowania architektury o zredukowanej liście rozkazów. Przykłady przedstawimy w dalszej części rozdziału. Rozpocniemy od przedyskutowania powodów stosowania współczesnych architektur o złożonej liście rozkazów.

Dlaczego CISC

Odnotowaliśmy tendencję do stosowania bogatszych list rozkazów, które zawierają zarówno większą liczbę rozkazów, jak i rozkazy bardziej złożone. Powody występowania tej tendencji były dwa: dążenie do uproszczenia kompilatorów oraz dążenie do poprawy wydajności. Podłożem obu tych przyczyn było przedstawienie się części programistów na języki wysokiego poziomu; architekci dążyli do zaprojektowania maszyn, które zapewniałyby lepsze wsparcie języków wysokiego poziomu.

Nie mamy zamiaru stwierdzić w tym rozdziale, że projektanci CISC obrali niewłaściwy kierunek. Technologia RISC jest bardzo młoda i debata CISC przeciwko RISC nie została jeszcze rozstrzygnięta. W rzeczywistości, ponieważ technologia wciąż ewoluje, a architektury te mają raczej naturę widm, ocena typu czarne-białe prawdopodobnie nigdy nie wystąpi. Dlatego celem dalszych rozważań jest po prostu wypunktowanie pewnych potencjalnych pułapek rozwiązania CISC i ułatwienie zrozumienia motywów zwolenników architektury RISC.

Pierwsza spośród wymienionych przyczyn, to znaczy uproszczenie kompilatorów, wygląda na oczywistą. Zadaniem twórcy kompilatorów jest generowanie ciągu rozkazów maszynowych dla każdej instrukcji w języku wysokiego poziomu. Jeśli istnieją rozkazy maszynowe, które zastępują instrukcje w języku

wysokiego poziomu, to zadanie to jest uproszczone. Takie rozumowanie zostało przedyskutowane przez badaczy RISC [HENN82, RAD183, PATT82b]. Stwierdzili oni, że złożone rozkazy maszynowe są często trudne do wykorzystania, ponieważ kompilator musi odnaleźć te przypadki, które dokładnie pasują do konstrukcji. Zadanie optymalizacji generowanego kodu w celu minimalizacji jego rozmiaru, zredukowania liczby wykonań rozkazów i spotęgowania potokowania jest o wiele trudniejsze w przypadku złożonej listy rozkazów. Dowodem na to są cytowane wcześniej badania wykazujące, że większość rozkazów w skompilowanych programach stanowią rozkazy stosunkowo proste.

Inną ważną, cytowaną już przyczyną jest oczekiwanie, że CISC pozwoli uzyskiwać mniejsze i szybsze programy. Przeanalizujemy oba aspekty stwierdzenia: że programy będą mniejsze i że będą szybciej realizowane.

Mniejsze programy mają dwie zalety. Po pierwsze, występują oszczędności pamięci. Wobec taniości współczesnych pamięci ta potencjalna zaleta nie jest już tak atrakcyjna. Ważniejsze jest, że mniejsze programy powinny umożliwić poprawienie wydajności i to na dwa sposoby. Po pierwsze, mniej rozkazów oznacza mniej bajtów do pobrania. Po drugie zaś, w środowisku stronicowanym mniejsze programy zajmują mniej stron, co powoduje zmniejszenie liczby błędów stron.

Problemem występującym w tej linii rozumowania jest to, że nie jest pewne, czy program CISC będzie mniejszy niż odpowiedni program RISC. W wielu przypadkach program CISC wyrażony w symbolicznym języku maszynowym może być *krótszy* (to znaczy może zawierać mniej rozkazów), jednak liczba zajmowanych bitów pamięci może nie być zauważalnie *mniejsza*. W tabeli 12.6 są pokazane wyniki trzech badań, w których porównywano rozmiary skompilowanych programów w języku C dla różnych maszyn łącznie z RISC I o zredukowanej liście rozkazów. Zauważmy, że CISC przynosi nieznaczne lub nie przynosi żadnych oszczędności w porównaniu z RISC. Jest również interesujące, że komputer VAX dysponujący o wiele bardziej złożoną listą rozkazów niż PDP-11 umożliwia osiągnięcie niewielkich oszczędności w porównaniu z tym ostatnim. Wyniki te zostały potwierdzone przez badaczy IBM [RAD183], którzy stwierdzili, że program skompilowany na komputer IBM 801 (RISC) był równy 0,9 programu skompilowanego na IBM S/370. Badanie dotyczyło zestawu programów w języku PL/I.

TABELA 12.6. Rozmiar programu w stosunku do RISC I

	[PATT82a] 11 programów C	[KATE83] 12 programów C	[HEAT84] 5 programów C
RISC I	1,0	1,0	1,0
VAX-11/780	0,8	0,67	
M68000	0,9		0,9
Z8002	1,2		1,12
PDP-11/70	0,9	0,71	

Istnieje kilka przyczyn tych raczej zaskakujących wyników. Zauważyliśmy już, że kompilatory dla CISC wykazują tendencję do faworyzowania prostszych rozkazów, więc zwiększając złożoność rozkazów rzadko odgrywa rolę. Ponadto, ponieważ w CISC występuje więcej rozkazów, wymagane są dłuższe kody operacji, co powoduje wydłużenie rozkazów. Wreszcie maszyny RISC wykazują tendencję do akcentowania raczej odniesień do rejestrów niż do pamięci, a te pierwsze wymagają mniej bitów. Przykład tego ostatniego zjawiska omawiamy poniżej (patrz rys. 12.5).

Tak więc oczekiwanie, że CISC umożliwi powstanie mniejszych programów wraz z towarzyszącymi im zaletami, może nie być spełnione. Drugim czynnikiem motywującym wzrastającą złożoność list rozkazów było przyspieszenie wykonywania rozkazów. Oczekiwanie, że złożona operacja w języku wysokiego poziomu zostanie wykonana szybciej jako pojedynczy rozkaz maszynowy niż jako ciąg bardziej prymitywnych rozkazów, wygląda na uzasadnione. Jednak ze względu na tendencję do stosowania prostszych rozkazów może to nie być prawdą. Jednostka sterująca musi być bardziej skomplikowana i (lub) przechowywanie mikroprogramów sterowania musi zajmować większą pojemność, co wynika z bogatszej listy rozkazów. Oba te czynniki wydłużają czas wykonywania prostych rozkazów.

W rzeczywistości niektórzy badacze stwierdzili, że przyspieszenie wykonywania złożonych funkcji wynika nie tyle z potężnej złożoności rozkazów maszynowych, co z ich przechowywania w szybkiej pamięci sterującej [RAD183]. W rezultacie pamięć sterująca działa jako podreźczna pamięć rozkazów. Wobec

(a)

8	16	16	16
poa	a	C	A
16	16	16	16

Pamięć-pamięć

I = 56, D = 96, M = 152

(b)

8	16	16	16
poa	a	C	a
16	16	16	16

Pamięć-pamięć

I = 168, D = 288, M = 456

8	16	16	16
poa	a	C	a
16	16	16	16

Pamięć-pamięć

I = 60, a = 0, n = 60

8	4	16
Ładuj	RB	a
Ładuj	RC	C
Dodaj	RA	RB
Dodaj	RA	RC
Zapisz	RA	V

i^o^sa-fj-ffsfo

I = 104, D = 96, M = 200

r – rozmiar wykonywanych rozkazów
Q – rozmiar przetwarzanych danych
M = r + Q = całkowity ruch do (i z) pamięci

RYSEK 12.5. Porównanie rozwiązań rejestr-rejestr oraz pamięć-pamięć: (a) $V \leftarrow B + C$; (b) $A \leftarrow B + C$, $B \leftarrow A + C$, $D \leftarrow D - B$

tego projektant sprzętu powinien spróbować określić, które podprogramy lub funkcje będą używane najczęściej, i wprowadzić je do pamięci sterującej przez wdrożenie ich w postaci mikrokodu. Wyniki okazały się mniej niż zachęcające. W związku z tym w systemach S/370 rozkazy takie, jak Translate and Extended-Precision-Floating-Point-Divide (translacja i dzielenie zmiennopozycyjne o zwiększonej dokładności) rezydują w szybkiej pamięci sterującej, podczas gdy sekwencje dotyczące tworzenia wywołań procedur lub inicjowania programu obsługi przerwań znajdują się w wolniejszej pamięci głównej.

Nie jest więc oczywiste, że tendencja do zwiększania złożoności list rozkazów jest właściwa. Skłoniło to wiele grup projektantów do wybrania przeciwstawnej drogi.

Własności architektur o zredukowanej liście rozkazów

Chociaż było wiele różnych podejść do architektury o zredukowanej liście rozkazów, pewne własności pozostają wspólne dla wszystkich. Są one wymienione w tabeli 12.7 i opisane poniżej. Przykłady przeanalizujemy w dalszym ciągu rozdziału.

TABELA 12.7. Własności architektury o zredukowanej liście rozkazów

Jeden rozkaz na cykl
Operacje „z rejestru do rejestru”
Proste tryby adresowania
Proste formaty rozkazów

Pierwszą z własności wymienionych w tabeli 12.7 jest to, że na jeden cykl maszynowy przypada jeden rozkaz maszynowy. Cykl maszynowy jest definiowany jako czas wymagany do pobrania 2 argumentów z rejestrów, przeprowadzenia operacji ALU i zapisania wyniku w rejestrze. Rozkazy RISC nie powinny więc być bardziej skomplikowane i wolniej wykonywane niż mikrorozkazy w maszynach CISC. Przy prostych, 1-cyklowych rozkazach nie występuje potrzeba mikrokodu; rozkazy maszynowe mogą być implementowane sprzętowo. Takie rozkazy powinny być wykonywane szybciej niż porównywalne rozkazy w innych maszynach, ponieważ podczas wykonywania rozkazów nie ma potrzeby sięgania do mikroprogramu w pamięci sterującej.

Drugą własnością jest to, że większość operacji powinna mieć charakter rejestr-do-rejestru, jedynie w połączeniu z prostymi operacjami dostępu do pamięci (LOAD i STORE). Własność ta powoduje uproszczenie listy rozkazów i wobec tego jednostki sterującej. Lista rozkazów RISC może na przykład zawierać tylko jeden lub dwa rozkazy ADD (np. dodaj całkowite, dodaj z przeniesieniem); VAX ma 25 różnych rozkazów ADD. Inną korzyścią jest to, że taka architektura skłania do optymalizacji rejestrów, dzięki czemu często wywoływane argumenty pozostają w szybkiej pamięci.

Nacisk na operacje rejestr-do-rejestru jest charakterystyczny dla projektów RISC. Inne współczesne maszyny dysponują takimi rozkazami, jednak obejmują także operacje pamięć-do-pamięci oraz mieszane operacje rejestr-pamięć. Próby porównania tych rozwiązań podejmowano w latach siedemdziesiątych, jeszcze przed pojawieniem się architektur RISC. Na rysunku 12.5a jest pokazane przyjęte podejście. Hipotetyczne architektury oceniano pod kątem rozmiaru programów i liczby bitów zapisywanych i pobieranych z pamięci. Uzyskane wyniki doprowadziły jednego z badaczy do zasugerowania, że przyszłe architektury w ogóle nie powinny zawierać rejestrów [MYER78]. Można się zastanawiać, co sądziłby on dzisiaj, gdy firma Pyramid wprowadziła na rynek maszynę RISC zawierającą ni mniej, ni więcej, tylko 528 rejestrów!

W badaniach tych umknęło rozpoznanie częstego dostępu do małej liczby lokalnych skalarów oraz to, że dzięki dużemu bankowi rejestrów lub optymalizacji za pomocą kompilatorów większość argumentów może być przez długi czas trzymana w rejestrach. Dlatego też rys. 12.5b może stanowić uczciwsze porównanie.

Trzecią własnością wymienioną w tabeli 12.7 jest użycie prostych trybów adresowania. W prawie wszystkich rozkazach używa się prostego adresowania rejestrowego. Może też być włączonych kilka dodatkowych trybów, takich jak tryb przesunięciowy lub względny wobec licznika rozkazów. Inne, bardziej złożone tryby mogą być składane programowo z prostszych. I znów własność ta umożliwia uproszczenie listy rozkazów i jednostki sterującej.

Ostatnią ze wspólnych własności jest używanie prostych formatów rozkazu. Na ogół stosuje się jeden lub kilka formatów rozkazu. Długość rozkazów jest stała i wyrównana z granicami słów. Położenia pól, w szczególności kodu systemu operacyjnego, są ustalone. Własność ta ma wiele zalet. Przy ustalonych polach, dekodowanie kodu operacji i dostęp do argumentów w rejestrach mogą być dokonywane jednocześnie. Uproszczone formaty umożliwiają uproszczenie jednostki sterującej. Pobieranie rozkazów jest zoptymalizowane, ponieważ pobierane są jednostki o długości słowa. Oznacza to także, że pojedyncze rozkazy nie wykraczają poza granice stron.

Własności te potraktowane łącznie mogą służyć do określenia potencjalnych korzyści z zastosowania rozwiązania RISC. Korzyści te można podzielić na dwie główne kategorie: związane z wydajnością i związane z wdrażaniem układów VLSI.

W odniesieniu do wydajności można przedstawić następujące „poszlaki”. Po pierwsze, mogą być opracowane bardziej efektywne kompilatory optymalizujące. Przy bardziej prymitywnych rozkazach jest więcej sposobności do zmniejszania liczby pętli, zreorganizowania programów pod kątem efektywności, maksymalizacji wykorzystania rejestrów i tak dalej. Jest nawet możliwe obliczanie części złożonych rozkazów w czasie kompilacji. Na przykład rozkaz Move Characters (MVC) w S/370 powoduje przemieszczenie łańcucha znaków z jednej lokacji do drugiej. Za każdym razem, gdy jest on wykonywany, przesunięcie

zależy od długości łańcucha, od tego, czy i w którym kierunku lokacje nakładają się, oraz jakie jest wyrównanie (*alignment*). W większości przypadków odpowiedzi na te pytania są znane w czasie kompilacji. Wobec tego kompilator mógłby tworzyć zoptymalizowaną sekwencję prymitywnych rozkazów dla tej funkcji.

Drugim, już odnotowanym punktem jest to, że w każdym przypadku większość rozkazów generowanych przez kompilator stanowią rozkazy stosunkowo proste. Wydaje się rozsądne, że jednostka sterująca zbudowana specjalnie dla tych rozkazów i wykorzystująca niewiele (lub wcale) mikrorozkazów mogłaby wykonywać je szybciej niż porównywalna jednostka CISC.

Punkt trzeci odnosi się do potokowego przetwarzania rozkazów. Badacze architektury RISC przeczuwają, że metoda potokowego przetwarzania rozkazów może być stosowana bardziej efektywnie ze zredukowaną listą rozkazów. Przeanalizujemy teraz ten punkt bardziej szczegółowo.

Ostatnim i nieco mniej znaczącym punktem jest to, że programy RISC powinny lepiej reagować na przerwania, ponieważ przerwania są dokonywane najczęściej między operacjami elementarnymi. Architektury o złożonych rozkazach albo ograniczają przerwania do granic rozkazów, albo muszą określić specjalne punkty dopuszczalnych przerw i wdrożyć mechanizmy powtórnego uruchomienia rozkazu.

Poprawa wydajności wynikająca z architektury o zredukowanej liście rozkazów jest daleka od udowodnienia. Przeprowadzono wiele badań, jednak nie na maszynach o porównywalnej technologii i mocy. Ponadto, w większości badań nie dążono do oddzielenia efektów zredukowanej listy rozkazów i efektów dużego zestawu rejestrów. „Poszlaki” są jednak sugestywne.

Drugi obszar potencjalnych korzyści, który jest wyraźniej zarysowany, wiąże się z wdrażaniem układów VLSI. Gdy stosuje się układy VLSI, projektowanie i implementacja procesora ulega podstawowym zmianom. Tradycyjny procesor, taki jak IBM S/370 czy VAX, składa się z jednego lub wielu modułów drukowanych zawierających znormalizowane układy SSI i MSI. Wraz z wynalezieniem układów LSI i VLSI stało się możliwe zmieszczenie całego procesora w jednym mikroukładzie. W przypadku jednoukładowego procesora występują dwa argumenty przemawiające za stosowaniem strategii RISC. Po pierwsze – zagadnienie wydajności: opóźnienia wewnątrz mikroukładu są o wiele mniejsze niż między mikroukładami. Racjonalne jest więc poświęcenie cennej powierzchni mikroukładu na implementację tych operacji, które występują często. Widzieliśmy, że w rzeczywistości proste rozkazy i odniesienia do lokalnych skalarów są najczęstszymi działaniami. Mikroukłady RISC z Berkeley zostały zaprojektowane właśnie przy uwzględnieniu tych rozważań. Podczas gdy w typowym mikroprocesorze jednoukładowym około połowy powierzchni jest przeznaczone na pamięć sterującą zawierającą mikrorozkazy, w mikroukładzie RISC I jednostka sterująca zajmuje zaledwie około 6% powierzchni [SHER84].

Drugim zagadnieniem związanym z układami VLSI jest czas projektowania i wdrażania. Procesor VLSI jest trudny do opracowania. Zamiast polegać na dostępnych podzespołach SSI/MSI, projektant musi zaprojektować cały układ i jego strukturę już na poziomie podzespołu. W przypadku architektury o zredukowanej liście rozkazów proces ten jest daleko łatwiejszy, co widać w tabeli 12.8 [FITZ81]. Jeśli ponadto wydajność mikroukładu RISC będzie równoważna porównywalnym mikroprocesorom CISC, to zalety rozwiązania RISC stają się ewidentne.

TABELA 12.8. Pracochłonność projektowania układu i topologii wybranych mikroprocesorów

Procesor	Tranzystory (tysiący)	Projektowanie układu (osobomiesięcy)	Projektowanie topologii (osobomiesięcy)
RISC I	44	15	12
RISC II	41	18	12
M68000	68	100	70
Z8000	18	60	70
Intel iAPx-432	110	170	90

Własności CISC a RISC

Po początkowym entuzjazmie w stosunku do maszyn RISC zaczęło narastać przekonanie, że (1) projekty RISC mogą zyskać na włączeniu pewnych elementów CISC oraz że (2) projekty CISC mogą zyskać na włączeniu pewnych elementów RISC. W rezultacie nowsze projekty RISC, w tym PowerPC, nie są już „czystymi” RISC, a nowsze projekty CISC, w tym Pentium, zawierają pewne własności RISC.

Interesujące porównanie zawarte w pracy [MASH94] daje pewien wgląd w to zagadnienie (tabela 12.9). W tabeli wymieniono pewną liczbę procesorów i porównano ich własności. Dla celów tego porównania przyjęto, że typowe dla RISC są następujące własności:

1. Rozkaz o pojedynczej długości.
2. Długość ta wynosi zwykle 4 bajty.
3. Mała liczba trybów adresowania danych, zwykle mniejsza niż pięć. Parametr ten jest trudny do sprecyzowania. Tryb rejestrowy i znakowy nie są w tej tablicy liczone, natomiast różne formaty o różnych rozmiarach wyrównania są liczone oddzielnie.
4. Brak adresowania pośredniego, które wymaga jednego dostępu do pamięci w celu otrzymania adresu argumentu z pamięci.
5. Brak operacji łączących ładowanie/zapis z arytmetyką (np. dodaj z pamięci, dodaj do pamięci).
6. Co najwyżej jeden argument w rozkazie jest adresowany w pamięci.
7. Nie jest wykonywane dowolne wyrównywanie danych dla operacji ładowania/zapisu.

8. Maksymalna liczba zastosowań jednostki zarządzania pamięcią (MMU) w odniesieniu do adresu danych w rozkazie.
9. Liczba bitów specyfikatora rejestru liczb całkowitych jest równa 5 lub większa. Oznacza to, że przynajmniej 32 rejestry liczb całkowitych mogą być jednocześnie adresowane w sposób jawny.
10. Liczba bitów specyfikatora rejestru liczb zmiennopozycyjnych jest równa 4 lub większa. Oznacza to, że przynajmniej 16 rejestrów zmiennopozycyjnych może być jednocześnie adresowanych w sposób jawny.

Tabela 12.9. Własności wybranych procesorów

Procesor	Złożoność dekodowania			Trudność przetwarzania potokowego					Wspieranie kompilatora	
	liczba rozmiarów rozkazów	maksymalny rozmiar rozkazu (bitów)	liczba trybów adresowania	adresowanie pośrednie	ładuj/zapisz przy arytmetyczmie z danej	maksymalna liczba argumentów porównań	dozwolone adresowanie nie wyrównane	maksymalna liczba zastosowań MMU	liczba bitów specyfikatora rejestru liczb całkowitych	liczba bitów specyfikatora rejestru zmiennopozycyjnego
AMD29000	1	4	1	nie	nie	1	nie	1	8	3 ^a
MIPS R2000	1	4	1	nie	nie	1	nie	1	5	4
SPARC	1	4	2	nie	nie	1	nie	1	5	4
MC88000	1	4	3	nie	nie	1	nie	1	5	4
HP PA	1	4	10 ^a	nie	nie	1	nie	1	5	4
IBM RT/PC	2 ^a	4	1	nie	nie	1	nie	1	4 ^a	3 ^a
IBM RS/6000	1	4	4	nie	nie	1	tak	1	5	5
Intel i860	1	4	4	nie	nie	1	nie	1	5	4
IBM 3090	4	8	2 ^b	nie ^b	tak	2	tak	4	4	2
Intel 80486	12	12	15	nie ^b	tak	2	tak	4	3	3
NSC 32016	21	21	23	tak	tak	2	tak	4	3	3
MC68040	11	22	44	tak	tak	2	tak	8	4	3
VAX	56	56	22	tak	tak	6	tak	24	4	0
Clipper	4 ^a	8 ^a	9 ^a	nie	nie	1	0	2	4 ^a	3 ^a
Intel 80960	T	8 ^a	9 ^a	nie	nie	1	tak ^a	–	5	3 ^a

^a Procesor RISC, który nie podporządkowuje się tej własności,^b Procesor CISC, który nie podporządkowuje się tej własności.

Cechy od 1 do 3 są wskaźnikami złożoności dekodowania rozkazu. Cechy od 4 do 8 sugerują łatwość lub trudność przetwarzania potokowego, zwłaszcza wobec wymagań pamięci wirtualnej. Cechy 9 i 10 wiążą się ze zdolnością do wykorzystywania zalet kompilatorów.

Pierwszych 8 procesorów w tabeli ma w sposób oczywisty architekturę RISC, następnych 5 - oczywiście CISC, a ostatnie 2 procesory w rzeczywistości mają wiele cech CISC, chociaż są często uważane za RISC.

12.5

PRZETWARZANIE POTOKOWE W ARCHITEKTURZE RISC**Przetwarzanie potokowe rozkazów regularnych**

Jak stwierdziliśmy w podrozdz. 11.4, w celu zwiększenia wydajności stosuje się często potokowe przetwarzanie rozkazów. Rozważmy to zagadnienie w kontekście architektury RISC. Większość rozkazów jest typu rejestr-rejestr, a cykl rozkazu ma następujące dwie fazy:

- a I. Pobranie rozkazu.
- E. Wykonanie rozkazu. Wykonywana jest operacja ALU, przy czym dane wejściowe są pobierane z rejestrów i do rejestrów są kierowane wyniki.

W przypadku operacji ładowania i zapisu wymagane są trzy fazy:

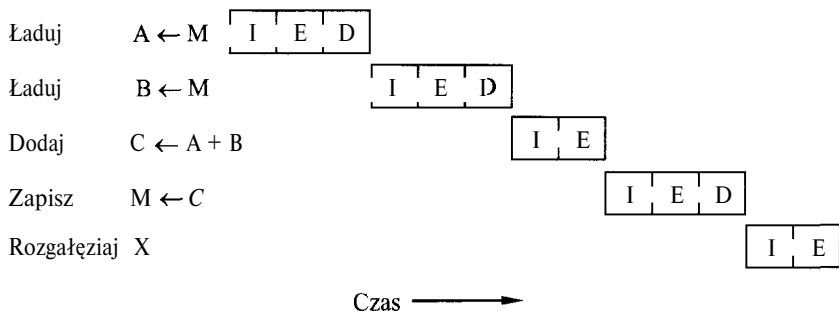
- a L Pobranie rozkazu.
- E. Wykonanie rozkazu. Obliczenie adresu w pamięci.
- a D. Pamięć. Operacja rejestr-pamięć lub pamięć-rejestr.

Na rysunku 12.6 widać przebieg czasowy ciągu rozkazów bez przetwarzania potokowego. Jest to oczywiście proces marnotrawny. Nawet bardzo proste przetwarzanie potokowe może istotnie zwiększyć wydajność. Na rysunku 12.7 jest pokazany 2-etapowy schemat przetwarzania potokowego, w którym fazy I oraz E dwóch różnych rozkazów są realizowane równocześnie. Rozwiązanie to może umożliwić niemal dwukrotne zwiększenie szybkości wykonywania rozkazów w porównaniu ze schematem szeregowym. Dwa problemy uniemożliwiają osiągnięcie maksymalnego przyspieszenia. Po pierwsze zakładamy, że stosowana jest pamięć z jednym portem i w określonej fazie możliwy jest tylko jeden dostęp do pamięci. Wymaga to uwzględnienia stanu oczekiwania w przypadku niektórych rozkazów. Po drugie, rozkaz rozgałęzienia przerywa szeregowy proces wykonywania. W celu dostosowania się do tych ograniczeń przy minimalnej rozbudowie układów, za pomocą kompilatora lub assemblera w strumieniu rozkazów można umieścić rozkaz NOOP*.

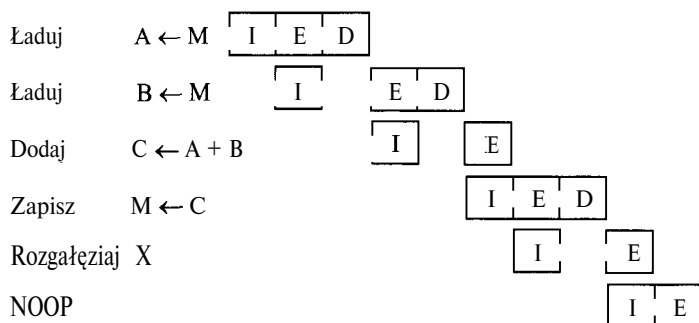
Dalszą poprawę przetwarzania potokowego można uzyskać poprzez umożliwienie dwóch dostępów do pamięci w ciągu tej samej fazy. Prowadzi to do sekwencji pokazanej na rys. 12.8. Teraz może mieć miejsce nakładanie się do

* Rozkaz NOOP oznacza brak operacji (*no operation*), czyli rozkaz pusty (*przyp. tłum.*).

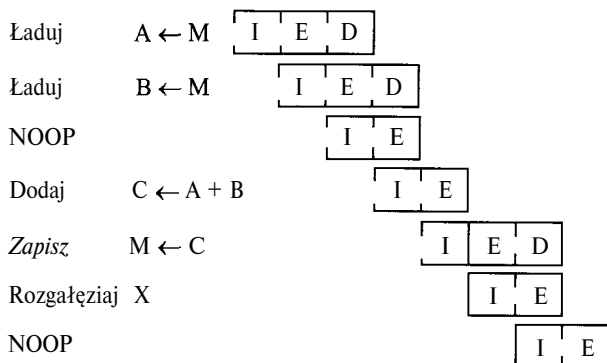
3 rozkazów, a poprawa wydajności może być nawet 3-krotna. I znów rozkazy rozgałęzienia powodują niepełne wykorzystanie możliwości przyspieszenia. Zauważmy, że zależności między danymi również mają pewien wpływ. Jeśli rozkaz wymaga argumentu, który został zmieniony przez poprzedni rozkaz, potrzebne jest opóźnienie. Podobnie jak poprzednio, można się do tego dostosować za pomocą rozkazu NOOP.



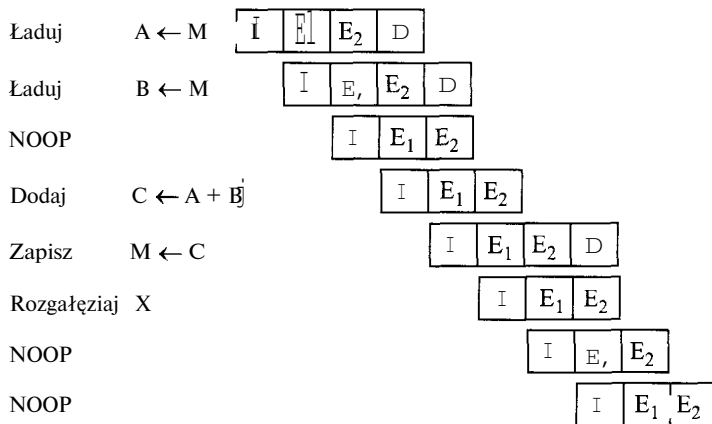
RYSUNEK 12.6. Przebieg czasowy wykonywania sekwencyjnego



RYSUNEK 12.7. Przebieg czasowy przetwarzania potokowego 2-etapowego



RYSUNEK 12.8. Przebieg czasowy przetwarzania potokowego 3-etapowego



RYSUNEK 12.9. Przebieg czasowy przetwarzania potokowego 4-etapowego

Przetwarzanie potokowe przedyskutowane dotychczas funkcjonuje najlepiej, jeśli trzy fazy trwają w przybliżeniu tyle samo. Ponieważ faza E obejmuje zwykle operację ALU, może ona być dłuższa. W tym przypadku możemy ją podzielić na dwie podfazy:

- E₁. Odczyt pliku z rejestru.
- E₂. Operacja ALU i zapis w rejestrze.

Ze względu na prostotę i regularność listy rozkazów, zaprojektowanie schematu 3- lub 4-fazowego jest łatwe. Na rysunku 12.9 widać potok 4-etapowy. W określonej chwili w trakcie wykonywania może się znajdować do 4 rozkazów, co daje maksymalny potencjalny czynnik przyspieszenia równy 4. Zauważmy, że w celu dostosowania się do opóźnień spowodowanych przez rozgałęzianie i zależność danych, znów potrzebne są rozkazy NOOP.

Optymalizacja przetwarzania potokowego

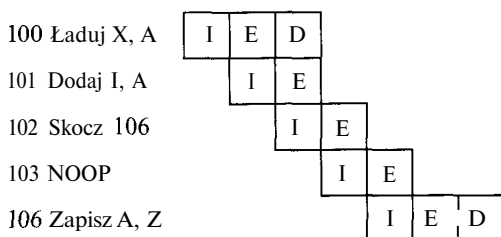
Dzięki prostocie i regularności rozkazów RISC schematy przetwarzania potokowego mogą być efektywnie stosowane. Istnieje pewne zróżnicowanie czasu wykonywania rozkazów i jest możliwe przystosowanie potoku do tej sytuacji. Jak jednak widzieliśmy, zależność danych i rozgałęzianie redukują ogólną szybkość wykonywania.

W celu skompensowania tych zjawisk opracowano metody reorganizacji kodu. Rozważmy najpierw rozkazy rozgałęziania. Przy rozgałęzianiu opóźnionym jako metodzie zwiększania efektywności potoku używa się rozgałęzienia, które nie daje wyniku, aż do zakończenia następnego rozkazu. Ta dziwna procedura jest zilustrowana w tabeli 12.10. W pierwszej kolumnie widzimy zwyczajny, symboliczny rozkaz w języku maszynowym. Po wykonaniu rozkazu 102 następnym rozkazem przewidzianym do wykonania jest 105. W celu zapewnienia płynności potoku po tym rozkazie wstawiany jest rozkaz NOOP. Można jednak zwiększyć wydajność, zamieniając miejscami rozkazy 101 i 102. Wynik jest po-

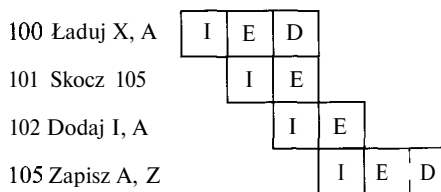
TABELA 12.10. Rozgałęzienie normalne i opóźnione

Adres	Rozgałęzienie normalne		Rozgałęzienie opóźnione		Zoptymalizowane rozgałęzienie opóźnione	
100	LOAD	X,A	LOAD	X,A	LOAD	X,A
101	ADD	I,A	ADD	I,A	JUMP	105
102	JUMP	105	JUMP	106	ADD	I,A
103	ADD	A,B	NOOP		ADD	A,B
104	SUB	C,B	ADD	A,B	SUB	C,B
105	STORE	A,Z	SUB	C,B	STORE	A,Z
106			STORE	A,Z		

(a)



(b)



RYSUNEK 12.10. Zastosowanie opóźnionego rozgałęziania: (a) wstawiony rozkaz pusty (NOOP); (b) rozkazy odwrócone

kazany na rys. 12.10. Przed rozkazem ADD (dodaj) pobierany jest rozkaz JUMP (skok). Zauważmy jednak, że rozkaz ADD jest pobierany, zanim wykonywany rozkaz JUMP ma szansę zmienić stan licznika programu. Wobec tego oryginalna semantyka programu jest zachowana.

Taka zamiana rozkazów jest skuteczna w przypadku rozgałęzień bezwarunkowych, wywołań i powrotów. W odniesieniu do rozgałęzień warunkowych procedura ta nie może być stosowana na ślepo. Jeśli warunek testowany w ramach rozgałęzienia może być zmieniony przez rozkaz bezpośrednio poprzedzający, to kompilator musi powstrzymać się od zamiany rozkazów, a zamiast tego wstawić rozkaz NOOP. Doświadczenie z systemami Berkeley RISC oraz IBM 801 wskazuje, że większość rozkazów rozgałęzienia warunkowego może być w ten sposób zoptymalizowana [PATT82a, RAD183].

Podobna taktyka, znana jako **opóźnione ładowanie**, może być zastosowana w odniesieniu do rozkazów LOAD (ładowania). W przypadku rozkazów LOAD rejestr docelowy tej operacji jest blokowany przez procesor. Następnie procesor kontynuuje wykonywanie strumienia rozkazów aż do osiągnięcia rozkazu wywołującego ładowanie tego rejestru, po czym pozostaje bezczynny do końca ładowania. Jeśli kompilator może tak zmienić porządek rozkazów, żeby podczas przebiegu ładowania w potoku mogła być wykonywana użyteczna praca, to efektywność wzrasta.

Na zakończenie zauważmy, że projektowanie potoku rozkazów nie może być prowadzone w oderwaniu od innych zabiegów optymalizacyjnych stosowanych w kompilatorze.

nych w systemie. W pracy [BRAD91b] wskazano na przykład, że szeregowanie rozkazów w potoku oraz dynamiczne przydzielanie rejestrów muszą być rozpatrywane łącznie, co umożliwia osiągnięcie największej efektywności.

12.6

MOTOROLA 88000

Pierwszym procesorem RISC firmy Motorola jest 88000. Choć jego numer mógłby wskazywać na pokrewieństwo z rodziną procesorów CISC MC68000, jednak różnice są znaczne. MC68000 jest przedstawicielem współczesnych procesorów CISC, podczas gdy 88000 można uznać za czysty przykład architektury RISC.

Lista rozkazów

W tabeli 12.11 są wymienione rozkazy procesora 88000, a na rys. 12.11 są pokazane formaty rozkazów. Podobnie jak praktycznie we wszystkich systemach RISC, w procesorze 88000 zastosowano stałą, wynoszącą 32 bity, długość rozkazów. Pierwsze 6 bitów tworzy kod operacji. W większości rozkazów po tym kodzie występuje 5-bitowe pole rejestru (wyniku i pierwszego argumentu). Pozostałe 16 bitów może zawierać argument natychmiastowy lub adres względny (przesunięcie) dla rozkazu rozgałęzienia warunkowego. W przypadku rozkazów angażujących 3 rejestry (2 źródłowe i 1 wynikowy lub docelowy) pozostałych 11 bitów przeznaczono na rozszerzenie kodu operacji w celu sprecyzowania określonych operacji.

Zauważmy, że we wszystkich formatach położenia głównych pól są zrównane. Dzięki temu układy logiczne dekodowania rozkazu procesora zawsze sięgają do rejestrów źródłowych i docelowych z tej samej lokacji względnej wewnątrz rozkazu. Przyspiesza to dekodowanie rozkazów i upraszcza projektowanie potoku.

Większość rozkazów odwołuje się tylko do argumentów zawartych w rejestrach. Tylko kilka prostych rozkazów ładowania i zapisu odwołuje się do pamięci. Bity rozszerzenia kodu operacji wskazują ładowanie i zapis słowa (32 bity), półsłowa i bajtu. W dwóch ostatnich przypadkach ładowanie do rejestru może być traktowane jako dotyczące liczb ze znakiem lub bez znaku. Liczby znakowane mają rozszerzony kod znaku aż do wypełnienia 32-bitowego rejestru docelowego w notacji uzupełnienia do dwóch. Liczby pozbawione znaku są uzupełniane zerami.

Wszelkie odniesienia do pamięci są tworzone przez dodanie zawartości rejestru podstawowego do drugiego argumentu. W celu zachowania prostoty i jednolitości operacji taki sam schemat logiczny, jaki jest używany w przypadku rozkazów arytmetycznych, jest również stosowany przy odnoszeniu się do pamięci-

TABELA 12.11. Lista rozkazów procesora Motorola 88000

Arytmetyka całkowitoliczbowa		Rozkazy dotyczące pól bitowych	
ADD	dodaj	CLR	wyczyść pole bitowe
ADDU	dodaj bez znaku	SET	wstaw pole bitowe
SUB	odejmij	EXT	wydział pole bitowe ze znakiem
SUBU	odejmij bez znaku	EXTU	wydział pole bitowe bez znaku
MUL	pomnóż	MAKE	utwórz pole bitowe
DIV	podziel	ROT	dokonaj rotacji pola bitowego
DIVU	podziel bez znaku	FFO	znajdź pierwszy wyczyszczony bit
CMP	porównaj	FFI	znajdź pierwszy ustawiony bit
Arytmetyka zmiennopozycyjna		Rozkazy sterowania przepływem	
FADD	dodaj zmiennopozycyjne	BBO	rozgałęziaj na bicie wyczyszczonym
FSUB	odejmij zmiennopozycyjne	BB I	rozgałęziaj na bicie ustawionym
FMUL	pomnóż zmiennopozycyjne	Bcnd	rozgałęziaj warunkowo
FDIV	podziel zmiennopozycyjne	BR	rozgałęziaj bezwarunkowo
FCMP	porównaj zmiennopozycyjne	BSR	rozgałęziaj do podprogramu
FLT	przekształć całkowitą na zmiennopozycyjną	JMP	skocz bezwarunkowo
INT	zaokrąglaj zmiennopozycyjną do całkowitej	JSR	skocz do podprogramu
NINT	zaokrąglaj zmiennopozycyjną do najbliższej całkowitej	TBO	pułapka na bicie wyczyszczonym
TRNC	zaokrąglaj zmiennopozycyjną w dół do całkowitej	TBI	pułapka na bicie ustawionym
FLDCR	ładuj ze zmiennopozycyjnego rejestru sterowania	TBND	pułapka na sprawdzaniu granicy
FSTCR	zapisz w zmiennopozycyjnym rejestrze sterowania	Tcnd	pułapka warunkowa
FXCR	wymień zmiennopozycyjny rejestr sterowania	RTE	powrót z wyjątku
Rozkazy logiczne		Rozkazy ładowania/zapisu/wymiany	
AND	I	LD	ładuj rejestr z pamięci
MASK	maskowanie logiczne argumentu natychmiastowego	LDA	ładuj adres
OR	LUB	LDCR	ładuj z rejestru sterowania
XOR	LUB wykluczające	ST	zapisz rejestr w pamięci
		STCR	zapisz w rejestrze sterowania
		XMEM	wymień rejestr z pamięcią
		XCR	wymień rejestr sterowania

ci. W przypadku dodawania liczb całkowitych drugi argument może być albo 16-bitową wielkością natychmiastową, albo zawartością rejestru 32-bitowego. Odpowiednimi trybami adresowania są: pośredni rejestrowy z natychmiastową wartością nie znakowaną oraz pośredni rejestrowy indeksowany. Ponadto przewidziano indeksowanie skalowane w celu wspierania przetwarzania tablic.¹

¹ Opis indeksowania skalowanego znajduje się w rozdz. 10 w części dotyczącej trybów adresowania Pentium.

6	5	5	11	5
Kod operacji	D	S1	Kod operacji	S2

6	5	5	6	5	5
Kod operacji	D	S1	Podkod operacji	Szerokość	Wyrównanie

6	5	5	16
Kod operacji	D	S1	Kod operacji

6	5	5	16
Kod operacji	M/B	S1	Wyrównanie

6	26
Kod operacji	Wyrównanie

D - rejestr przeznaczenia
 S1 - rejestr źródłowy 1
 S2 - rejestr źródłowy 2
 M/B - maska lub bit

RYSUNEK 12.11. Formaty rozkazu procesora Motorola 88000: (a) rozkazy rejestr-rejestr; (b) rozkazy związane z maską bitową; (c) rozkazy dotyczące argumentów natychmiastowych; (d) rozkazy skosu warunkowego; (e) rozkazy rozgałęzienia bezwarunkowego

Unikalnym aspektem listy rozkazów jest włączenie 8 rozkazów dotyczących pól bitowych, zdefiniowanych w tabeli 12.12. Pola, na których operują te rozkazy, mogą być dowolnej szerokości i mogą być zlokalizowane gdziekolwiek w ramach słowa. Układy (sprzętowe) pól bitowych mogą zerować, ustawiać, wyciągać i wstawiać pola do rejestrów. W zasadzie układy te mogą w jednym cyklu przesunąć dowolną liczbę bitów do pola o dowolnej szerokości. Jedynym ograniczeniem jest to, że suma wielkości przesunięcia plus szerokość zmienianego pola musi być mniejsza od szerokości rejestru 32-bitowego. Możliwość ta jest wygodna w przypadku takich operacji, jak obliczanie wykładnika liczby zmiennopozycyjnej lub danych Boole'a z rejestru zawierającego zespół znaczników stanu.

Architektura

Na rysunku 12.12 widać schemat blokowy architektury 88000. Główny układ procesora składa się z wielu niezależnych jednostek funkcyjnych połączonych z tablicą rejestrów o wielu portach. Jednostki funkcyjne mogą działać niezależnie i współbieżnie, umożliwiając bardzo efektywne wykonywanie rozkazów przetwarzania.

TABELA 12.12. Rozkazy dotyczące pól bitowych procesora Motorola 88000

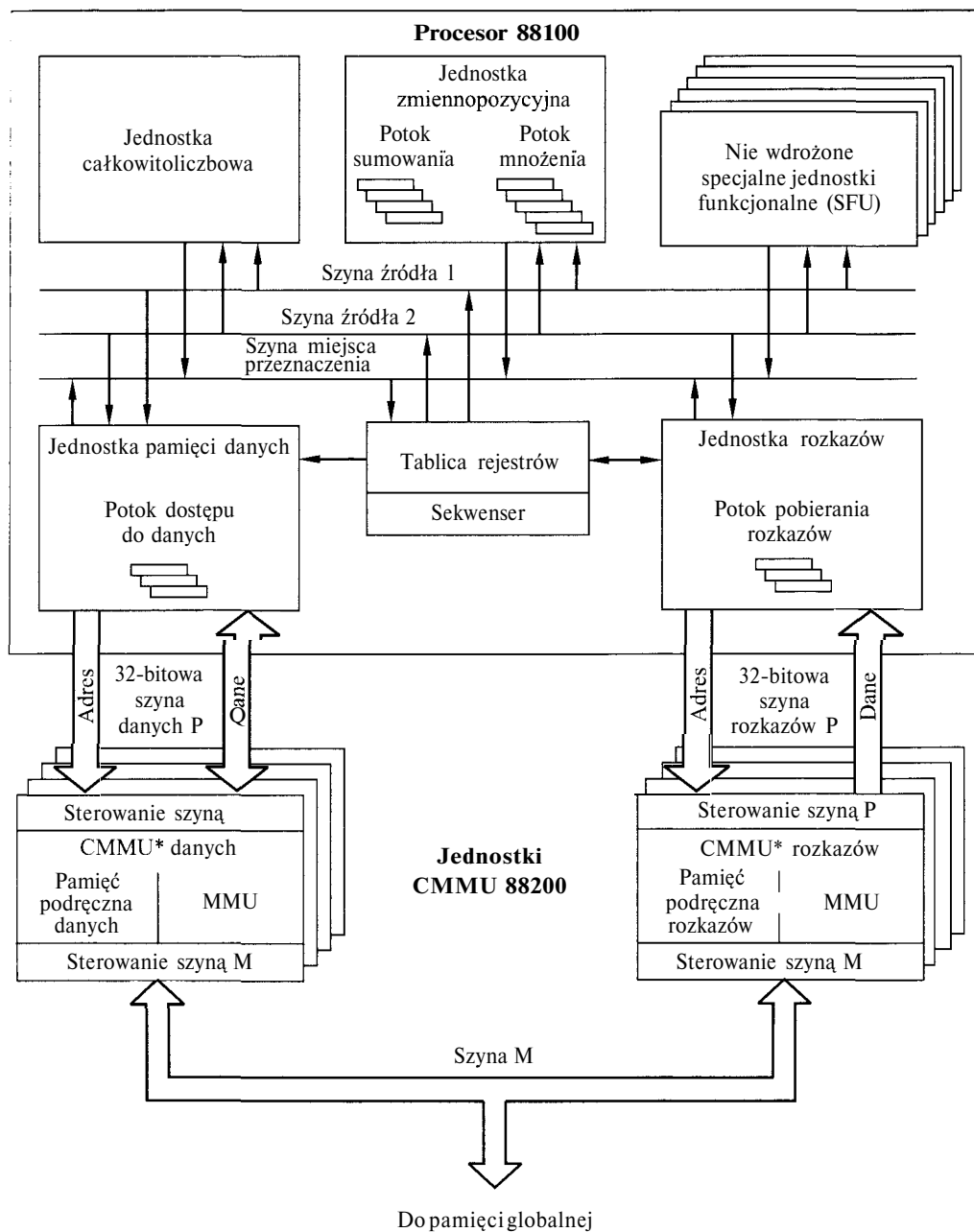
Nazwa	Rozkaz	Funkcja
CLR	zeruj pole bitowe	$D \leftarrow S; D[(o + w - 1) \dots o] \leftarrow 0s$
SET	ustaw jedynki w polu bitowym	$D \leftarrow S; D[(o + w - 1) \dots o] \leftarrow 1s$
EXT	wydziel pole bitowe	if ($w = 0$) then begin $D[31 \dots (32 - o)] \leftarrow S[31]; D[(31 - o) \dots 0] \leftarrow S[31 \dots o]$ end
EXTU	wydziel pole ze znakiem	else begin $D[31 \dots w] \leftarrow S[o + w - 1]; D[(w - 1) \dots 0] \leftarrow S[(o + w - 1) \dots o]$ end
	bitowe	if ($w = 0$) then begin $D[31 \dots (32 - o)] \leftarrow 0; D[(31 - o) \dots 0] \leftarrow S[31 \dots o]$ end
	bez znaku	else begin $D[31 \dots w] \leftarrow 0; D[(w - 1) \dots 0] \leftarrow S[(o + w - 1) \dots o]$ end
MAK	utwórz pole	if ($w = 0$) then $D \leftarrow \text{shiftL}(S, o)$
	bitowe	else begin $D \leftarrow 0; D[(o + w - 1) \dots o] \leftarrow S[(w - 1) \dots (0)]$ end
ROT	dokonaj rotacji pola bitowego	$D \leftarrow \text{rotateR}(S, o)$
FFO	w prawo znajdź	$i \leftarrow 31$
	pierwszy bit	while ($i \geq 0$ and $S2[i] = 1$) do $i \leftarrow i - 1$;
	zerowy	if ($i < 0$) then $D \leftarrow 32$ else $D \leftarrow i$;
FFI	znajdź	$i \leftarrow 31$
	pierwszy bit	while ($i \geq 0$ and $S2[i] = 0$) do $i \leftarrow i - 1$;
	jedynkowy	if ($i < 0$) then $D \leftarrow 32$ else $D \leftarrow i$;

D - rejestr docelowy, o - wyrównanie (*offset*), S - rejestr źródłowy, w - szerokość

- *Jednostka całkowitoliczbowa.* Wykonuje wszystkie operacje arytmetyczne na liczbach całkowitych, operacje na polach bitowych, operacje boolowskie oraz dostępu do rejestru sterowania.
- Q *Jednostka zmiennopozycyjna.* Wykonuje funkcje zgodne z IEEE 754. Składa się z 5-etapowego potoku sumatora i oddzielnego 6-etapowego potoku mnożnika. Potoki te umożliwiają współbieżne wykonywanie wielu operacji zmiennopozycyjnych.
- a *Jednostka rozkazów.* Odpowiedzialna za pobieranie rozkazów i rozprowadzanie zdekodowanego kodu operacji w postaci sygnałów sterujących do odpowiednich jednostek wykonujących zawartych w procesorze.
- a *Jednostka pamięci danych.* Odpowiedzialna za ładowanie i zapisywanie argumentów między procesorem a pamięcią zewnętrzną.

Magistrala pamięci (*M bus*) łączy dwie jednostki zarządzania pamięcią podłączoną z systemem pamięci. Jedna z nich dotyczy danych, druga zaś rozkazów. Takie właśnie rozwiązanie jest wdrażane w wielu współczesnych systemach RISC. Architektura taka pozwala na pobieranie rozkazów równocześnie z przesyłaniem argumentów między procesorem a pamięcią, co umożliwia zwiększenie szybkości².

² Producenci określają zwykle tę architekturę jako **architekturę harwardzką**. Jednak oryginalne zastosowanie tego terminu dotyczyło oddzielenia rozkazów i danych w pamięci głównej, a nie w podręcznej.



RYSUNEK 12.12. Schemat blokowy systemu 88000

Zarządzanie rejestrami

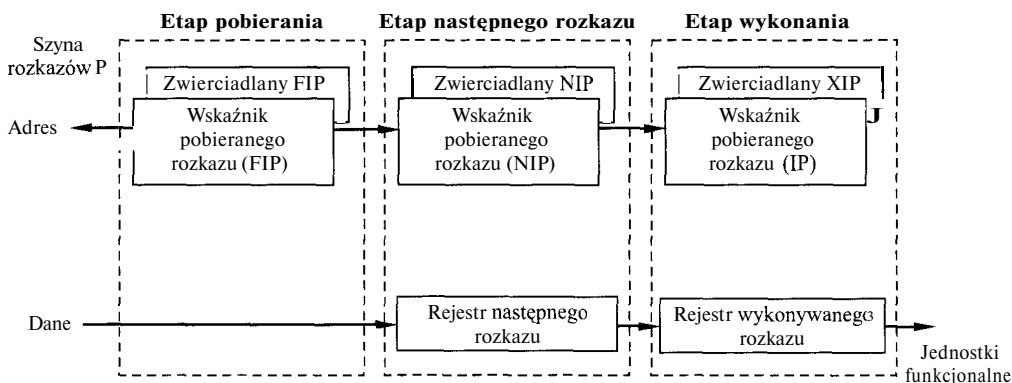
Tablica rejestrów składa się z 32 32-bitowych rejestrów ogólnego przeznaczenia (roboczych). Rejestr RO jest sprzętowo wypełniony zerami. Jest to wygodne w przypadku zapisywania zer w innych rejestrach lub w pamięci. Rejestr R1 jest używany do automatycznego zapisywania adresów powrotnych w przypadku rozkazów rozgałęziania lub skoku do podprogramu.

W odniesieniu do architektury 88000 potrzebne są pewne środki umożliwiające wszystkim jednostkom funkcyjnym wspólne używanie tablicy rejestrów bez niszczenia danych pozostających w rejestrach. Mechanizm zastosowany do tego celu w 88000 nazywa się **rejestrówą tablicą wyników** (ang. *register scoreboarding*). Procesor zawiera rejestr - tablicę wyników - zawierający po jednym bicie odpowiadającym każdemu z 31 rejestrów (RO jest wyłączony, ponieważ nie jest on aktualizowany). Za każdym razem, gdy jednostka rozkazów wysłała rozkaz, którego wykonanie zabiera więcej niż 1 cykl zegara, ustawiany jest bit w rejestrze wyników odpowiadający rejestrowi docelowemu tego rozkazu. Wszystkie jednostki funkcyjne mają swobodę niezależnego dostępu do rejestrów i korzystają z niej, dopóki nie muszą pobrać danych z rejestru, którego bit jest ustawiony. Wówczas jednostka funkcyjna musi czekać, aż pożądaný rejestr zostanie uwolniony.

Potok jednostki rozkazów

Każda z jednostek funkcyjnych 88000 ma strukturę potokową w celu zapewnienia możliwie wysokiego stopnia równoległości. Na rysunku 12.13 jest pokazany 3-etapowy potok w jednostce rozkazów. Jednostka pobiera rozkaz z pamięci podręcznej rozkazów za pośrednictwem magistrali określanej jako PBUS.

Etap pobierania obejmuje rejestr nazywany wskaźnikiem pobieranego rozkazu (ang. *Fetch Instruction Pointer - FIP*), który zawiera adres rozkazu przewi-



RYСУNEK 12.13. Potok jednostki rozkazów procesora Motorola 88000

dzianego do pobrania. Na początku każdego cyklu zegara, jeśli nie występują stany zatrzymania potoku lub oczekiwania na pamięć, FIP przekazuje adres do pamięci podręcznej rozkazów.

Podczas następnego etapu adres zawarty w rejestrze FIP jest przenoszony do rejestru określanego jako wskaźnik następnego rozkazu (ang. *Next Instruction Pointer* - NIP), a rozkaz jest pobierany z pamięci i umieszczany w rejestrze następnego rozkazu. Na tym etapie rozkaz jest częściowo dekodowany, a także następuje wyprzedzające pobranie potrzebnych argumentów z tablicy rejestrów i przygotowanie ich do przesłania do odpowiedniej jednostki funkcyjnej.

Podczas etapu wykonywania adres zawarty w rejestrze NIP jest przenoszony do rejestru zwanego wskaźnikiem wykonywanego rozkazu (ang. *Execute Instruction Pointer* - XIP), rozkaz zaś trafia do rejestru wykonywanego rozkazu. Na tym etapie rozkaz jest kierowany do odpowiedniej jednostki funkcyjnej.

Dla każdego wskaźnika rozkazu utrzymywany jest **rejestr lustrzany** (ang. *shadow register*). Jeśli podczas cyklu występuje wyjątek, rejestry lustrzane są zamrażane w celu zachowania wartości w czasie trwania wyjątku. Wartości te są przywracane po przetworzeniu wyjątku.

12.7

MIPS R4000

Jeden z pierwszych dostępnych na rynku zestawów mikroukładów został opracowany w MIPS Computer Systems. Inspiracją dla niego był system eksperymentalny również noszący nazwę MIPS, opracowany w Stanfordzie [HENN84]. Najnowszym członkiem rodziny MIPS jest procesor R4000. Ma on zasadniczo tę samą architekturę i listę rozkazów, co wcześniejsze modele MIPS: R2000, R3000 i R6000. Najbardziej znaczącą różnicą jest to, że w R4000 wszystkie wewnętrzne i zewnętrzne ścieżki danych, a także adresy, rejestry i ALU są 64-bitowe.

Zastosowanie architektury 64-bitowej zamiast 32-bitowej ma wiele zalet. Większa jest przestrzeń adresowa - wystarczająco, aby system operacyjny odwzorował ponad terabajt plików bezpośrednio w pamięci wirtualnej w celu ułatwienia dostępu. Ponieważ obecnie powszechne są napędy dyskowe o pojemności 1 GB i większej, 4-gigabajtowa przestrzeń adresowa maszyny 32-bitowej staje się ograniczeniem. Dzięki architekturze 64-bitowej procesor R4000 może przetwarzać za jednym razem takie dane, jak liczby zmiennopozycyjne IEEE o pojedynczej dokładności lub łańcuchy znaków obejmujące do 8 znaków.

Mikroukład procesora R4000 składa się z dwóch sekcji, z których jedna zawiera procesor a druga koprocesor do zarządzania pamięcią. Procesor ma bardzo prostą architekturę. Intencją było zaprojektowanie systemu, w którym układy logiczne wykonywania rozkazów byłyby maksymalnie proste, co pozostawiałoby

TABELA 12.13. Lista rozkazów procesorów MIPS serii R

Operacja	Opis	Operacja	Opis
Rozkazy ładowania/zapisu		Rozkazy mnożenia/dzielenia	
LB	ładuj bajt	MULT	pomnóż
LBU	ładuj bajt bez znaku	MULTU	pomnóż bez znaku
LH	ładuj półsłowo	DIV	podziel
LHU	ładuj półsłowo bez znaku	DIVU	podziel bez znaku
LW	ładuj słowo	MFHI	przenieś z HI
LWL	ładuj słowo w lewo	MTHI	przenieś do HI
LWR	ładuj słowo w prawo	MFLO	przenieś z LO
SB	zapisz bajt	MTLO	przenieś do LO
SH	zapisz półsłowo	Rozkazy skoku i rozgałęzienia	
SW	zapisz słowo	J	skocz
SWL	zapisz słowo w lewo	JAL	skocz i połącz
SWR	zapisz słowo w prawo	JR	skocz do rejestru
Rozkazy arytmetyczne (argument natychmiastowy w ALU)		JALR	skocz i połącz rejestr
ADDI	dodaj natychmiastowy	BEQ	rozgałęziaj, gdy równe
ADDIU	dodaj natychmiastowy bez znaku	BNE	rozgałęziaj, gdy nie równe
SLTI	ustaw na mniej niż natychmiastowy	BLEZ	rozgałęziaj, gdy mniejsza lub równa zera
SLTIU	ustaw na mniej niż natychmiastowy bez znaku	BGTZ	rozgałęziaj, gdy większa od zera
ANDI	AND z argumentem natychmiastowym	BLTZ	rozgałęziaj, gdy mniejsza od zera
ORI	OR z argumentem natychmiastowym	BGEZ	rozgałęziaj, gdy większa lub równa zera
XORI	Exclusive-OR z argumentem natychmiastowym	BLTZAL	rozgałęziaj, gdy mniejsza od zera i łącznika
LUI	ładuj argument natychmiastowy do górnej połowy rejestru	BGEZAL	rozgałęziaj, gdy większa lub równa zera i łącznikowi
Rozkazy arytmetyczne (3-argumentowe, typu R)		Rozkazy dotyczące koprocatora	
ADD	dodaj	LWCz	ładuj słowo do koprocatora
ADDU	dodaj bez znaku	SWCz	zapisz słowo w koprocatorze
SUB	odejmij	MTCz	przenieś do koprocatora
SUBU	odejmij bez znaku	MFCz	przenieś z koprocatora
SLT	ustaw na mniej niż	CTCz	przenieś sterowanie do koprocatora
SLTU	ustaw na mniej niż bez znaku	CFCz	przenieś sterowanie z koprocatora
AND	wykonaj AND	COPz	operacja koprocatora
OR	wykonaj OR	BCzT	rozgałęziaj, gdy „z” koprocatora prawdziwe
XOR	wykonaj XOR	BCzF	rozgałęziaj, gdy „z” koprocatora fałszywe
NOR	wykonaj NOR	Rozkazy specjalne	
Rozkazy przesunięcia		SYSCALL	wywołanie systemowe
SLL	przesuń logicznie w lewo	BREAK	przerwanie
SRL	przesuń logicznie w prawo		
SRA	przesuń arytmetycznie w prawo		
SLLV	przesuń zmienną logicznie w lewo		
SRLV	przesuń zmienną logicznie w prawo		
SRAV	przesuń zmienną arytmetycznie w prawo		

miejsce na układy logiczne zwiększające wydajność (np. na całą jednostkę zarządzania pamięcią).

Procesor wykorzystuje 32 rejestry 64-bitowe. Umożliwia również obsługę do 128 KB szybkiej pamięci podręcznej, w połowie przeznaczonych na rozkazy, a w połowie na dane. Stosunkowo duża pamięć podręczna (IBM 3090 obsługuje 128÷256 KB pamięci podręcznej) pozwala systemowi na trzymanie dużych zbiorów kodów programu i danych lokalnych wobec procesora, rozładowywanie magistrali pamięci głównej oraz na uniknięcie dużej tablicy rejestrów i towarzyszących jej układów logicznych okienkowania.

Lista rozkazów

W tabeli 12.13 są wymienione podstawowe rozkazy wszystkich procesorów MIPS serii R, a w tabeli 12.14 są podane dodatkowe rozkazy zaimplementowane w R4000. Wszystkie rozkazy procesora są zakodowane w jednym formacie 32-bitowym. Wszystkie operacje na danych są typu rejestr-rejestr; jedynie odniesienia do pamięci są czystymi operacjami ładowania/zapisu.

Procesor R4000 nie używa kodów warunkowych. Jeśli rozkaz generuje warunek, to odpowiednie znaczniki stanu są przechowywane w rejestrze ogólnego przeznaczenia. Dzięki temu uniknięto specjalnych układów logicznych zajmujących się kodami warunkowymi, które oddziałują na mechanizm przetwarzania potokowego oraz zmienianie kolejności rozkazów przez kompilator. Zamiast tego wykorzystano już istniejący mechanizm wdrożony w celu zajmowania się zależnościami danych rejestrowych. Ponadto, warunki odwzorowane w tablicach rejestrów podlegają takiej samej optymalizacji w czasie kompilowania obejmującej rozmieszczanie i powtarzanie użycie, jak inne wartości przechowywane w rejestrach.

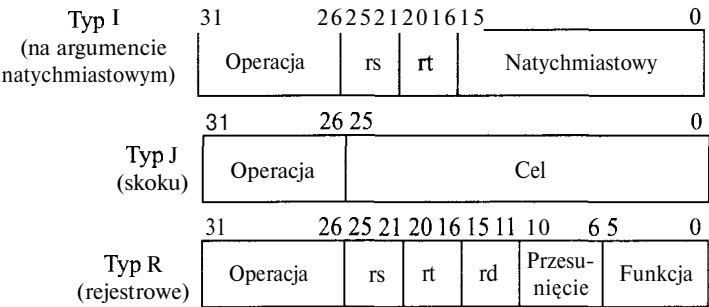
Podobnie jak większość maszyn RISC, procesor MIPS używa rozkazu o pojedynczej, 32-bitowej długości. Upraszcza to pobieranie i dekodowanie rozkazów, a także upraszcza współpracę z jednostką zarządzania pamięcią wirtualną podczas pobierania rozkazów (rozkazy nie przekraczają granic słów lub stron). Zastosowano trzy formaty rozkazów (rys. 12.14) ze wspólnym formatowaniem kodów operacji i odniesień do rejestrów, co upraszcza dekodowanie rozkazów. Wyniki bardziej złożonych rozkazów mogą być syntetyzowane w czasie kompilacji.

Sprzętowo zaimplementowano jedynie najprostszy i najczęściej używany tryb adresowania. Wszystkie odniesienia do pamięci obejmują 16-bitowe wyrównanie w stosunku do rejestru 32-bitowego. Na przykład rozkaz „ładuj słowo” ma postać:

lw r2, 128(r3) ładuj słowo zawarte pod adresem 128 względem rejestru 3 do rejestru 2

TABELA 12.14. Dodatkowe rozkazy R4000

Operacja	Opis	Operacja	Opis
Rozkazy ładowania/zapisu		Rozkazy wyjątku	
LL	ładuj połączony	TGE	przerwij (<i>trap</i>), jeśli większa niż lub równa
S C	zapisz warunkowy	TGEU	przerwij, jeśli większa niż lub równa bez znaku
SYNC	synchronizacja	TLT	przerwij, jeśli mniejsza niż
Rozkazy skoku i rozgałęzienia		TLTU	przerwij, jeśli mniejsza niż bez znaku
BEQL	rozgałęziaj, gdy prawdopodobnie równa	TEQ	przerwij, jeśli równa
BNEL	rozgałęziaj, gdy prawdopodobnie nie równa	TNE	przerwij, jeśli nie równa
BLEZL	rozgałęziaj, gdy prawdopodobnie mniejsza niż lub równa zeru	TGEI	przerwij, jeśli większa lub równa natychmiastowemu
BGTZL	rozgałęziaj, gdy prawdopodobnie większa niż zero	TGEIU	przerwij, jeśli większa lub równa natychmiastowemu bez znaku
BLTZL	rozgałęziaj, gdy prawdopodobnie mniejsza od lub równa zeru	TLTI	przerwij, jeśli mniejsza niż natychmiastowy
BGEZL	rozgałęziaj, gdy prawdopodobnie większa od lub równa zeru	TLTIU	przerwij, jeśli mniejsza niż natychmiastowy bez znaku
BLTZALL	rozgałęziaj, gdy prawdopodobnie mniejsza od zera i łącznika	TEQI	przerwij, jeśli równa natychmiastowemu
BGEZALL	rozgałęziaj, gdy prawdopodobnie większa lub równa od zera i łącznika	TNEI	przerwij, jeśli nie równa natychmiastowemu
BCzTL	rozgałęziaj, gdy prawdopodobnie „z” koprocatora prawdziwe	Rozkazy dotyczące koprocatora	
CDzFL	rozgałęziaj, gdy prawdopodobnie „z” koprocatora fałszywe	LDCz	ładuj podwójne do koprocatora
		SDCz	zapisz podwójne w koprocatorze



Operacja	6-bitowy kod operacji
rs	5-bitowy specyfikator rejestru źródłowego
rt	5-bitowy specyfikator rejestru źródłowego/rejestru przeznaczenia
Natychmiastowy	16-bitowy argument natychmiastowy, rozgałęzienie lub przemieszczenie adresu
Cel	26-bitowy adres celu skoku
rd	5-bitowy specyfikator rejestru przeznaczenia
Przesunięcie	5-bitowa wielkość przesunięcia
Funkcja	6-bitowy specyfikator funkcji ALU/przesunięcie

RSUNEK 12.14. Formaty rozkazów MIPS

Każdy z 32 rejestrów ogólnego przeznaczenia może być użyty jako rejestr podstawowy. Jeden rejestr, a mianowicie r0, zawsze zawiera zera.

Kompilator używa wielu rozkazów maszynowych w celu syntetyzowania trybów adresowania typowych dla maszyn konwencjonalnych. Pewne przykłady zawiera tabela 12.15 [CHOW87]. Pokazano w niej użycie rozkazu lui (*load upper immediate*). Rozkaz ten powoduje załadowanie 16-bitowej wartości natychmiastowej do górnej połowy rejestru i wyzerowanie dolnej połowy.

TABELA 12.15. Synteza pozostałych trybów adresowania za pomocą trybu adresowania MIPS

Rozkaz jawny	Rozkaz rzeczywisty
lw r2, <16-bitowe wyrównanie>	lw r2, <16-bitowe wyrównanie> (r0)
lw r2, <32-bitowe wyrównanie>	lui r1, < 16 wyższych bitów wyrównania> lw r2, < 16 niższych bitów wyrównania> (r1)
lw r2, <32-bitowe wyrównanie> (r4)	lui r1, < 16 wyższych bitów wyrównania> addu r1, r1, r4 lw r2, < 16 niższych bitów wyrównania> (r1)

Potok rozkazów

Dzięki uproszczonej architekturze rozkazów procesor MIPS może osiągać bardzo efektywne przetwarzanie potokowe. Zapoznanie się z ewolucją przetwarzania potokowego procesorów MIPS jest pouczające, ponieważ ilustruje ona ogólną ewolucję przetwarzania potokowego w architekturze RISC.

Pierwsze, doświadczalne systemy RISC i pierwsza generacja komercyjnych procesorów RISC osiągały szybkość wykonywania sięgającą jednego rozkazu w jednym cyklu zegara systemowego. Aby zwiększyć tę wydajność, opracowano dwie klasy procesorów, umożliwiające wykonywanie wielu rozkazów w cyklu zegara: architektury superskalarną i superpotokową. W architekturze superskalarnej jest w zasadzie powielany każdy etap potoku, dzięki czemu na tym samym etapie potoku może być jednocześnie przetwarzanych dwa lub więcej rozkazów. Natomiast w architekturze superpotokowej stosuje się większą liczbę bardziej rozdrobnionych etapów potokowania. Dzięki większej liczbie etapów w tym samym czasie może się znajdować w potoku więcej rozkazów, co zwiększa równoległość.

Oba rozwiązania mają swoje ograniczenia. W przypadku superskalarnego przetwarzania potokowego zależności między rozkazami w różnych potokach mogą spowolnić system. Wymagane są także dodatkowe układy logiczne do koordynowania tych zależności. W przypadku przetwarzania superpotokowego występują straty związane z przenoszeniem rozkazów z etapu do etapu.

Analizie architektury superskalarnej jest poświęcony rozdz. 13. Procesor MIPS R4000 jest dobrym przykładem architektury superpotokowej RISC.

Na rysunku 12.15a są pokazane potoki rozkazów procesora R3000. W tym procesorze potok przesuwany jest raz w ciągu jednego cyklu zegara. Kompilator MIPS może zmienić kolejność rozkazów w celu wypełnienia kodami przerwy wynikających z opóźnień na poziomie 70÷90% czasu. Wszystkie rozkazy są przetwarzane w tej samej sekwencji pięciu etapów potoku. Etapy te to:

- pobranie rozkazu,
- a pobranie argumentu źródłowego z tablicy rejestrów,
- a operacja ALU lub generowanie adresu argumentu,
- a odniesienie do danych w pamięci,
- zapis w tablicy rejestrów.

Jak widać na rys. 12.15a, występuje tutaj nie tylko równoległość wynikająca z przetwarzania potokowego, ale także równoległość w ramach wykonywania pojedynczego rozkazu. Cykl zegara o długości 60 ns jest podzielony na dwie fazy

(a) Cykl zegarowy

ϕ_1	ϕ_2	ϕ_1	ϕ_2	ϕ_1	ϕ_2	ϕ_1	ϕ_2	ϕ_1	ϕ_2
IF		RD		ALU		MEM		WB	

	Pamięć podręczna rozkazów	RF	ALU OP		Pamięć podręczna danych	WB
ITBL		IDEC	DA	DTLB		
		IA			IF - pobieranie instrukcji	RD - odczyt danych

IF - pobieranie rozkazów

RD - odczyt

MEM - dostęp do pamięci

WB - zapis opóźniony

Pamięć podręczna rozkazów - dostęp do pamięci podręcznej rozkazów

RF - pobranie argumentu z rejestru

Pamięć podręczna danych - dostęp do pamięci podręcznej danych

ITBL - dostęp do bufora translacji

adresów tablic stron rozkazów

IDEC - dekodowanie rozkazu

IA - obliczanie adresu rozkazu

DA - obliczanie wirtualnego adresu danych

DTLB - dostęp do bufora translacji adresów tablic stron danych

TC - sprawdzenie znacznika pamięci podręcznej danych

(b)

Cykl	Cykl	Cykl	Cykl	Cykl	Cykl
ITLB	Pamięć podręczna rozkazów	RF	ALU	DTLB	Pamięć podręczna danych
					WB

(c)

Cykl	Cykl	Cykl	Cykl	Cykl
Pamięć podręczna rozkazów	RF	ALU	Pamięć podręczna danych	TC
				WB

RYSUNEK 12.15. Wzbogacenie potoku R3000: (a) uszczegółowiony potok R3000; (b) zmodyfikowany potok R3000 o zredukowanych czasach oczekiwania; (c) zoptymalizowany potok R3000 z równoległym dostępem do TLB (bufora translacji adresów tablic stron) oraz do pamięci podręcznej

30-nanosekundowe. Operacje dostępu do rozkazu zewnętrznego i do danych w pamięci podręcznej wymagają (każda z nich) 60 ns, podobnie jak główne operacje wewnętrzne (OP, DA, IA). Dekodowanie rozkazu jest operacją prostszą, wymagającą tylko jednej fazy 30 ns, nakładającej się z pobraniem zawartości rejestru w ramach tego samego rozkazu. Obliczanie adresu w ramach rozkazu rozgałęzienia również nakłada się z dekodowaniem rozkazu i pobraniem zawartości rejestru, tak więc rozgałęzienie przy rozkazie i może adresować dostęp ICACHE do rozkazu $i + 2$. Podobnie ładowanie rozkazu i powoduje pobranie danych, które są natychmiast używane przez kod operacji rozkazu $i + 1$, podczas gdy wynik operacji ALU lub przesunięcia jest bez opóźnienia kierowany do rozkazu $i + 1$. To ściśle sprzężenie między rozkazami umożliwia wysoce wydajne przetwarzanie potokowe.

Fazy, na które dzielony jest cykl zegara, oznaczmy przez $\Phi 1$ i $\Phi 2$. Funkcje realizowane w każdej fazie są zebrane w tabeli 12.16.

TABELA 12.16. Etapy potoku R3000

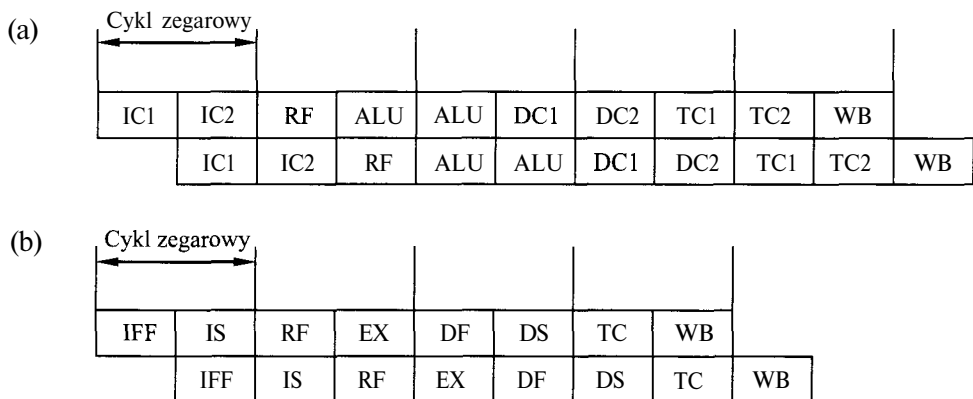
Etap potoku	Faza	Działanie
IF	$\Phi 1$	Używając TLB, przekształć adres wirtualny rozkazu na adres fizyczny (po decyzji rozgałęzienia)
IF	$\Phi 2$	Wyślij adres fizyczny do (rejestru) adresu rozkazu
RD	$\Phi 1$	Spowoduj powrót rozkazu z pamięci podręcznej rozkazów
RD	$\Phi 2$	Porównaj znaczniki i ważność pobranego rozkazu
RD	$\Phi 2$	Dekoduj rozkaz
RD	$\Phi 2$	Odczytaj tablicę rejestrów
RD	$\Phi 2$	Jeśli następuje rozgałęzienie, oblicz adres docelowy rozgałęzienia
ALU	$\Phi 1 + \Phi 2$	Jeśli następuje operacja rejestr-rejestr, wykonywana jest operacja arytmetyczna lub logiczna
ALU	$\Phi 1$	Jeśli następuje rozgałęzienie, decyduj, czy jest ono wykonywane.
ALU	$\Phi 1$	Jeśli następuje odniesienie do pamięci (ładowanie lub zapis), oblicz wirtualny adres danych
ALU	$\Phi 2$	Jeśli następuje odniesienie do pamięci, przekształć adres wirtualny danych na adres fizyczny posługując się TLB
MEM	$\Phi 1$	Jeśli następuje odniesienie do pamięci, wyślij adres fizyczny do pamięci podręcznej danych
MEM	$\Phi 2$	Jeśli następuje odniesienie do pamięci, spowoduj powrót danych z pamięci podręcznej danych i sprawdź znaczniki
WB	$\Phi 1$	Zapisz w tablicy rejestrów

Procesor R4000 zawiera wiele udoskonaleń technicznych w porównaniu z R3000. Zastosowanie bardziej zaawansowanej technologii pozwala na skrócenie cyklu zegara do połowy, tzn. do 30 ns, oraz na podobne skrócenie czasu dostępu do tablicy rejestrów. Ponadto większa jest gęstość upakowania elementów w mikroukładzie, co pozwoliło na włączenie do tego samego mikroukładu pamięci podręcznych rozkazów i danych. Zanim przeanalizujemy przetwarzanie potokowe w procesorze R4000, rozważmy, jak można poprawić wydajność potoku procesora R3000 za pomocą techniki wykorzystywanej w R4000.

Pierwszy krok jest pokazany na rys. 12.15b. Pamiętajmy, że cykle na tym rysunku zajmują połowę czasu cykli z rys. 12.15a. Dzięki włączeniu do tego samego mikroukładu wszystkich modułów procesora, etapy pamięci podręcznych rozkazów i danych są krótsze o połowę; mogą więc one nadal zajmować tylko jeden cykl zegara. Podobnie, ponieważ przyspieszono dostęp do tablicy rejestrów, zapis i odczyt w rejestrze nadal zajmują połowę cyklu zegara.

Ponieważ pamięci podręczne procesora R4000 zostały scalone w tym samym mikroukładzie, proces tłumaczenia adresu wirtualnego na fizyczny może opóźniać dostęp do pamięci podręcznej. Opóźnienie to jest redukowane przez wdrożenie wirtualnie indeksowanych pamięci podręcznych oraz wprowadzenie równoległego dostępu do pamięci podręcznej i translacji adresu. Na rysunku 12.15c jest pokazany zoptymalizowany potok R3000 z wyżej opisanymi ulepszeniami. Ze względu na kompresję zdarzeń sprawdzanie wskaźników pamięci podręcznej danych jest prowadzone oddzielnie w cyklu następującym po dostępie do pamięci podręcznej.

W systemie superpotokowym te same układy elektroniczne są używane kilkakrotnie w czasie jednego cyklu dzięki wstawieniu rejestrów potoku rozszczepiających każdy etap. W zasadzie każdy etap superpotoku przebiega z wielokrotnością częstotliwości zegara, przy czym wielokrotność zależy od stopnia superpotokowania. Technologia R4000 dysponuje szybkością i gęstością, które umożliwiają superpotokowanie w stopniu 2. Na rysunku 12.16a jest pokazany zoptymalizowany potok R3000 wykorzystujący to superpotokowanie. Zauważmy, że jest to w zasadzie taka sama struktura dynamiczna, jak na rys. 12.15c.



IFF - pobieranie rozkazu - pierwsza połowa
 IS - pobieranie rozkazu - druga połowa
 RF - pobranie argumentu z rejestru
 EX - wykonywanie rozkazów
 IC - pamięć podręczna rozkazów

DC - pamięć podręczna danych
 DF - pamięć podręczna danych - pierwsza połowa
 DS - pamięć podręczna danych - druga połowa
 TC - sprawdzenie znacznika

RYSUNEK 12.16. Teoretyczny superpotok R3000 i rzeczywisty superpotok R4000: (a) superpotokowe wdrożenie zoptymalizowanego potoku R3000; (b) potok R4000

Mogą być poczynione dalsze ulepszenia. Dla procesora R4000 zaprojektowano o wiele większy i wyspecjalizowany sumator. Umożliwia to prowadzenie operacji ALU z 2-krotnie większą szybkością. Inne ulepszenia pozwalają na 2-krotne przyspieszenie operacji ładowania i zapisu. Otrzymany w wyniku tych zabiegów potok jest pokazany na rys. 12.16b.

Potok procesora R4000 ma 8 etapów, co oznacza, że w potoku może się jednocześnie znajdować do 8 rozkazów. Potok przesuwa się z szybkością 2 etapów w jednym cyklu zegara. Etapy potoku są następujące:

- Q *Pobieranie rozkazu — pierwsza połowa.* Adres wirtualny jest przekazywany do pamięci podręcznej rozkazów oraz do bufora translacji adresów TLB (ang. *translation lookaside buffer*).
- *Pobieranie rozkazu - druga połowa.* Pamięć podręczna rozkazu przekazuje rozkaz, a w TLB jest generowany adres fizyczny.
- a *Tablica rejestrów.* Równolegle następują trzy działania:
 - dekodowany jest rozkaz i dokonywane jest sprawdzenie warunków współzależności (tzn. sprawdzenie, czy rozkaz ten zależy od wyniku poprzedniego rozkazu);
 - sprawdzany jest wskaźnik pamięci podręcznej rozkazów;
 - z tablicy rejestrów pobierane są argumenty.
- *Wykonywanie rozkazu.* Może występować jedno z trzech działań:
 - jeśli rozkaz dotyczy operacji rejestr-rejestr, to ALU wykonuje operację arytmetyczną lub logiczną;
 - jeśli rozkaz jest rozkazem ładowania lub zapisu, to jest obliczany wirtualny adres danych;
 - jeśli rozkaz jest rozgałęzieniem, to jest obliczany wirtualny adres docelowy i sprawdzane są warunki rozgałęzienia.
- Q *Pierwszy etap dostępu do pamięci podręcznej danych.* Adres wirtualny jest przekazywany do pamięci podręcznej danych oraz do TLB.
- *Drugi etap dostępu do pamięci podręcznej danych.* Pamięć podręczna danych przekazuje dane, a w TLB jest generowany adres fizyczny.
- a *Sprawdzenie wskaźnika.* W przypadku operacji ładowania i zapisu sprawdzany jest wskaźnik pamięci podręcznej.
- Q *Zapis.* Wynik jest zapisywany w tablicy rejestrów.

12.8

PORÓWNANIE ARCHITEKTUR RISC I CISC

Przez wiele lat główną tendencją objawiającą się w rozwoju architektury i organizacji komputerów było zwiększanie złożoności procesora: więcej rozkazów, więcej trybów adresowania, więcej wyspecjalizowanych rejestrów i tak dalej. Rozwój architektury RISC oznaczał całkowite odejście od filozofii leżącej u podstaw tej tendencji. Naturalnie pojawienie się systemów RISC i publikacji

wysławiających cnoty RISC spowodowało reakcję ze strony tego, co można określić jako główny nurt architektury komputerów.

Prace nad oceną zalet rozwiązania RISC można podzielić na dwie kategorie:

- a *Ilościowe*. Dążenie do porównania rozmiaru programów i szybkości ich realizacji na maszynach RISC i CISC wykonanych z wykorzystaniem porównywalnej technologii.
- a *Jakościowe*. Badanie zagadnień takich, jak wspieranie języków wysokiego poziomu i optymalne wykorzystywanie możliwości VLSI.

Większość prac nad oceną ilościową została wykonana przez zespoły pracujące nad systemami RISC [PATT82b, HEAT84, PATT84]; prace te w znacznej większości faworyzowały rozwiązanie RISC. Inni badali to zagadnienie i pozostali nieprzekonani [COLW85a, FLYN87, DAVI87]. Istnieje kilka problemów z przeprowadzeniem takich porównań [SERL86]:

- a Nie ma pary maszyn RISC i CISC, które są porównywalne pod względem całkowitych kosztów, poziomu technologii, stopnia scalenia, złożoności kompilatora, wsparcia systemu operacyjnego itd.
- Nie istnieje rozstrzygający zbiór testów programów. Wydajność zależy od programu.
- a Trudne jest oddzielenie wpływów sprzętu od wyników związanych z biegłością twórców kompilatorów.
- a Większość analiz porównawczych dotyczących architektury RISC była wykonana raczej na maszynach-zabawkach niż na wyrobach komercyjnych. Ponadto większość maszyn dostępnych na rynku i reklamowanych jako RISC ma zarówno cechy architektury RISC, jak i CISC. Dlatego uczciwe porównanie z komercyjną maszyną o czystej architekturze CISC (np. VAX, Intel 80386) jest trudne.

Ocena jakościowa jest subiektywna niemal z definicji. Kilku badaczy zwróciło uwagę na prace [COLW85a, WALL85], jednak w najlepszym razie ich rezultaty są niejednoznaczne; są też kwestionowane [PATT85b] i oczywiście również bronione [COLW85b].

Sukces rozwiązania RISC na rynku pozostaje sprawą przyszłości. W miarę postępowania prac badawczo-rozwojowych i wdrożeń produkcyjnych, dyskusje są kontynuowane.

W ciągu ostatnich lat polemika - RISC versus CISC - w znacznej mierze zanikła. Stało się tak z powodu stopniowego przenikania się technologii. W miarę wzrostu gęstości upakowania mikroukładów i szybkości działania samego sprzętu systemy RISC stały się bardziej złożone. Jednocześnie w ramach dążenia do osiągnięcia maksymalnej wydajności, projektowanie architektur CISC skupiło się na zagadnieniach tradycyjnie związanych z RISC, takich jak wzrost liczby rejestrów ogólnego przeznaczenia i zwiększenie nacisku na projektowanie potoku rozkazów.

12.9

POLECANA LITERATURA

Prace [FELD94], [TAB91] i [DEWA90] zawierają obszernie omówienie systemów RISC. Dobre omówienie koncepcji RISC znajduje się w podręcznikach [WARD90], [PATT94] i [HENN90].

Bardzo szczegółowy opis procesora Motorola 88000 zawarto w [ALSU90]. Rozdział poświęcony temu mikroprocesorowi znajduje się w [TAB91]. [KANE92] zawiera szczegółowe omówienie dostępnej na rynku maszyny MIPS. W [MIRA92] znajduje się dobry przegląd MIPS R4000. W [BASH91] przedyskutowano ewolucję od potoku R3000 do superpotoku R4000.

ALSU90 Alsup M.: „The Motorola's 88000 Family Architecture”. *IEEE Micro*, June 1990.

BASH91 Bashteen A., Lui L., Mullan J.: „A Superpipeline Approach to the MIPS Architecture”. *Proceedings, COMPCON Spring '91*, February 1991.

DEWA90 Dewar R., Smosna M.: *Microprocessors: A Programmers View*. New York, McGraw-Hill, 1990.

FELD94 Feldman J., Retter C.: *Computer Architecture*. New York, McGraw-Hill, 1994.

HENN90 Hennessy J., Patterson D.: *Computer Architecture: A Quantitative Approach*. San Mateo, Morgan Kaufmann, 1990.

KANE92 Kane G., Heinrich J.: *MIPS RISC Architecture*. Englewood Cliffs, Prentice Hall, 1992.

MIRA92 Mirapuri S., Woodacre M., Vasseghi N.: „The MIPS R4000 Processor”. *IEEE Micro*, April 1992.

PATT94 Patterson D., Hennessy J.: *Computer Organization and Design: The Hardware/Software Interface*. San Mateo, Morgan Kaufmann, 1994.

TAB91 Tabak D.: *Advanced Microprocessors*. New York, McGraw-Hill, 1991.

WARD90 Ward S., Halstead R.: *Computation Structures*. Cambridge, MIT Press, 1990.

12.10

PROBLEMY DO ROZWIĄZANIA

12.1. Rozważ przebieg wywołań i powrotów przedstawiony na rys. 4.31. Ile przepełnień i niedomiarów (z których każde powoduje operację zachowywania i odnawiania zawartości rejestru) nastąpi przy rozmiarze okna:

- (a) 5;
- (b) 8;
- (c) 16.

12.2. Rozważ architekturę, w której 100 rejestrów użyto do budowy cyklicznego bufora przedstawionego na rys. 12.2. W każdym z okien 5 rejestrów wykorzystuje się do przechowywania parametrów. Załóż, że liczba zmiennych lokalnych używanych w procedurze ma równomierny rozkład od 5 do 14; oznacza to, że 10% używa 5 zmiennych, 10% używa 6 zmiennych itd. Załóż wreszcie, że wywołania i powroty następują zgodnie z rys. 4.31. Jaka kombinacja rozmiaru okna i liczby okien jest optymalna?

- 12.3. Omawiając rys. 12.2, stwierdziliśmy, że tylko dwie pierwsze części okna są zachowywane i odtwarzane. Dlaczego nie jest konieczne zachowywanie zawartości rejestrów tymczasowych?
- 12.4. Chcemy wyznaczyć czas wykonywania danego programu, stosując różne schematy przetwarzania potokowego rozważane w podrozdz. 12.5.

Oznaczmy:

N - liczba wykonanych rozkazów,
 D - liczba dostępów do pamięci,
 J - liczba rozkazów skoku.

W przypadku prostego schematu sekwencyjnego (rys. 12.6) czas wykonania wynosi $2N + D$ faz. Wyprowadź wzory dla przetwarzania potokowego 2-, 3- i 4-etapowego.

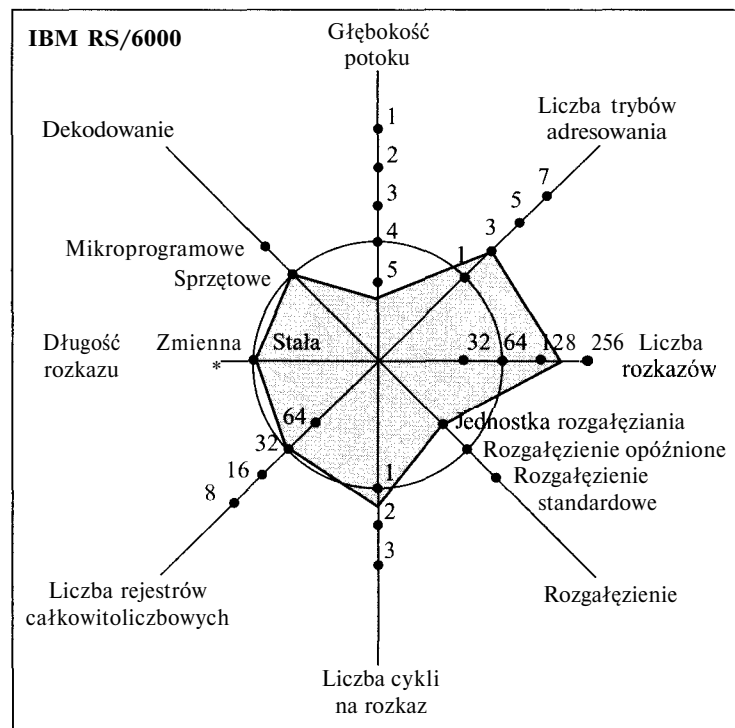
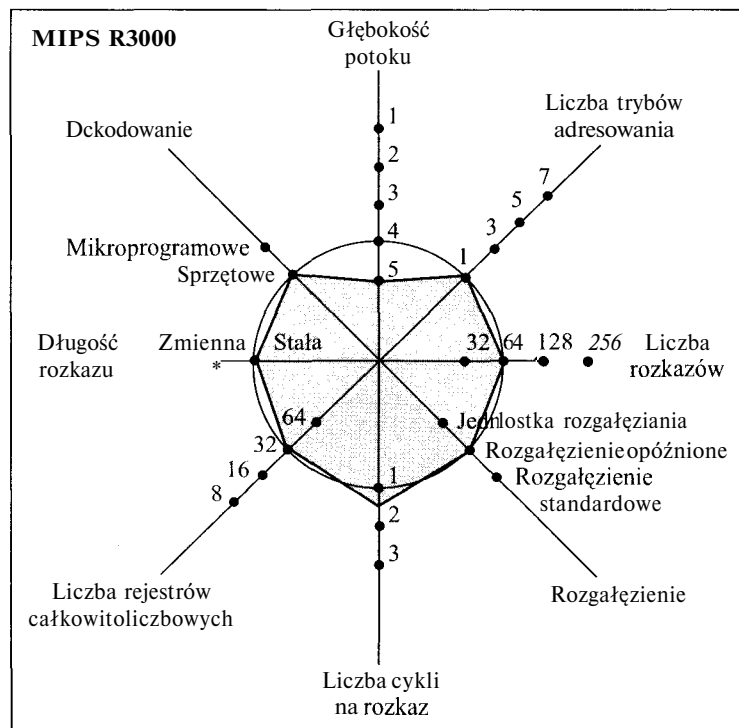
- 12.5. Wykreśl na nowo rys. 12.9 biorąc pod uwagę metodę opóźnionego ładowania rozgałęzień.
- 12.6. Pokaż równoważnik MIPS tych trybów adresowania Pentium (tab. 10.2), które mogą być syntetyzowane w MIPS.
- 12.7. W wielu przypadkach wspólne rozkazy maszynowe, nie wymienione w ramach listy rozkazów MIPS, mogą być syntetyzowane w postaci pojedynczego rozkazu MIPS. Pokaż tę możliwość dla:
- (a) przeniesienie z rejestru do rejestru;
 - (b) inkrementacja, dekrementacja;
 - (c) uzupełnienie;
 - (d) zanegowanie;
 - (e) zerowanie.

- 12.8. Rozważ następujący fragment programu w języku wysokiego poziomu:

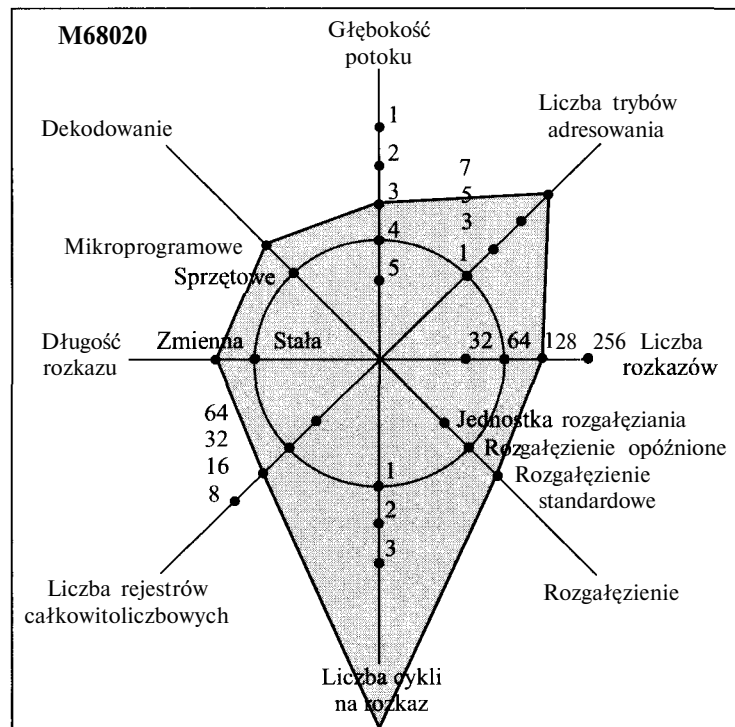
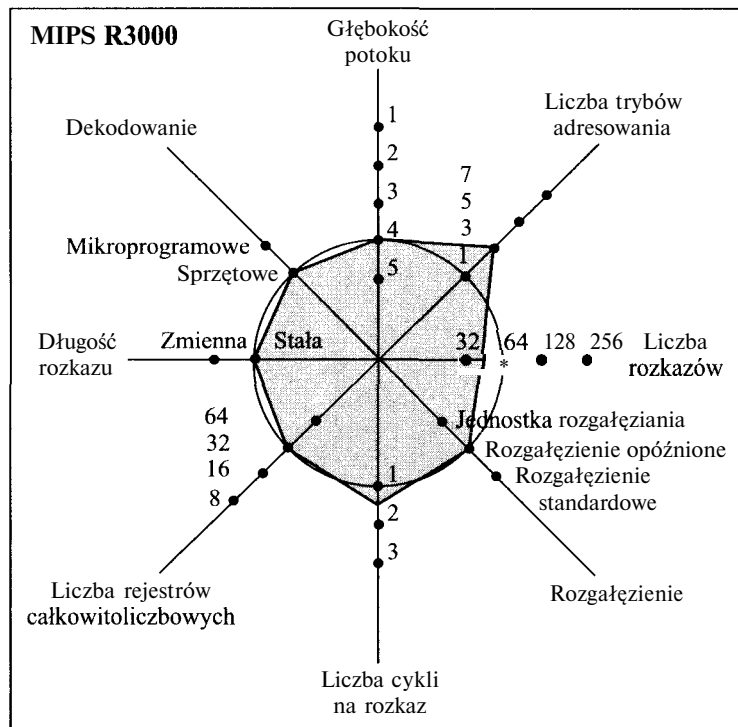
```
for I in 1 ... 100 loop
  S ← S + Q(I).VAL
end loop;
```

Załóż, że Q jest tablicą rekordów 32-bajtowych i że pole VAL znajduje się w pierwszych 4 bajtach każdego rekordu. Stosując kod procesora 80x86, możemy skompilować ten fragment programu następująco:

	MOV	ECX, 1	;użyj rejestru ECX do przechowania I
LP:	IMUL	EAX, ECX, 32	;znajdź przesunięcie w EAX
	MOV	EBX, Q[EAX]	;ładuj pole VAL
	ADD	S, EBX	;dodaj do S
	INC	ECX	;zwiększ stan I
	CMP	ECX, 100	;sprawdź w stosunku do granicy
	JNE	LP	;pętla aż do I = 100.



RYSUNEK 12.17. Grafy Kiviata dla wybranych komputerów



RYSUNEK 12.17 (cd.)

Program ten wykorzystuje rozkaz IMUL, powodujący mnożenie drugiego argumentu przez wartość natychmiastową w trzecim argumencie i umieszczenie wyniku w pierwszym argumencie (patrz problem 10.18). Zwolennik architektury RISC chciałby zademonstrować, że inteligentny kompilator może wyeliminować niepotrzebnie złożone rozkazy, takie jak IMUL. Zademonstruj to, przerabiając powyższy program procesora 80x86 bez użycia rozkazu IMUL.

- 12.9. Lista rozkazów procesora 88000 zawiera zarówno rozkaz Branch to Subroutine (rozgałęziaj do podprogramu), jak i Jump to Subroutine (skocz do podprogramu). Zastanów się, jaka różnica może wystąpić między nimi i dlaczego oba te rozkazy mogą być potrzebne w tak ograniczonej liście rozkazów.
- 12.10. W [ESPO92] autorzy proponują użycie wykresów Kiviata do zobrazowania najistotniejszych parametrów architektury RISC. Wykresy takie zostały użyte w [SIEW82] do przedstawienia parametrów wydajnościowych różnych maszyn. W tym przypadku wykres Kiviata został użyty do oceny stopnia, w jakim procesor wykazuje cechy RISC. Jak można zauważyć na rys. 12.17, przy porównywaniu procesorów rozważa się 8 zmiennych. Zmienne te są wykreślane w regularnych odstępach kątowych. Na okręgu są zaznaczone punkty, które na każdej osi powinny być uważane za „typowe” wartości RISC. Wykres nie tylko umożliwi podsumowanie głównych własności architektury, ale także obrazuje graficznie stopień, w jakim system zbliża się do czystego projektu RISC.

Rysunek zawiera wykresy dla procesorów MIPS R3000 i Motorola 88000, które mają wyraźne cechy architektury RISC. Pokazano też IBM RS/6000, który jest prekursorem architektury PowerPC. Jest to złożony projekt o niektórych wyraźnych cechach RISC, w którym jednak inne cechy odpowiadają bardziej CISC. Pokazano również wykres dla procesora Motorola 68020. Natura CISC tego projektu jest łatwo czytelna.

Uzyskaj konieczne informacje i nakreśl wykresy Kiviata dla następujących procesorów i skomentuj wyniki:

- (a) IBM S/370;
- (b) VAX;
- (c) Pentium;
- (d) PowerPC.

- 12.11. Dodaj do tabeli 12.9 zapisy dotyczące procesorów:

- (a) Pentium;
- (b) PowerPC.

Procesory superskalarne

W procesorze o architekturze superskalarnej zwykłe rozkazy, na przykład dotyczące arytmetyki liczb całkowitych lub zmiennopozycyjnych, ładowania, zapisu i rozgałęzienia warunkowego, mogą być równocześnie inicjowane i niezależnie wykonywane. Rozwiązanie takie stwarza wiele złożonych problemów projektowych związanych z potokowym przetwarzaniem rozkazów.

Projekty superskalarne pojawiły się na scenie, deptając po piętach architektury RISC. Chociaż architektura o uproszczonej liście rozkazów maszyn RISC sprzyja metodom superskalarным, to rozwiązania superskalarne mogą być stosowane zarówno w architekturze RISC, jak i CISC. Jednak praktycznie wszystkie wdrożenia superskalarne zostały oparte na architekturze RISC.

Podczas gdy okres dojrzewania - między początkiem badań nad RISC w postaci maszyn IBM 801 i Berkeley RISC I a ukazaniem się komercyjnych maszyn RISC - trwał siedem lub osiem lat, pierwsze maszyny superskalarne ukazały się rok lub dwa po wymyśleniu terminu „superskalarny”. Rozwiązanie superskalarne w połączeniu z architekturą RISC lub bliską RISC wydaje się być najbardziej ekscytującym obszarem prac badawczych i rozwojowych dotyczących organizacji i architektury komputerów w ciągu najbliższych kilku lat.

Rozpocniemy ten rozdział od przeglądu podejścia superskalarного, przedstawiając je przetwarzaniu superpotokowemu. Następnie przedstawimy główne zagadnienia projektowe związane z rozwiązaniem superskalarным, a na zakończenie zostaną przedstawione dwa najbardziej znaczące komercyjne procesory superskalarne.

13.1

PRZEGLĄD

Termin **superskalarny**, użyty po raz pierwszy w roku 1987 [AGER87], odnosi się do maszyny, która została zaprojektowana pod kątem zwiększenia wydajności wykonywania rozkazów skalarnych. To podejście jest przeciwieństwem po-

dobnego zamierzenia dotyczącego procesorów wektorowych, omówionego w rozdziale 16. W większości zastosowań główna część operacji jest prowadzona na wielkościach skalarnych. Wobec tego rozwiązanie superskalarne reprezentuje następny krok w ewolucji wysoko wydajnych procesorów o ogólnym przeznaczeniu.

Wielu badaczy analizowało procesory zbliżone do superskalarnych, a wyniki ich badań wskazują na możliwość pewnej poprawy wydajności. Stwierdzony stopień poprawy wydajności jest przedstawiony w tabeli 13.1. Różnice rezultatów wynikają zarówno z różnic sprzętowych symulowanych maszyn, jak i z symulowanych zastosowań.

TABELA 13.1. Doniesienia o przyspieszeniu pracy komputerów dzięki rozwiązaniu superskalarnemu

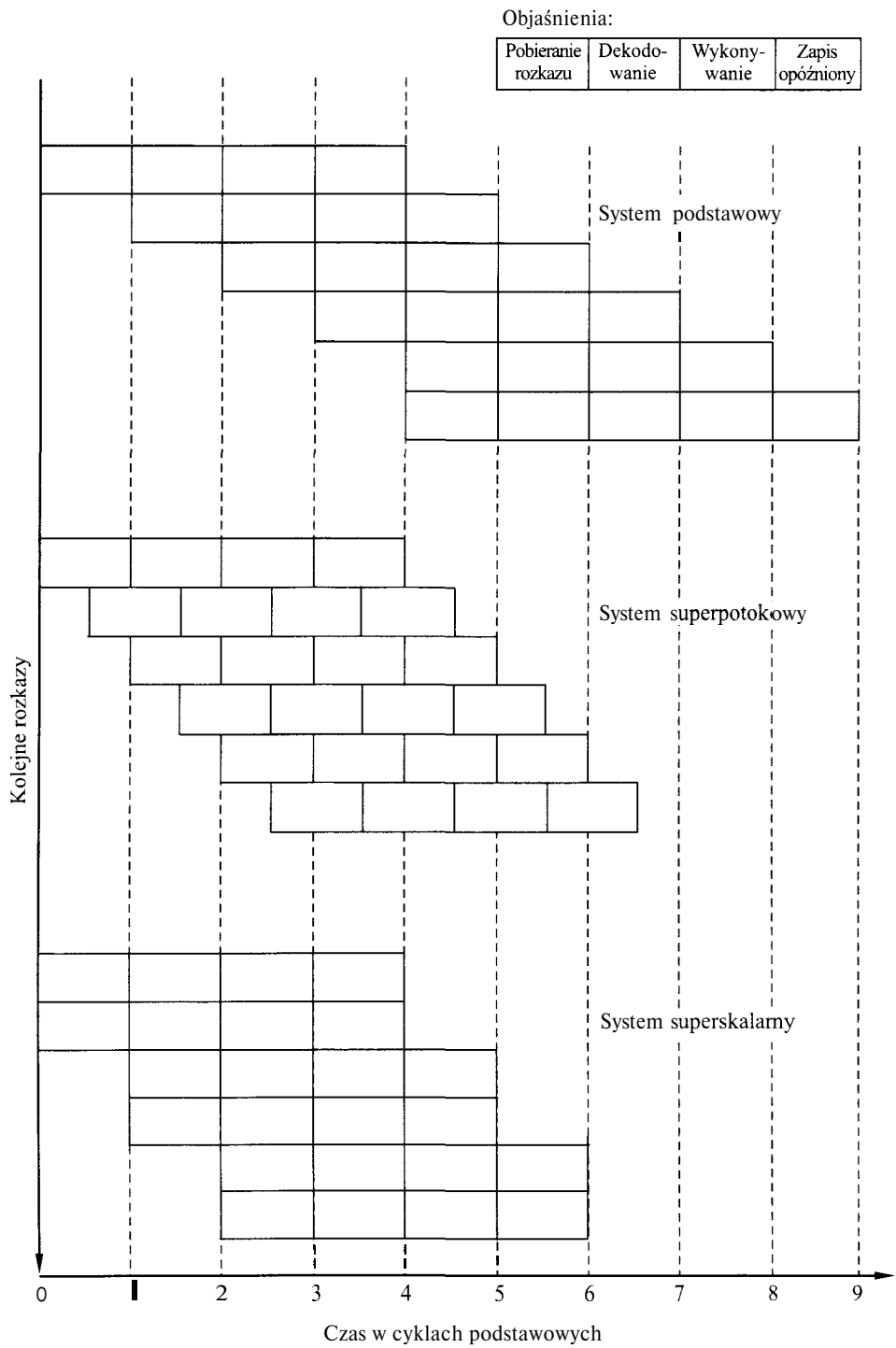
Źródło	Przyspieszenie
[TJAD70]	1,8
[KUCK72]	8
[WEIS84]	1,58
[ACOS86]	2,7
[SOHI87]	1,8
[SMIT89]	2,3
[JOUN89]	2,2
[LEE91]	7

Rozwiązanie superskalarne a przetwarzanie superpotokowe

Alternatywnym sposobem zwiększenia wydajności jest przetwarzanie superpotokowe; termin ten został po raz pierwszy wprowadzony w roku 1988 [JOUN88]. W przetwarzaniu superpotokowym wykorzystuje się fakt, że podczas wielu etapów potoku wykonuje się zadania zajmujące mniej niż połowę cyklu zegara. Wobec tego podwojenie szybkości zegara wewnętrznego umożliwia wykonywanie dwóch zadań w tym samym cyklu zegara zewnętrznego. Rozwiązanie takie widzieliśmy na przykładzie MIPS R4000.

Na rysunku 13.1 są porównane oba te rozwiązania. W górnej części wykresu znajduje się zwykły potok, stanowiący podstawę do porównań. W podstawowym potoku występuje jeden rozkaz w cyklu zegara i może być wykonywany jeden etap potoku w jednym cyklu zegara. Potok ten ma cztery etapy: pobranie rozkazu, dekodowanie operacji, wykonanie operacji i zapisanie wyniku. Załóżmy, że chociaż kilka rozkazów jest wykonywanych równocześnie, w określonej chwili tylko jeden rozkaz znajduje się na etapie wykonywania.

Następna część wykresu ilustruje rozwiązanie superpotokowe, umożliwiające realizowanie dwóch etapów potoku w każdym cyklu zegara. Alternatywnym sposobem widzenia tego rozwiązania jest rozdzielenie funkcji wykonywanych na każdym etapie na dwie nienakładające się części, z których każda może być wykonywana w połowie cyklu zegara. Rozwiązanie superpotokowe tego rodzaju



RYSUNEK 13.1. Porównanie rozwiązań superskalarnego i superpotokowego

jest określone jako rozwiązanie stopnia 2. Wreszcie najniższa część wykresu ilustruje rozwiązanie superskalarne, umożliwiające równoległe wykonywanie dwóch przypadków każdego etapu. Możliwe są oczywiście superpotoki i rozwiązania superskalarne wyższego stopnia.

Zarówno w rozwiązaniu superpotokowym, jak i superskalarным przedstawionych na rys. 13.1 występuje taka sama liczba rozkazów wykonywanych jednocześnie w stanie ustalonym. Procesor superpotokowy przegrywa z procesorem superskalarным na początku programu i przy każdym punkcie docelowym rozgałęzienia.

Ograniczenia

Rozwiązanie superskalarne polega na możliwości równoległego wykonywania wielu rozkazów. Termin *paralelizm na poziomie rozkazu* odnosi się do stopnia, w którym rozkazy programu mogą przeciętnie być wykonywane równoległe. W celu maksymalizacji paralelizmu na poziomie rozkazu można stosować kombinację optymalizacji za pomocą kompilatora oraz metod sprzętowych. Przed analizowaniem metod projektowania maszyn superskalarnych pod kątem zwiększania paralelizmu na poziomie rozkazu, musimy przyjrzeć się podstawowym ograniczeniom paralelizmu, którym system musi podołać. W pracy [JOHN91] wymieniono pięć ograniczeń:

- a prawdziwa zależność danych,
- a zależność proceduralna,
- a konflikt dotyczący zasobów,
- a zależność wyjściowa,
- antyzależność.

W pozostałej części podrozdziału przeanalizujemy pierwsze trzy z tych ograniczeń. Rozważania dotyczące pozostałych muszą poczekać do następnego podrozdziału.

Prawdziwa zależność danych

Rozważmy następującą sekwencję rozkazów:

```
add    r1, r2    ;ładuj rejestr r1 zawartością rejestru r2 plus zawartość r1
move   r3, r1    ;ładuj rejestr r3 zawartością r1
```

Drugi rozkaz może być pobrany i zdekodowany, jednak nie może być wykonany, zanim nie zostanie wykonany pierwszy rozkaz. Powodem jest to, że drugi rozkaz potrzebuje danych tworzonych za pomocą pierwszego rozkazu. Sytuacja taka jest określana jako **prawdziwa zależność danych** (nazywana również **zależnością przepływu** lub **zależnością zapis-odczyt**).

Na rysunku 13.2 jest zilustrowana ta zależność w maszynie superskalarnej stopnia 2. Gdy nie występuje zależność, oba rozkazy mogą być pobierane i wykonywane równolegle. Jeśli natomiast istnieje zależność danych między pierwszym a drugim rozkazem, to drugi rozkaz ulega opóźnieniu o tyle cykli zegara, ile jest wymaganych do usunięcia zależności. Ogólnie rzecz biorąc, dowolny rozkaz musi być opóźniony, aż zostaną utworzone jego wszystkie wartości wejściowe.

Prawdziwa zależność danych ogranicza wydajność każdego rodzaju potoku. W prostym potoku skalarnym powyższy ciąg rozkazów nie wywoła opóźnienia. Rozważmy jednak następującą sekwencję:

```
load  r1, eff ;ładuj rejestr r1 zawartością efektywnego adresu pamięci eff
move  r3, r1  ;ładuj rejestr r3 zawartością r1
```

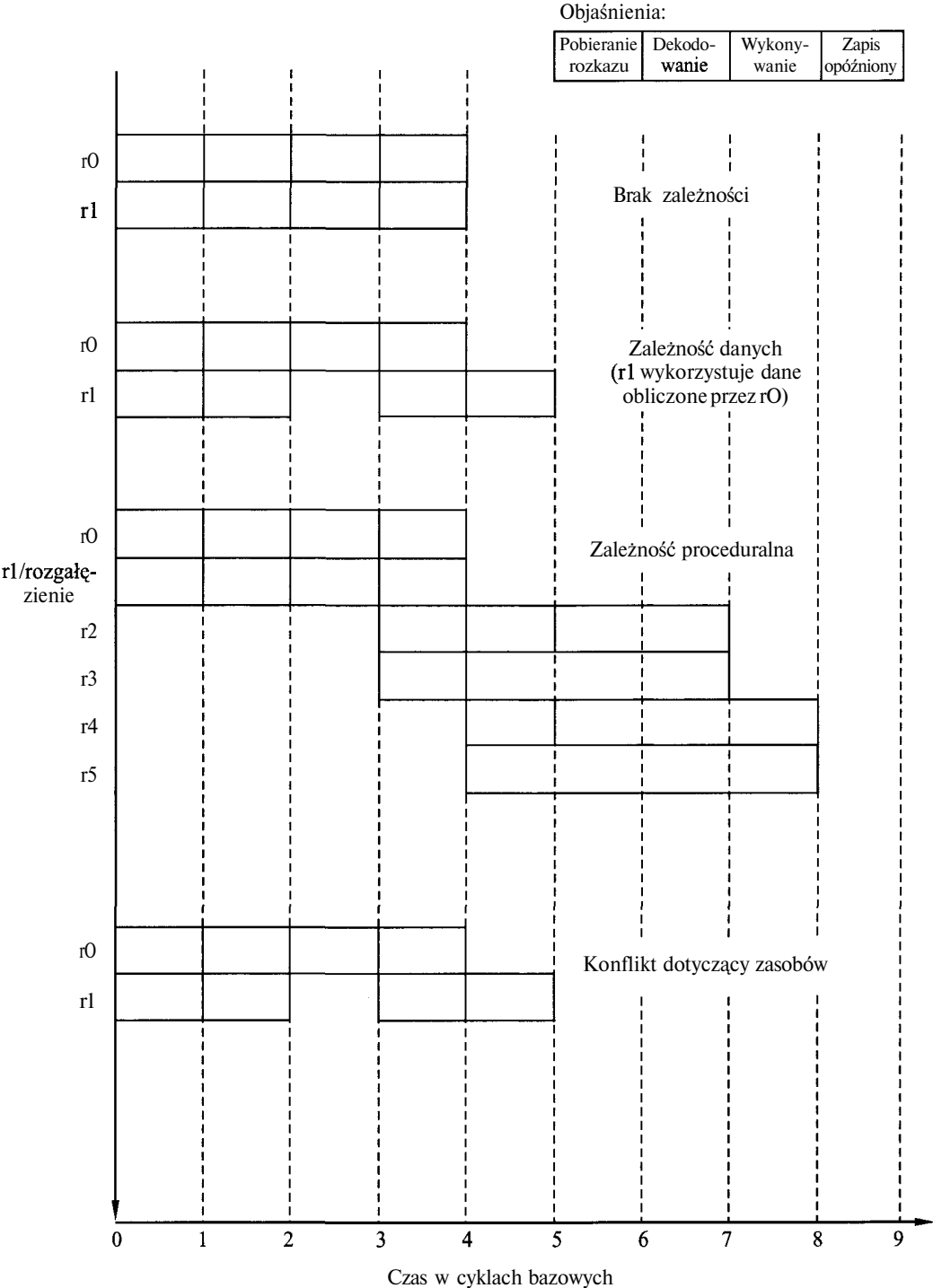
Typowy procesor RISC potrzebuje dwóch lub więcej cykli w celu przeprowadzenia ładowania z pamięci ze względu na opóźnienie dostępu do pamięci znajdującej się poza mikroukładem. Jednym ze sposobów skompensowania tego opóźnienia jest zmiana kolejności rozkazów przez kompilator, dzięki czemu jeden lub więcej kolejnych rozkazów nie zależących od ładowania z pamięci może rozpocząć przepływ przez potok. Schemat ten jest mniej efektywny w przypadku potoku superskalarnego. Niezależne rozkazy wykonywane podczas ładowania najprawdopodobniej zostaną wykonane podczas pierwszego cyklu ładowania, pozostawiając procesor bez zajęcia do zakończenia ładowania.

Zależności proceduralne

Jak stwierdziliśmy w rozdziale 11, obecność rozgałęzień w sekwencji rozkazów komplikuje działanie potoku. Rozkazy następujące po rozgałęzieniu (dokonanym lub nie dokonanym) wykazują zależność proceduralną od rozgałęzienia i nie mogą być wykonywane przed zakończeniem rozgałęzienia. Na rysunku 13.2 jest pokazany wpływ rozgałęzienia na potok superskalarny stopnia 2.

Jak widzieliśmy, ten rodzaj zależności proceduralnej oddziałuje także na potok skalarny. Jednak podobnie jak poprzednio, konsekwencje w odniesieniu do potoku superskalarnego są poważniejsze, ze względu na większe możliwości tracone przy każdym opóźnieniu.

Jeśli są używane rozkazy o zmiennej długości, to powstaje inny rodzaj zależności proceduralnej. Ponieważ długość określonego rozkazu nie jest znana, musi on być przynajmniej częściowo zdekodowany, zanim będzie mógł być pobrany następny rozkaz. Nie jest potrzebne jednoczesne pobieranie wymagane przez potok superskalarny. Jest to jeden z powodów, dla których metody superskalarne są łatwiejsze do zastosowania w architekturach RISC lub zbliżonych do RISC, które mają ustalone długości rozkazów.



RYSUNEK 13.2. Wpływ zależności

Konflikt dotyczący zasobów

Konflikt dotyczący zasobów polega na jednoczesnym rywalizowaniu dwóch lub wielu rozkazów o te same zasoby. Przykładami takich zasobów są pamięci, pamięci podręczne, magistrale, porty tablic rejestrów i jednostki funkcjonalne (np. sumator ALU).

W odniesieniu do potoku konflikt dotyczący zasobów wygląda podobnie jak w przypadku zależności danych (rys. 13.2). Istnieją jednak pewne różnice. Po pierwsze, problemy dotyczące zasobów mogą być przewyżczone przez powielenie zasobów, podczas gdy prawdziwa zależność danych nie może być wyeliminowana. Po drugie, gdy czas trwania operacji jest długi, konflikt dotyczący zasobów może być zminimalizowany przez zastosowanie przetwarzania potokowego w odpowiedniej jednostce funkcjonalnej. Na przykład we wszystkich jednostkach funkcjonalnych w procesorze Motorola 88000 zastosowano przetwarzanie potokowe.

13.2

ZAGADNIENIA DOTYCZĄCE PROJEKTOWANIA

Paralelizm na poziomie rozkazu i paralelizm na poziomie maszyny

W pracy [JOUP89a] istnieje rozróżnienie między dwiema powiązаныmi koncepcjami paralelizmu na poziomie rozkazów i paralelizmu na poziomie maszyny. **Paralelizm na poziomie rozkazów** występuje, gdy rozkazy w sekwencji są niezależne i wobec tego mogą być wykonywane równolegle przez nakładanie (ang. *overlapping*).

Jako przykład koncepcji paralelizmu na poziomie rozkazów rozważmy dwa następujące fragmenty kodów [JOUP89b]:

Load R1 ← R2(23)	Add R3 ← R3, „1”
Add R3 ← R3, „1”	Add R4 ← R3, R2
Add R4 ← R4, R3	Store [R4] ← R0
(a) równoległość = 3	(b) równoległość = 1

Trzy rozkazy we fragmencie (a) są niezależne i teoretycznie wszystkie trzy mogłyby być wykonywane równolegle. W przeciwieństwie do tego trzy rozkazy we fragmencie (b) nie mogą być wykonywane równolegle, ponieważ drugi rozkaz używa wyników pierwszego, a trzeci używa wyników drugiego.

Paralelizm na poziomie rozkazów jest określony przez częstość prawdziwych zależności danych oraz zależności proceduralnych w kodzie. Z kolei te czynniki są zależne od architektury listy rozkazów i od zastosowania. Paralelizm na poziomie rozkazów jest również zdeterminowany przez efekt określony

w pracy [JOUP89a] jako opóźnienie operacji (ang. *operation latency*): czas wymagany do tego, aby wynik rozkazu stał się dostępny jako argument następnego rozkazu. Parametr ten określa, jakie opóźnienie spowoduje zależność danych lub zależność proceduralna.

Paralelizm na poziomie maszyny jest miarą zdolności procesora do wykorzystania paralelizmu na poziomie rozkazów. Paralelizm maszynowy jest zterminowany liczbą rozkazów, które mogą być pobrane i wykonane w tym samym czasie (liczba równoległych potoków), oraz szybkością i złożonością mechanizmów, których procesor używa do znajdowania niezależnych rozkazów.

Oba rodzaje paralelizmu są ważnymi czynnikami zwiększania wydajności. Program może nie wykazywać dostatecznego paralelizmu na poziomie rozkazów, aby w pełni wykorzystana była zaleta paralelizmu maszynowego. Użycie architektury, w której lista rozkazów ma ustaloną długość, takiej jak RISC, umożliwi zwiększenie paralelizmu na poziomie rozkazów. Z drugiej strony ograniczony paralelizm maszynowy ograniczy wydajność niezależnie od natury programu.

Strategia wydawania rozkazów

Jak już stwierdziliśmy, paralelizm maszynowy nie jest po prostu sprawą dysponowania wieloma instancjami każdego etapu potoku. Procesor musi również móc identyfikować paralelizm na poziomie rozkazów i koordynować równoległe pobieranie, dekodowanie i wykonywanie rozkazów. W pracy [JOHN91] użyto terminu **wydawanie rozkazów** odnoszącego się do inicjowania wykonywania rozkazów w jednostkach funkcjonalnych procesora oraz terminu **strategia wydawania rozkazów** odnoszącego się do protokołu używanego przy wydawaniu rozkazów.

W istocie, procesor próbuje wyprzedzić bieżący punkt wykonywania w celu zlokalizowania rozkazów, które mogą być wprowadzone do potoku i wykonane. Pod tym względem istotne są:

- a kolejność pobierania rozkazów;
- ☐ kolejność wykonywania rozkazów;
- a kolejność, w której rozkazy zmieniają lokacje w rejestrach i w pamięci.

Im bardziej wyrafinowany jest procesor, tym mniej jest on uzależniony od ścisłych powiązań między tymi kolejnościami. Aby osiągnąć maksymalne wykorzystanie różnych elementów potoku, procesor musi zmieniać jedną lub więcej z wymienionych kolejności w stosunku do oryginalnej kolejności występującej przy wykonywaniu ściśle sekwencyjnym. Jedynym ograniczeniem dla procesora jest to, że wynik musi być poprawny. Wobec tego procesor musi się dostosować do różnych zależności i konfliktów, które omówiliśmy powyżej.

Ogólnie rzecz biorąc **superskalarne** strategie wydawania rozkazów możemy podzielić na następujące kategorie:

- kolejne wydawanie połączone z kolejnym kończeniem;
- a kolejne wydawanie połączone z kończeniem w zmienionej kolejności;
- a wydawanie w zmienionej kolejności połączone z kończeniem w zmienionej kolejności.

Kolejne wydawanie połączone z kolejnym kończeniem

Najprostsza strategia wydawania rozkazów jest wydawanie ich w dokładnej kolejności zgodnej z wykonywaniem sekwencyjnym (kolejne wydawanie) i zapisywanie wyników w tej samej kolejności (kolejne kończenie). Nawet skalarne potoki nie dopuszczają tak uproszczonej strategii. Jest jednak użyteczne rozważenie tej strategii jako podstawy do porównywania bardziej wyrafinowanych rozwiązań.

Na rysunku 13.3a widać przykład takiej strategii. Załóżmy, że potok superskalarny może jednocześnie pobierać i zdekodować dwa rozkazy, że ma trzy odrębne jednostki funkcjonalne (np. jednostkę arytmetyki liczb stałych i arytmetyki zmiennopozycyjnej) i że ma dwie instancje etapu zapisywania.

(a)

Dekodowanie	Wykonywanie	• Zapis opóźniony	Cykl
I1 I2			1
I3 I4	I1 I2		2
I3 I4	I1		3
I4		I1 I2	4
I5 I6			5
I5 I6	I5 I6	I3 I4	6
			7
		I5 I6	8

(b)

Dekodowanie	Wykonywanie	Zapis opóźniony	Cykl
I1 I2			1
I3 I4	I1 I2		2
I4	I1	I2	3
I5 I6		I1 I3	4
I5 I6	I5 I6	I4	5
		I5	6
		I6	7

(c)

Dekodowanie	Okno	Wykonywanie	Zapis opóźniony	Cykl
I1 I2				1
I3 I4	I1, I2	I1 I2		2
I5 I6	I3, I4	I1	I2	3
	I4, I5, I6		I1 I3	4
	I5	I5 I6	I4 I6	5
			I5	6

RYSUNEK 13.3. Superskalarne metody wydawania i kończenia rozkazów: (a) kolejne wydawanie i kolejne kończenie; (b) kolejne wydawanie, zmieniona kolejność kończenia; (c) zmieniona kolejność wydawania, zmieniona kolejność kończenia

Założmy też następujące ograniczenia dotyczące 6-rozkazowego fragmentu programu:

- I1 wymaga do wykonania 2 cykli.
- a I3 i I4 rywalizują o tę samą jednostkę funkcjonalną.
- Q I5 zależy od wartości tworzonej przez I4.
- I5 i I6 rywalizują o jednostkę funkcjonalną.

Rozkazy są pobierane jednocześnie po dwa i przechodzą do jednostki dekodowania. Ponieważ rozkazy są pobierane parami, następne dwa rozkazy muszą czekać, aż para etapów dekodowania w potoku będzie pusta. Aby zapewnić jednocześnie kończenie, wydawanie rozkazów ulega zatrzymaniu, gdy występuje konflikt dotyczący jednostki funkcjonalnej lub gdy jednostka funkcjonalna wymaga więcej niż jednego cyklu do generowania wyniku.

W tym przykładzie czas między dekodowaniem pierwszego rozkazu a zapisaniem ostatniego wyniku wynosi 8 cykli.

Kolejne wydawanie połączzone ze zmienioną kolejnością kończenia

W skalarnych procesorach RISC zmieniona kolejność kończenia jest stosowana w celu poprawienia wydajności rozkazów wymagających wielu cykli. W ten właśnie sposób przeprowadzane są operacje zmiennopozycyjne w procesorze 88000 Motoroli.

Na rysunku 13.3b jest pokazane zastosowanie tej strategii w procesorze superskalarnym. Rozkaz I2 może być zakończony przed I1. Umożliwia to wcześniejsze zakończenie rozkazu I3, co netto daje korzyść w postaci zaoszczędzenia jednego cyklu.

Przy zmienionej kolejności kończenia w określonym czasie na etapie wykonywania może się znajdować dowolna liczba rozkazów, aż do osiągnięcia maksymalnego stopnia paralelizmu maszynowego we wszystkich jednostkach funkcjonalnych. Wydawanie rozkazów jest zatrzymywane w wyniku konfliktu dotyczącego zasobów, zależności danych lub zależności proceduralnej.

Poza powyższymi ograniczeniami powstaje nowy rodzaj zależności, który określaliśmy wcześniej jako **zależność wyjściowa** (nazywana także **zależnością odczyt-zapis**). Następujący fragment kodu ilustruje tę zależność (op oznacza dowolną operację):

R3 := R3 op R5	(I1)
R4 := R3 + 1	(I2)
R3 := R5 + 1	(I3)
R7 := R3 op R4	(I4)

Rozkaz I2 nie może być wykonany przed rozkazem I1, ponieważ potrzebuje wyniku w rejestrze R3 tworzonego przez rozkaz I1; jest to przykład prawdziwej zależności danych, opisaney w podrozdz. 13.1. Podobnie rozkaz I4 musi czekać

na I3, ponieważ używa wyniku tworzonego przez I3. Co można powiedzieć o zależności między rozkazami I1 a I3? Nie występuje tu taka zależność danych, jaką zdefiniowaliśmy. Jeśli jednak wykonywanie rozkazu I3 zakończy się przed I1, to do wykonania I4 zostanie pobrana niewłaściwa zawartość rejestru R3. Wobec tego wykonywanie rozkazu I3 musi zakończyć się po rozkazie I1, żeby otrzymać prawidłowe wartości wyjściowe. Aby to zapewnić, wydawanie trzeciego rozkazu musi być wstrzymane, jeśli jego wynik może być później skasowany przez starszy rozkaz wymagający dłuższego czasu do zakończenia.

Zmieniona kolejność kończenia wymaga bardziej złożonych układów logicznych wydawania rozkazów niż kończenie kolejne. Ponadto trudniejsze jest postępowanie z przerwaniami i wyjątkami. Gdy następuje przerwanie, wykonywanie rozkazu w bieżącym punkcie jest zawieszane, a w późniejszym czasie wznowione. Procesor musi zapewnić, żeby przy wznowianiu było wzięte pod uwagę to, że wykonywanie rozkazów znajdujących się przed rozkazem powodującym przerwanie mogło już być zakończone.

Zmieniona kolejność wydawania w połączeniu ze zmienioną kolejnością kończenia

W przypadku kolejnego wydawania rozkazów procesor jedynie dekoduje rozkazy aż do wystąpienia zależności lub konfliktu. Żadne nowe rozkazy nie są dekodowane, dopóki konflikt nie jest rozstrzygnięty. W rezultacie procesor nie może wyprzedzić punktu konfliktu w celu znalezienia następnych rozkazów, które mogą być niezależne od rozkazów znajdujących się już w potoku i które mogą być z pożytkiem wprowadzone do potoku.

Aby umożliwić wydawanie rozkazów w zmienionej kolejności, konieczne jest rozłączenie etapów dekodowania i wykonywania w potoku. Czyni się to za pomocą bufora nazywanego **oknem rozkazu**. Przy takiej organizacji, rozkaz po zdekodowaniu przez procesor jest umieszczany w oknie rozkazu. Dopóki bufor ten nie jest zapełniony, procesor może kontynuować pobieranie i dekodowanie nowych rozkazów. Gdy jednostka funkcjonalna na etapie wykonywania staje się osiągalna, rozkaz może być wydany z okna rozkazu do etapu wykonywania. Może być wydany dowolny rozkaz pod warunkiem, że (a) potrzebuje on określonej jednostki funkcjonalnej, która jest osiągalna, oraz (b) żaden konflikt lub zależność nie blokuje tego rozkazu.

W wyniku zastosowania takiej organizacji procesor może „spoglądać w przyszłość”, co pozwala mu na zidentyfikowanie niezależnych rozkazów, które mogą być wprowadzone do etapu wykonywania. Rozkazy są wydawane przez okno rozkazu, przy czym nie ma większego znaczenia oryginalny porządek programu. Podobnie jak poprzednio, jedynym ograniczeniem jest prawidłowe wykonanie programu.

Strategia ta jest zilustrowana na rys. 13.3c. W każdym cyklu do etapu dekodowania są pobierane dwa rozkazy. W związku z ograniczonym rozmiarem bu-

fora w każdym cyklu dwa rozkazy z etapu dekodowania są kierowane do okna rozkazu. W omawianym przykładzie jest możliwe wydanie rozkazu I6 przed rozkazem I5 (pamiętamy, że I5 zależy od I4, lecz I6 nie). Wobec tego zarówno na etapie wykonywania, jak i na etapie zapisu został zaoszczędzony jeden cykl, a całkowita oszczędność w porównaniu z rys. 13.3b wynosi również 1 cykl.

Okno rozkazu zostało pokazane na rys. 13.3c w celu zilustrowania jego roli. Okno to nie jest jednak dodatkowym etapem potoku. Pobyt rozkazu w oknie świadczy po prostu o tym, że procesor ma na temat tego rozkazu informację wystarczającą do zdecydowania, kiedy może on być wydany.

Strategia zmienionej kolejności wydawania i zmienionej kolejności kończenia podlega tym samym ograniczeniom, które opisaliśmy wyżej. Rozkaz nie może być wydany, jeśli narusza to zależność lub powoduje konflikt. Różnica polega na tym, że więcej rozkazów jest dostępnych do wydawania, co zmniejsza prawdopodobieństwo zatrzymania potoku. Powstaje ponadto nowego rodzaju zależność, którą określiliśmy wcześniej jako **antyależność** (nazywana także **zależnością zapis-zapis**). Zależność tę ilustruje wcześniej rozważany fragment programu:

R3 := R3 op R5 (I1)

R4 := R3 + 1 (I2)

R3 := R5 + 1 (I3)

R7 := R3 op R4 (I4)

Wykonywanie rozkazu I3 nie może być zakończone, zanim nie zostanie rozpoczęte wykonywanie rozkazu I2 i nie zostaną pobrane jego argumenty. Dzieje się tak, ponieważ rozkaz I3 aktualizuje rejestr R3, który jest źródłem argumentu dla rozkazu I2. Stosowany jest termin **antyależność**, ponieważ ograniczenie jest podobne do prawdziwej zależności danych, jednak jest odwrócone: to nie pierwszy rozkaz tworzy wartość używaną przez drugi rozkaz, lecz drugi rozkaz niszczy wartość używaną przez pierwszy rozkaz.

Przemianowanie rejestrów

Stwierdziliśmy, że jeśli dozwolona jest zmiana kolejności wydawania rozkazów i (lub) zmiana kolejności ich kończenia, to stwarza to możliwość wystąpienia zależności wyjściowej i antyależności. Te rodzaje zależności różnią się od prawdziwej zależności danych i od konfliktów dotyczących zasobów, które odzwierciedlają przepływ danych w programie i sekwencję wykonywania rozkazów. Jednak zależności wyjściowe i antyależności wynikają z tego, że wartości w rejestrach nie odzwierciedlają już sekwencji wartości dyktowanej przez przebieg programu.

Gdy rozkazy są wydawane i kończone w pierwotnej sekwencji, możliwe jest określenie zawartości każdego rejestru w każdym punkcie wykonywania programu. Gdy natomiast są stosowane metody zmienionej kolejności, na podstawie

rozważania sekwencji rozkazów dyktowanej przez program nie można już w pełni określić zawartości rejestrów w każdej chwili. W wyniku tego poszczególne wartości rywalizują o użycie rejestrów, a procesor musi rozwiązywać te konflikty przez okolicznościowe zatrzymywanie etapu potoku.

Zarówno antyzależności, jak i zależności wyjściowe są przykładami konfliktów dotyczących przechowywania. Wiele rozkazów rywalizuje o użycie tych samych lokacji rejestrowych, generując ograniczenia potoku i zmniejszając przez to wydajność. Problem staje się jeszcze bardziej dotkliwy, gdy stosowane są metody optymalizacji rejestrów (omówione w rozdz. 12), ponieważ metody kompilatorowe zmierzają do maksymalizacji wykorzystania rejestrów, co zwiększa liczbę konfliktów dotyczących przechowywania.

Jedną z metod eliminowania tego rodzaju konfliktów jest oparta na tradycyjnym rozwiązaniu konfliktów dotyczących zasobów: na powielaniu zasobów. W tym kontekście metoda ta jest określana jako **przemianowanie rejestrów**. W zasadzie rejestry są przypisywane dynamicznie przez procesor i są związane z wartościami potrzebnymi do wykonywania rozkazów w różnych punktach czasowych. Gdy tworzona jest nowa wartość rejestru (np. gdy wykonywany jest rozkaz, którego wynik ma być docelowo umieszczony w rejestrze), wartości tej jest przypisywany nowy rejestr. Następne rozkazy, które sięgają do tej wartości jako do argumentu źródłowego w tym rejestrze, muszą przejść przez proces przemianowania: odniesienia do rejestrów w tych rozkazach muszą być zrewidowane, aby dotyczyły rejestrów zawierających potrzebne wartości. Wobec tego, takie same oryginalne odniesienia do rejestru w kilku różnych rozkazach mogą dotyczyć różnych rejestrów rzeczywistych, jeśli chodzi o różne wartości.

Rozważmy, jak można użyć przemianowania rejestrów w stosunku do analizowanego przez nas fragmentu programu:

$$R3_b := R3_a \text{ op } R5_a \quad (I1)$$

$$R4_b := R3_b + 1 \quad (I2)$$

$$R3_c := R5_a + 1 \quad (I3)$$

$$R7_b := R3_c \text{ op } R4_b \quad (I4)$$

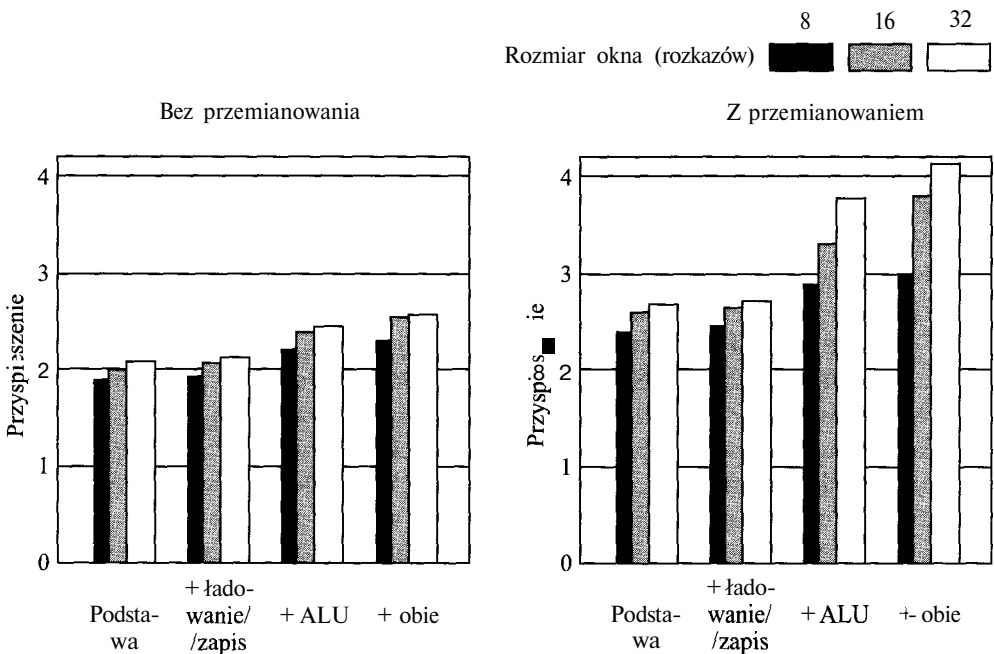
Odniesienie do rejestru bez indeksu oznacza odniesienie logiczne pochodzące z rozkazu. Odniesienie do rejestru z indeksem określa rejestr rzeczywisty przypisany do przechowywania nowej wartości. Gdy dokonywany jest nowy przydział w stosunku do określonego rejestru logicznego, odniesienie do tego samego rejestru logicznego pochodzące z następnego rozkazu przypisuje się najpóźniej przydzielonemu rejestrowi rzeczywistemu (najpóźniej w sensie sekwencji rozkazów programu).

W omawianym przykładzie utworzenie rejestru $R3_c$ w ramach rozkazu I3 zapobiega antyzależności w drugim rozkazie oraz zależności wyjściowej w pierwszym. Nie koliduje ona z prawidłową wartością wymaganą przez I4. W rezultacie rozkaz I3 może być wydany natychmiast; bez przemianowania I3 nie mógłby być wydany przed zakończeniem pierwszego rozkazu i wydaniem drugiego.

Paralelizm maszynowy

Przedyskutowaliśmy dotychczas trzy metody sprzętowe, które mogą być użyte w procesorze superskalarnym w celu zwiększenia wydajności: dublowanie zasobów, zmiana kolejności wydawania rozkazów i przemianowanie. Badanie zależności między tymi metodami przedstawiono w [SMIT89]. W badaniach tych posłużono się symulacją maszyny o własnościach MIPS R2000, uzupełnionej o różne własności superskalarne. Symulowano wiele różnych sekwencji programowych.

Wyniki są pokazane na rys. 13.4. Na każdym wykresie oś pionowa odpowiada przeciętnemu przyspieszeniu maszyny superskalarniej w stosunku do skalarnej. Na osi poziomej wymieniono cztery alternatywne organizacje procesora. W maszynie podstawowej nie zdublowano żadnej jednostki funkcjonalnej, możliwa jest jednak zmiana porządku wydawania rozkazów. Druga konfiguracja zawiera zdublowaną jednostkę ładowania/zapisu, która realizuje dostęp do pamięci podręcznej danych. W trzeciej konfiguracji zdublowano ALU, a w czwartej zarówno jednostkę ładowania/zapisu, jak i ALU. Na każdym wykresie pokazano wyniki odnoszące się do okna mieszczącego 8, 16 i 32 rozkazy, z czego wynika wyprzedzenie możliwe do uzyskania przez procesor. Różnica między obydwoma wykresami polega na tym, że w drugim dozwolone jest przemianowanie re-



RYSUNEK 13.4. Przyspieszenie wynikające z różnic w organizacji komputerów, bez zależności proceduralnych

jestrów. Oznacza to, że pierwszy wykres dotyczy maszyny ograniczanej przez wszelkie zależności, podczas gdy drugi odpowiada maszynie ograniczanej tylko przez prawdziwe zależności.

Oba wykresy potraktowane łącznie prowadzą do pewnych ważnych konkluzji. Pierwszą z nich jest to, że prawdopodobnie nie warto dodawać jednostki funkcjonalnej bez przemianowania rejestrów. Istnieje wprawdzie pewna niewielka poprawa wydajności, jednak jest ona obciążona wzrostem złożoności sprzętu. W połączeniu z przemianowaniem rejestrów, które eliminuje antyzależności i zależności wyjściowe, dodawanie jednostek funkcjonalnych pozwala na osiąganie istotnej poprawy. Zauważmy jednak, że istnieje znaczna różnica w stopniu poprawy między oknem rozkazu równym 8 a większymi oknami. Wskazuje to, że jeśli okno rozkazu jest zbyt małe, zależności danych uniemożliwiają efektywne wykorzystanie dodatkowych jednostek funkcjonalnych; procesor musi być w stanie osiągać wyprzedzenie na tyle duże, żeby mógł znaleźć niezależne rozkazy w celu pełniejszego wykorzystania sprzętu.

Przewidywanie rozgałęzień

Każda maszyna z przetwarzaniem potokowym o dużej wydajności musi się uporać z problemem rozgałęzień. Na przykład w procesorze Intel 80486 zareagowano na ten problem przez pobieranie zarówno następującego po rozgałęzieniu rozkazu sekwencyjnego, jak i spekulatywnego docelowego rozkazu związanego z rozgałęzieniem. Ponieważ jednak między wstępnym pobieraniem a wykonywaniem istnieją dwa etapy potoku, strategia ta wprowadza 2-cykłowe opóźnienie w przypadku, gdy rozgałęzienie następuje.

Wraz z powstaniem maszyn RISC wypróbowano strategię opóźnionego rozgałęziania. Pozwala ona procesorowi na obliczanie wyniku rozkazu warunkowego rozgałęzienia jeszcze przed wstępnym pobraniem niepotrzebnego rozkazu. W tej metodzie procesor zawsze wykonuje pojedynczy rozkaz, który następuje bezpośrednio po rozgałęzieniu. Umożliwia to ciągłe wypełnianie potoku, podczas gdy procesor pobiera nowy strumień rozkazów.

Wraz z rozwojem maszyn superskalarnych strategia opóźnionego rozgałęziania straciła część uroku. Stało się tak dlatego, że wiele rozkazów wymaga wykonywania w przedziale czasowym związanym z opóźnieniem, co stwarza kilka problemów wynikających z zależności rozkazów. Wobec tego maszyny superskalarne wróciły do poprzedzających RISC metod przewidywania rozgałęzień. Niektóre z nich, jak PowerPC 601, stosują prostą metodę statycznego przewidywania rozgałęzień. Bardziej złożone procesory, takie jak PowerPC 620 i Pentium, stosują dynamiczne przewidywanie rozgałęzień oparte na analizie historii rozgałęzień.

13.3

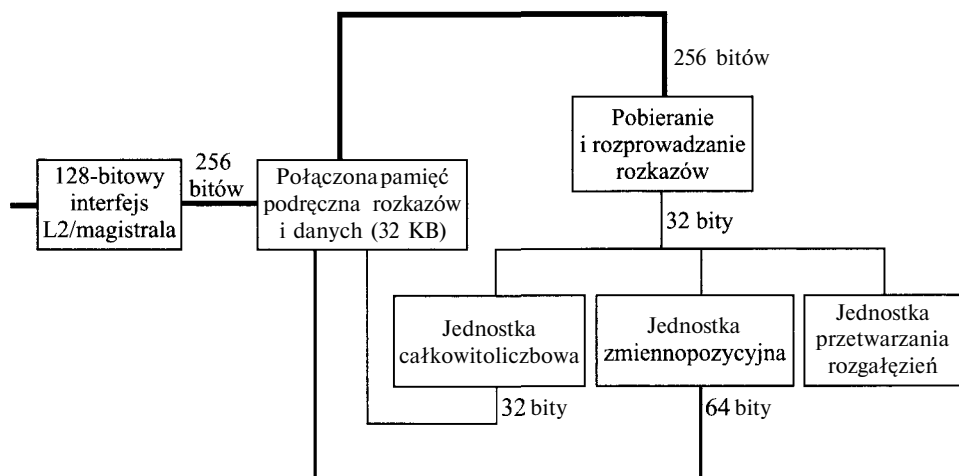
POWERPC

Architektura PowerPC jest bezpośrednim potomkiem IBM 801, RT PC oraz RS/6000, przy czym ostatnia z tych maszyn była również określana jako wdrożenie architektury POWER. Wszystkie te maszyny były maszynami RISC, jednak pierwszą wykazującą cechy superskalarne była RS/6000. W pierwszym modelu z serii PowerPC, a więc 601, zastosowano rozwiązanie superskalarne bardzo zbliżone do RS/6000. Następne modele PowerPC stanowiły dalsze rozwinięcie koncepcji superskalarnej. W tym podrozdziale skupimy się na modelu 601, który jest dobrym przykładem projektu superskalarnej opartego na RISC. Na koniec krótko przeanalizujemy model 620.

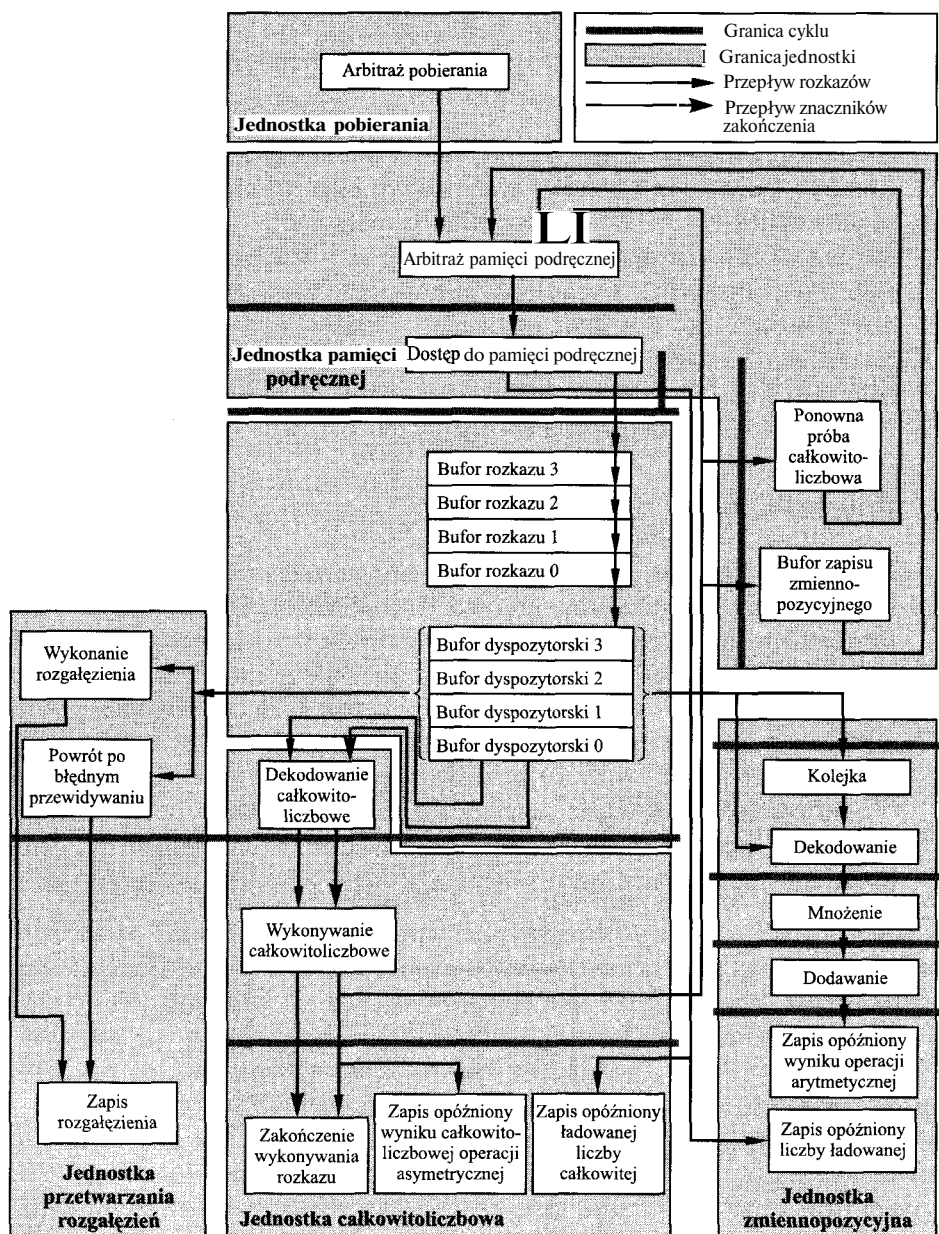
PowerPC 601

Na rysunku 13.5 jest pokazana w zarysie organizacja modelu 601. Podobnie jak inne maszyny superskalarne, procesor 601 składa się z niezależnych jednostek funkcjonalnych w celu zwiększenia możliwości równoległego wykonywania rozkazów. W szczególności rdzeń 601 stanowią trzy niezależne, przetwarzające potokowo jednostki wykonawcze: przetwarzania liczb całkowitych, przetwarzania liczb zmiennopozycyjnych i rozgałęzień. Łącznie jednostki te mogą jednocześnie wykonywać 3 rozkazy, co oznacza rozwiązanie superskalarne stopnia 3.

Na rysunku 13.6 widać schemat logiczny architektury 601; jest tu również pokazany przepływ rozkazów między jednostkami funkcjonalnymi. Jednostka pobierania może jednocześnie wstępnie pobierać z pamięci podręcznej do 8 roz-



RYSUNEK 13.5. Schemat blokowy procesora PowerPC 601



RYSUNEK 13.6. Struktura potoku procesora PowerPC 601 [POTT94]

kazów. Jednostka pamięci podręcznej obsługuje połączoną pamięć podręczną zarówno rozkazów, jak i danych. Jest ona odpowiedzialna za doprowadzanie rozkazów do innych jednostek oraz danych do rejestrów. Układy logiczne arbitrażu pamięci podręcznej wysyłają adres o najwyższym priorytecie dostępu do tej pamięci.

Jednostka dyspozytorska

Jednostka dyspozytorska pobiera rozkazy z pamięci podręcznej i ładuje je do kolejki dyspozytorskiej, w której może jednocześnie przebywać 8 rozkazów. Przetwarza ona ten strumień rozkazów, zapewniając ich stały dopływ do jednostek przetwarzania rozgałęzień, liczb całkowitych i zmiennopozycyjnych. Górna połowa kolejki działa po prostu jako bufor przetrzymujący rozkazy, zanim trafią one do dolnej połowy kolejki. Rolą górnej połowy jest zapewnienie tego, żeby jednostka dyspozytorska nie opóźniała się, czekając na rozkazy z pamięci podręcznej. W dolnej połowie kolejki rozkazy są rozprawdane zgodnie z następującym schematem:

- a *Jednostka przetwarzania rozgałęzień.* Przetwarza ona wszystkie rozgałęzienia. Najniżej położony rozkaz rozgałęzienia w dolnej połowie kolejki dyspozytorskiej jest wydawany do przetwarzania, jeśli tylko jednostka ta to akceptuje.
- a *Jednostka zmiennopozycyjna.* Wykonuje wszystkie rozkazy zmiennopozycyjne. Rozkaz zmiennopozycyjny najniżej położony w dolnej połowie kolejki dyspozytorskiej jest wydawany do tej jednostki, jeśli jej potok rozkazów nie jest pełny.
- a *Jednostka całkowitoliczbowa.* Wykonuje operacje na liczbach całkowitych, operacje ładowania/zapisu między tablicą rejestrów a pamięcią podręczną oraz operacje porównywania liczb całkowitych. Rozkaz dotyczący liczb całkowitych jest wydawany tylko wtedy, gdy dostanie się on na dno kolejki dyspozytorskiej.

Możliwość wydawania rozkazów rozgałęzienia i rozkazów zmiennopozycyjnych w zmienionym porządku z kolejki dyspozytorskiej pomaga utrzymywać wypełnienie potoków rozkazów w jednostkach przetwarzania rozgałęzień i zmiennopozycyjnej oraz ułatwia szybkie przechodzenie rozkazów przez kolejkę dyspozytorską.

Jednostka dyspozytorska zawiera także układy logiczne, które pozwalają jej obliczać adresy wstępnego pobierania. Kontynuuje ona sekwencyjne pobieranie rozkazów, aż rozkaz rozgałęzienia zostanie przeniesiony do dolnej połowy kolejki. Gdy jednostka przetwarzania rozgałęzień przetworzy rozkaz, może ona zaktualizować adres wstępnego pobierania, dzięki czemu następne rozkazy są pobierane spod nowego adresu i wprowadzane do kolejki dyspozytorskiej.

Potoki rozkazów

Na rysunku 13.7 są pokazane potoki rozkazów w różnych jednostkach funkcjonalnych. Cykl pobierania jest wspólny dla wszystkich rozkazów; pobieranie następuje, zanim rozkaz zostanie skierowany do określonej jednostki. Drugi cykl rozpoczyna się od skierowania rozkazu do określonej jednostki. Nakłada się on

Rozkazy
rozgałęzienia

Pobieranie	Rozprowadzanie Dekodowanie Wykonywanie Przewidywanie

Rozkazy
całkowitoliczbowe

Pobieranie	Rozprowadzenie Dekodowanie	Wykony- wanie	Zapis opóźniony

Rozkazy
ładowania/zapisu

Pobieranie	Rozprowadzenie Dekodowanie	Generowanie adresu	Pamięć podręczna	Zapis opóźniony

Rozkazy
zmiennopozycyjne

Pobieranie	Rozprowadzenie	Dekodo- wanie	Wykony- wanie 1	Wykony- wanie 2	Zapis opóźniony

RYSUNEK 13.7. Potok procesora PowerPC 601

z innymi czynnościami wewnątrz tej jednostki. Podczas każdego cyklu zegara jednostka dyspozytorska rozważa cztery dolne rozkazy kolejki i rozdysponowuje do trzech rozkazów.

W przypadku rozkazu rozgałęzienia drugi cykl obejmuje dekodowanie i wykonywanie rozkazów, jak również przewidywanie rozgałęzień. Tę ostatnią czynność omówimy w dalszym ciągu.

Jednostka całkowitoliczbowa ma do czynienia z rozkazami, które powodują operacje ładowania/zapisu z pamięcią (w tym również ładowanie/zapis liczb zmiennopozycyjnych), przesunięcia z rejestru do rejestru lub operacje ALU. W przypadku ładowania/zapisu ma miejsce cykl generowania adresu, po którym następuje wysłanie otrzymanego adresu do pamięci podręcznej i - jeśli to konieczne - cykl zapisu do rejestru. W przypadku pozostałych rozkazów pamięć podręczna nie jest angażowana i występuje tylko cykl wykonywania, a po nim zapis do rejestru.

Rozkazy zmiennopozycyjne są przetwarzane w podobnym potoku, jednak w tym potoku występują dwa cykle wykonywania, co odzwierciedla złożoność operacji zmiennopozycyjnych.

Warto odnotować kilka dodatkowych spostrzeżeń. Rejestr warunkowy zawiera 8 niezależnych 4-bitowych pól kodu warunkowego. Umożliwia to przechowywanie wielu kodów warunkowych, co redukuje blokowanie lub zależność między rozkazami. Kompilator może na przykład przekształcić sekwencję


```
    porównaj
    rozgałęzienie
    porównaj
    rozgałęzienie
    :
na sekwencję
    porównaj
    porównaj
    :
    rozgałęzienie
    rozgałęzienie
    :
```

Ponieważ każda jednostka funkcjonalna może wysłać swoje kody warunkowe do różnych pól w rejestrze warunkowym, można zapobiec blokowaniu między rozkazami spowodowanemu przez wspólne kody.

Istnienie rejestrów zapisu i odtworzenia (SRR) w procesorze rozgałęzień pozwala mu na przetwarzanie prostych przerw i przerw programowych bez angażowania układów logicznych innych jednostek funkcjonalnych.

Ponieważ procesor 601 może wydawać rozkazy rozgałęzienia i zmiennopozycyjne w zmienionej kolejności, wymagane jest sterowanie zapewniające właściwe wykonywanie tych rozkazów. Gdy ma miejsce zależność (tzn. gdy rozkaz potrzebuje argumentu, który nie został jeszcze obliczony przez poprzedni rozkaz), potok w odpowiedniej jednostce funkcjonalnej ulega zatrzymaniu.

Przetwarzanie rozgałęzień

Kluczem do wysokiej wydajności maszyny RISC lub superskalarnej jest jej zdolność do optymalizacji wykorzystania potoku. Zwykle najbardziej krytycznym elementem projektowania jest sposób przetwarzania rozgałęzień. W przypadku PowerPC za przetwarzanie rozgałęzień jest odpowiedzialna jednostka rozgałęzień. Jednostka ta została zaprojektowana w taki sposób, że w wielu przypadkach rozgałęzienia nie mają wpływu na przebieg wykonywania rozkazów w innych jednostkach. W celu osiągnięcia zero-cyklowego rozgałęziania zastosowano następujące strategie:

1. Wbudowano układy logiczne przeglądające bufor dyspozytorski w poszukiwaniu rozgałęzień. Generowane są docelowe adresy rozgałęzienia, gdy rozgałęzienie pojawia się po raz pierwszy w dolnej części kolejki i gdy żadne poprzednie rozgałęzienia nie zawiesiły wykonywania.
2. Dokonywana jest próba określenia wyniku rozgałęzienia warunkowego. Jeśli kod warunkowy został ustawiony z dostatecznym wyprzedzeniem, to wynik ten może być określony. W każdym przypadku tuż po napotkaniu rozkazu rozgałęzienia układy logiczne określają:

- (a) Czy rozgałęzienie nastąpi - dotyczy to rozgałęzień bezwarunkowych oraz tych rozgałęzień warunkowych, których kod warunkowy jest znany i wskazuje na rozgałęzienie.
- (b) Czy rozgałęzienie nie nastąpi - dotyczy to rozgałęzień warunkowych, których kod warunkowy jest znany i wskazuje, że rozgałęzienie nie nastąpi.
- (c) Czy wynik nie może jeszcze być określony. W tym przypadku zakłada się wystąpienie rozgałęzienia w odniesieniu do rozgałęzień wstecznych (typowych dla pętli) oraz że rozgałęzienie nie nastąpi w odniesieniu do rozgałęzień ku przodowi. Rozkazy sekwencyjne następujące po rozgałęzieniu są kierowane do jednostek funkcjonalnych w trybie warunkowym. Gdy tylko wartość kodu warunkowego zostanie określona w jednostce wykonującej, jednostka rozgałęzień albo kasuje rozkazy w potoku i kontynuuje działanie z pobranym adresem docelowym w przypadku wystąpienia rozgałęzienia, albo sygnalizuje, że mają być wykonywane rozkazy warunkowe. Kompilator może użyć jednego bitu w kodzie rozkazu w celu odwrócenia tej „zaocznej” prognozy.

Strategia przewidywania rozgałęzień na podstawie historii została odrzucona, ponieważ zdaniem projektantów daje ona minimalne korzyści.

Jako przykład rezultatu przewidywania rozgałęzień rozważmy program z rys. 13.8 i załóżmy, że procesor rozgałęzień przewiduje niewystąpienie rozgałęzienia warunkowego (ocena „zaoczna” dotycząca rozgałęzienia ku przodowi). Na rysunku 13.9a pokazano wpływ tego przewidywania na potok, jeśli rzeczywiście rozgałęzienie nie nastąpiło. Podczas pierwszego cyklu do kolejki dyspozytorskiej jest ładowanych 8 rozkazów. Sześć pierwszych rozkazów to rozkazy dotyczące liczb całkowitych i są one wydawane do jednostki liczb całkowitych w tempie jednego rozkazu na cykl. Rozkaz rozgałęzienia warunkowego nie może być rozdysponowany, zanim nie dotrze do dolnej części kolejki dyspozytorskiej, co następuje w cyklu 5. Jednostka rozgałęzień przewiduje, że to rozgałęzienie nie nastąpi, wobec tego rozdysponowany warunkowo jest następny rozkaz w sekwencji (oznaczony jako D'). Rozgałęzienie nie może być rozstrzygnięte, zanim nie zostanie wykonany rozkaz porównania w cyklu 8. Wtedy właśnie procesor rozgałęzień potwierdza prawidłowość przewidywania i wykonywanie jest kontynuowane. Nie występują opóźnienia i potok pozostaje pełny.

Zauważmy, że podczas cykli od 4 do 8 nie są pobierane żadne rozkazy. Dzieje się tak, ponieważ podczas tych cykli pamięć podręczna jest zajęta przez etapy dostępu do pamięci podręcznej dotyczące 5 rozkazów ładowania. Mimo to strumień rozkazów nie jest opóźniany, ponieważ kolejka dyspozytorska może zawierać 8 rozkazów.

Na rysunku 13.9b jest pokazany wpływ błędnego przewidywania rozgałęzienia (tzn. w rzeczywistości rozgałęzienie nastąpiło). W tym przypadku 3 rozkazy znajdujące się na etapie pobierania muszą być wyrzucone, a pobieranie jest wznawiane w odniesieniu do rozkazów umieszczonych po ELSE. W rezultacie

(a) if (a > 0)

a = a + b + c + d + e;

else a = a - b - c - d - e;

(b)

#r1 wskazuje na a

#r1+4 wskazuje na b

#r1+8 wskazuje na c

#r1+12 wskazuje na d

#r1+16 wskazuje na e

lwz r8=a (r1)

#ładuj a

lwz r12=b (r1, 4)

#ładuj b

lwz r9=c (r1, 8)

#ładuj c

lwz r10=d (r1, 12)

#ładuj d

lwz r11=e (r1, 16)

#ładuj e

cmpi

cr0=r8, 0

#porównaj natychmiastowe

bc ELSE, cr0>=false

#rozgałęziaj, jeśli nieprawdziwy

IF:

add r12=r8, r12

#dodaj

add r12=r12, r9

#dodaj

add r12=r12, r10

#dodaj

add r4=r12, r11

#dodaj

stw a (r1) =r4

#zapisz

b OUT

#rozgałęzienie bezwarunkowe

ELSE:

subf r12=r12, r8

#odejmij

subf r12=r9, r12

#odejmij

subf r12=r10, r12

#odejmij

subf r4=r11, r12

#odejmij

stw a (r1) =r4

#zapisz

OUT:

RYSUNEK 13.8. Przykład programu z rozgałęzieniem warunkowym [WEIS94]: (a) program w języku C; (b) program asemblerowy

etap wykonywania potoku jednostki liczb całkowitych pozostaje beczynny podczas cykli 9 i 10, co oznacza stratę 2 cykli spowodowaną przez błędne przewidywanie.

PowerPC 620

Procesor 620 jest pierwszą 64-bitową implementacją architektury PowerPC. Ogólny schemat blokowy tego procesora jest pokazany na rys. 4.26. Godną uwagi własnością tego wdrożenia jest występowanie sześciu niezależnych jednostek wykonawczych:

- a jednostki rozkazów,
- a trzech jednostek całkowitoliczbowych,
- a jednostki ładowania/zapisu,
- a jednostki zmiennopozycyjnej.

(a)		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	lwz r8=a (r1)	F	D	E	C	W											
	lwz r12=b (r1, 4)	F	.	D	E	C	W										
	lwz r9=c (r1, 8)	F	*	.	D	E	C	W									
	lwz r10=d(r1, 12)	F	«	.	.	D	E	C	W								
	lwz r11=e(r1, 16)	F	.	.	.	.	D	E	C	W							
	cmpi cr0=r8, ũ	F	.	.	.	.	.	D	E								
	bc	F	.	.	.	S											
	ELSE, cr0/gt=false																
IF:	add r12=r8, r12	F	.	.	.	.	.	.	D'	E	W						
	add r12=r12, r9				F	.	.	.	.	.	D	E	W				
	add r12=r12, r10				F	.	.	.	.	.	.	D	E	W			
	add r4=r12, r11										F	.	D	E	W		
	stw a (r1) =r4										F	.	.	D	E	C	
	b OUT																
ELSE:	subf r12=r8, r12																
	subf r12=r12, r9																
	subf r12=r12, r10																
	subf r4=r12, r11																
	stw a (r1) =r4																
OUT:																	

(b)		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	lwz r8=a (r1)	F	D	E	C	W											
	lwz r12=b (r1, 4)	F	.	D	E	C	W										
	lwz r9=c (r1, 8)	F	.	.	D	E	C	W									
	lwz r10=d (r1, 12)	F	.	.	.	D	E	C	W								
	lwz r11=e(r1, 16)	F	.	.	.	.	D	E	C	W							
	cmpi cr0=r8, ũ	F	.	.	.	.	.	D	E								
	bc	F	.	.	.	S											
	ELSE, cr0/gt=false																
IF:	add r12=r8, r12	F	.	.	.	.	.	.	D'								
	add r12=r12, r9				F	.	.	.	.	.							
	add r12=r12, r10				F	.	.	.	.	.							
	add r4=r12, r11																
	stw a (r1) =r4																
	b OUT																
ELSE:	subf r12=r8, r12									F	D	E	W				
	subf r12=r12, r9									F	.	D	E	W			
	subf r12=r12, r10									F	.	.	D	E	W		
	subf r4=r12, r11									F	.	.	.	D	E	W	
	stw a (r1) =r4									F	.	.	.	.	D	E	C
OUT:																	

F - pobieranie

D - rozprowadzanie/dekodowanie

E - wykonywanie/adres

C - dostęp do pamięci podręcznej

W - zapis opóźniony

S - rozprowadzenie

RYSUNEK 13.9. Przewidywanie rozgałęzienia: nie wystąpi [WEIS94]: (a) przewidywanie prawidłowe: rozgałęzienie nie wystąpiło; (b) przewidywanie nieprawidłowe: rozgałęzienie wystąpiło

Organizacja ta umożliwia procesorowi jednoczesne rozdysponowywanie do 4 rozkazów przeznaczonych dla 3 jednostek liczb całkowitych i jednostki zmiennopozycyjnej.

W procesorze 620 zastosowano wydajną strategię przewidywania rozgałęzień obejmującą układy logiczne przewidywania, bufory przemianowania rejestrów i stacje rezerwowe wewnątrz jednostek wykonawczych. Gdy rozkaz jest pobierany, przypisuje się go do buforu przemianowania w celu tymczasowego przechowywania wyników rozkazu, takich jak zawartości rejestrów. Dzięki zastosowaniu buforów przemianowania, procesor może wykonywać spekulatywnie rozkazy oparte na przewidywaniu rozgałęzienia; jeśli przewidywanie okaże się błędne, to wyniki spekulatywnych rozkazów mogą być wyrzucone bez naruszania tablicy rejestrów. Gdy natomiast wynik rozgałęzienia jest potwierdzony, tymczasowe wyniki mogą być zapisane na stałe.

Każda jednostka ma dwie lub więcej stacji rezerwowych, w których są przechowywane rozkazy, które czekają na wyniki innych rozkazów. Rozwiązanie to umożliwia usunięcie tych rozkazów z jednostki rozkazów, co pozwala jej na kontynuowanie rozdysponowywania rozkazów do innych jednostek wykonawczych.

Procesor 620 może spekulatywnie wykonywać do 4 nierozstrzygniętych rozkazów rozgałęzień (wobec jednego w procesorze 601). Przewidywanie rozgałęzień jest oparte na wykorzystaniu tablicy historii rozgałęzień zawierającej 2048 zapisów. Symulacje przeprowadzone przez projektantów PowerPC wykazały, że wskaźnik celności przewidywań wynosi 90% [THOM94].

13.4

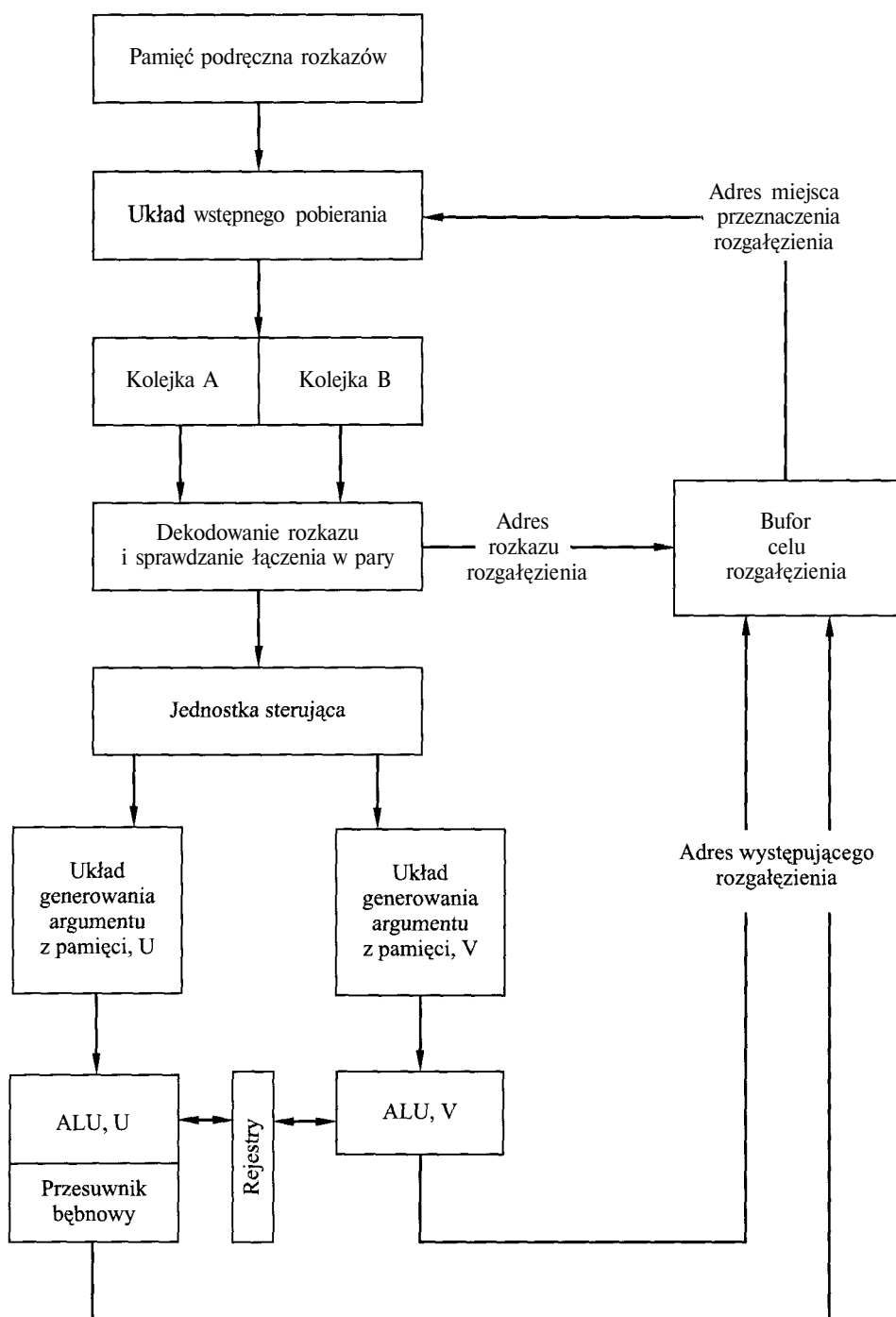
PENTIUM

Chociaż koncepcja projektowania superskalarne jest na ogół związana z architekturą RISC, te same rozwiązania superskalarne mogą być stosowane w architekturze CISC. Być może najbardziej godnym uwagi przykładem tego jest Pentium. W porównaniu z większością współczesnych projektów RISC, takich jak PowerPC 620, superskalarne właściwości Pentium są proste. Mimo to przyczyniają się one do istotnej poprawy wydajności.

Ogólny schemat blokowy Pentium został pokazany na rys. 4.24. Na rysunku 13.10 jest więcej szczegółów; uwypuklono te części procesora, które są istotne przy analizowaniu potokowego przetwarzania rozkazów.

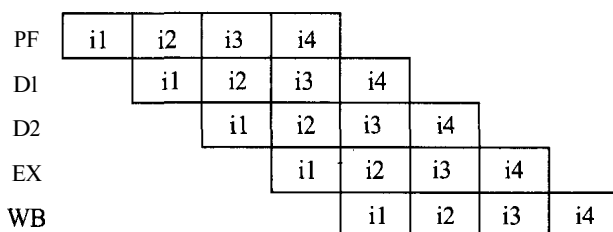
Podobnie jak procesor 80486, Pentium używa 5-etapowego potoku rozkazów dotyczących liczb całkowitych:

- a *Pobieranie wstępne.* Procesor pobiera wstępnie rozkazy z pamięci podręcznej rozkazów. Rozkazy mają zmienną długość i są przechowywane w buforze.

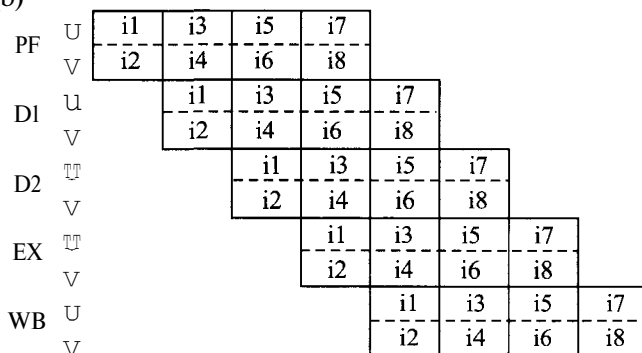


RYSUNEK 13.10. Struktura potoku całkowitoliczbowego procesora Pentium

(a)



(b)



RYSUNEK 13.11. Potoki procesorów: (a) Intel 80486; (b) Pentium

- a *Pierwszy etap dekodowania.* Procesor dekoduje rozkaz w celu określenia kodu operacji oraz informacji adresowej. Na tym etapie są również wykonywane dwie funkcje omówione poniżej: kontrola dobierania parami i przewidywanie rozgałęzień.
- *Drugi etap dekodowania.* Na tym etapie są generowane adresy związane z odniesieniami do pamięci.
- a *Wykonywanie.* Na tym etapie procesor albo sięga do danych w pamięci podręcznej danych, albo oblicza wyniki w ALU, w przesuwniku (ang. *barrel shifter*) lub w innych jednostkach wykonawczych na ścieżce danych.
- a *Zapis.* Na tym etapie aktualizowana jest zawartość rejestrów i znaczników stanu na podstawie wyników rozkazów.

Z punktu widzenia potokowego przetwarzania rozkazów najbardziej znacząca różnica między Pentium a jego poprzednikiem (80486) polega na tym, że Pentium zawiera dwie oddzielne jednostki wykonawcze liczb całkowitych, oznaczone jako U i V. Te dwa rozwiązania są porównane na rys. 13.11.

Dobieranie rozkazów parami

Jednostki U i V mogą równolegle wykonywać rozkazy. Procesor musi jednak stwierdzić, czy takie równoległe wykonywanie nie prowadzi do potencjalnych konfliktów. Pierwszym wymaganiem jakie musi być spełnione, aby rozkazy

mogły być wykonywane równolegle, jest to, że oba rozkazy muszą być proste. Rozkazami prostymi są rozkazy wykonywane całkowicie układowo; nie wymagają one sterowania mikroprogramowego i w większości są wykonywane w ramach jednego cyklu zegara. Wyjątkami są rozkazy ALU rejestr-pamięć i pamięć-rejestr, które wymagają 2 lub 3 cykli zegara, jednak używają porządkowania układowego, co umożliwia dobierania ich w pary. W tabeli 13.2 są wymienione rozkazy, które mogą być dobierane w pary.

TABELA 13.2. Proste rozkazy Pentium

MOV	reg, reg	shift	reg, imm
MOV	reg, mem	shift	mem, imm
MOV	reg, imm		
MOV	mem, reg	INC	reg
MOV	mem, imm	INC	mem
		DEC	reg
alu	reg, reg	DEC	mem
alu	reg, mem	Pusit	reg
alu	reg, imm	POP	reg
alu	mem, reg	LEA	reg, mem
alu	mem, imm	JMP	near
		CALL	near
shift	reg, 1	Jcc	near
shift	mem, 1	NOP	

alu - ADD, ADC, AND, OR, XOR, SUB, SBB, CMP, TEST

shift - SAL, SAR, SHL, SHR, RCL, RCR, ROL, ROR

(RCL i RCR nie są zestawiane w pary z argumentem natychmiastowym)

W książce dotyczącej Pentium (*Data Book*) [INTE94a] określono reguły dobierania parami. Dwa kolejne rozkazy I1 i I2 mogą być skierowane równolegle do jednostek U i V zgodnie z następującymi regułami:

1. Oba rozkazy muszą być rozkazami prostymi.
2. Nie występują zależności typu odczyt-po-zapisie lub zapis-po-zapisie. Oznacza to, że miejsce przeznaczenia rozkazu I1 nie jest źródłem rozkazu I2 ani jego miejscem przeznaczenia.
3. Żaden z rozkazów nie zawiera równocześnie przemieszczenia i argumentu natychmiastowego.
4. Tylko rozkaz I1 może zawierać przedrostek rozkazu.

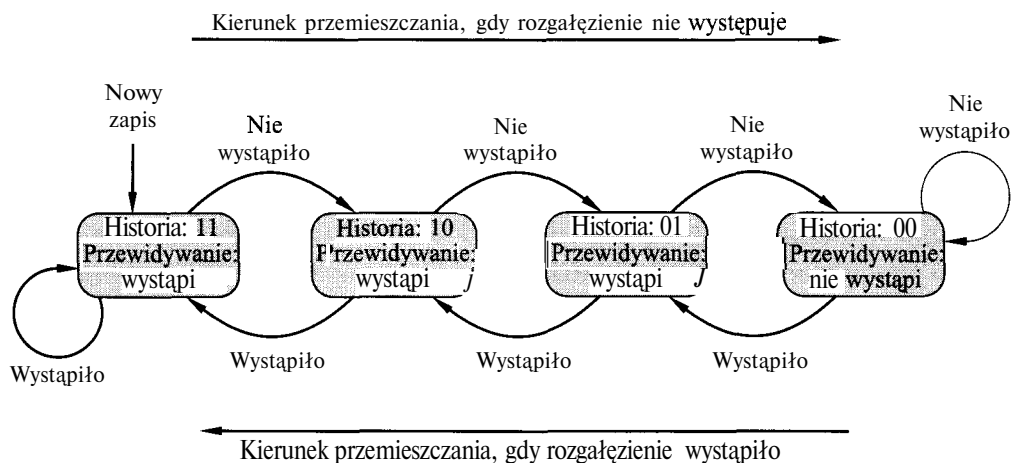
W ten sposób Pentium może osiągnąć stopień superskalarności równy 2. Z orientacyjnych wyników kilku niezależnych badań [POUN93] wynika, że wydajność Pentium w odniesieniu do programów całkowitoliczbowych niemal 2-krotnie przekracza wydajność procesora 80486. Wyniki te uzasadniają konkluzję, że bliźniacze superskalarne potoki Pentium stanowią decydującą nową architektoniczną tego procesora.

Przewidywanie rozgałęzień

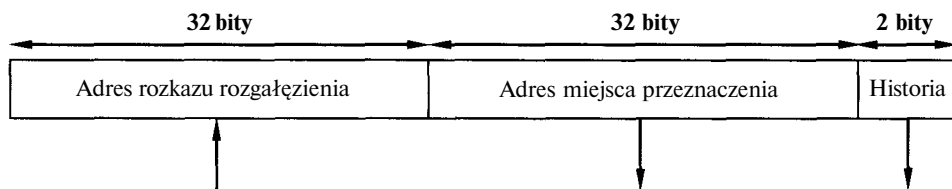
W Pentium zastosowano dynamiczną strategię przewidywania rozgałęzień opartą na historii ostatnich wykonań rozkazów rozgałęzienia. Na rysunku 13.12 widać ogólną sieć działań dotyczącą tego przewidywania. Zastosowano bufor adresów docelowych rozgałęzień (ang. *branch target buffer* – BTB) przechowujący informacje o ostatnio napotkanych rozkazach rozgałęzień. Gdy tylko w potoku rozkazów napotykan jest rozkaz rozgałęzienia, sprawdzany jest bufor BTB. Jeśli istnieje już zapis w buforze BTB, to jednostka rozkazów używa informacji historycznej związanej z tym zapisem w celu określenia, czy rozgałęzienie nastąpi. Jeśli przewidywane jest rozgałęzienie, to adres docelowy rozgałęzienia związany z tym zapisem służy do wstępnego pobierania docelowego rozkazu rozgałęzienia.

Gdy rozkaz jest wykonany, historyczna część odpowiedniego zapisu jest aktualizowana w celu odzwierciedlenia wyniku rozkazu rozgałęzienia. Jeśli rozkaz nie jest reprezentowany w buforze BTB, to adres tego rozkazu jest zapisywany w BTB; jeśli jest to konieczne, usuwany jest starszy zapis.

Gdy rozkazy są wstępnie pobrane, kieruje się je do jednej z dwóch kolejek rozkazów. W danej chwili czynna jest tylko jedna kolejka, a rozkazy są do niej wprowadzane i z niej wyprowadzane. Gdy przewidywane jest wystąpienie rozgałęzienia, kolejka aktualnie czynna jest zamrażana, a jednostka wstępnego pobierania rozpoczyna pobieranie rozkazów począwszy od adresu docelowego rozgałęzienia i kieruje je do drugiej kolejki. Jeśli przewidywanie okazało się prawidłowe, kontynuowana jest sekwencja rozkazów z aktualnie czynnej kolejki. Jeśli natomiast przewidywanie było błędne, to oba potoki są oczyszczane i procesor narażony jest na 3- lub 4-cykłowe opóźnienie, podczas gdy pobierany jest prawidłowy rozkaz docelowy.



RYSUNEK 13.12. Graf stanów bufora celu rozgałęzienia



RYSUNEK 13.13. Zapis bufora celu rozgałęzienia w procesorze Pentium

Bufor BTB jest zorganizowany w formie 4-etapowej sekcyjno-skojarzeniowej pamięci podręcznej o 256 wierszach. Każdy zapis (rys. 13.13) używa adresu rozkazu rozgałęzienia jako wskaźnika. Zapis zawiera także adres docelowy rozgałęzienia odnoszący się do ostatniego wystąpienia tego rozgałęzienia oraz 2-bitowe pole historii. Cztery możliwe stany tego pola odzwierciedlają najnowszą historię tego rozkazu. W nowym zapisie pole historii ma postać 11, a kolejne wykonania zmieniają to pole zgodnie z rys. 13.12. Procesor przewiduje, że rozgałęzienie nastąpi, chyba że wartość tego pola wynosi 11.

13.5

POLECANA LITERATURA

Praca [JOHN91] stanowi doskonałe książkowe ujęcie projektowania superskalarnego. W [JOUP89a] przeanalizowano paralelizm na poziomie rozkazu i różne metody maksymalizowania paralelizmu oraz porównano rozwiązania superpotokowe i superskalarne metodą symulacji.

Artykuł [POPE91] zawiera dokładne omówienie proponowanej maszyny superskalarnej oraz doskonały wykład zagadnień projektowania związanych ze zmianą kolejności rozkazów. Inne spojrzenie na proponowany system znajduje się w [KUGA91]; w tym artykule rozpatrzono większość spośród najważniejszych zagadnień projektowania odnoszących się do implementacji superskalarnych. W [LEE91] przeanalizowano metody programowe, które mogą być użyte do zwiększania wydajności superskalarnej. [WALL91] stanowi interesujące studium możliwego zasięgu wykorzystywania równoległości na poziomie rozkazu w procesorze superskalarnym. Pouczające porównanie potoków rozkazów w procesorach Intel 80486, MIPS R4000 i IBM RS/6000 zawiera [WAYN92]. Kształcącą analizę stosowania rozwiązania superskalarnego do wykonywania programów zorientowanych wektorowo można znaleźć w pracy [WEIS93].

Szczegółową analizę przetwarzania potokowego rozkazów w PowerPC 601 zawiera [POTT94]. Temat przetwarzania potokowego w procesorze 601 poruszono również w [RYAN93] i [BECK93], a także w [WEIS94]. PowerPC 620 opisano w [THOM94].

Dobry opis przetwarzania potokowego rozkazów w Pentium zawiera książka [ANDE93]. W [POUN93] skrytykowano aspekty projektowania superskalarnego w Pentium, a [SCHM94] zawiera interesującą analizę wydajności superskalarnej Pentium w odniesieniu do różnych fragmentów programów.

- ANDE93 Anderson D., Shanley T.: *Pentium Processor System Architecture*. Richardson, Mindshare Press, 1993.
- BECK93 Becker M. i in.: The PowerPC 601 Microprocessor. *IEEE Micro*, October 1993.
- JOHN91 Johnson M.: *Superscalar Microprocessor Design*. Englewood Cliffs, Prentice-Hall, 1991.
- JOUP89a Jouppi N., Wall D.: Available Instruction-Level Paralellism for Superscalar and Superpipelined Machines. *Proceedings, Third International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1989.
- KUGA91 Kuga M., Murakami K., Tomita S.: DSNS (Dynamically-hazard resolved, Statistically-code-scheduled, Nonuniform Superscalar): Yet Another Superscalar Processor Architecture. *Computer Architecture News*, June 1991.
- LEE91 Lee R., Kwok A., Briggs F.: The Floating Point Performance of a Superscalar SPARC Processor. *Proceedings, Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.
- POPE91 Popescu V. i in.: The Metaflow Architecture. *IEEE Micro*, June 1991.
- POTT94 Potter T. i in.: Resolution of Data and Control-Flow Dependencies in the PowerPC 601. *IEEE Micro*, October 1994.
- POUN93 Pountain D.: Pentium: More RISC than CISC. *Byte*, September 1993.
- RYAN93 Ryan B.: RISC Drives PowerPC. *Byte*, August 1993.
- SCHM94 Schmit M.: Optimizing Pentium Code. *Dr Dobbs's Journal*, January 1994.
- THOM94 Thompson T., Ryan B.: PowerPC 620 Soars. *Byte*, November 1994.
- WALL91 Wall D.: Limits of Instruction-Level Paralellism. *Proceedings, Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.
- WAYN92 Wayner P.: Processor Pipelines. *Byte*, January 1992.
- WEIS93 Weiss S.: Optimizing a Superscalar Machine to Run Vector Code. *IEEE Paralell and Distributed Technology*, May 1993.
- WEIS94 Weiss S., Smith J.: *POWER and PowerPC*. San Francisco, Morgan Kaufmann, 1994.

13.6

PROBLEMY DO ROZWIĄZANIA

- 13.1. Gdy w procesorze superskalarnym jest stosowana zmieniona kolejność kończenia rozkazów, wznawianie wykonywania po przetworzeniu przerwania jest skomplikowane, ponieważ warunek wyjątkowy może być wykryty jako rozkaz, który wytworzył swój wynik w zmienionej kolejności. Program nie może być wznowiony począwszy od rozkazu następującego po rozkazie, który spowodował wystąpienie wyjątku, ponieważ następne rozkazy zostały już wykonane i byłyby wykonane po raz drugi. Zasugeruj mechanizm lub mechanizmy postępowania w takiej sytuacji.
- 13.2. Rozważ następującą sekwencję rozkazów, których składnia obejmuje kolejno kod operacji, rejestr docelowy i jeden lub dwa rejestry źródłowe:

0	ADD	r3, r1, r2
1	LOAD	r6, [r3]
2	AND	r7, r5, 3
3	ADD	r1, r6, r7
4	SRL	r7, r0, 8
5	OR	r2, r4, r7
6	SUB	r5, r3, r4
7	ADD	r0, r1, 10
8	LOAD	r6, [r5]
9	SUB	r2, r1, r6
10	AND	r3, r7, 15

Założ użycie potoku 4-etapowego: pobranie, dekodowanie/wydanie, wykonanie, zapis. Założ, że wszystkie etapy potoku zajmują jeden cykl z wyjątkiem etapu wykonania. W przypadku prostych, całkowitoliczbowych rozkazów arytmetycznych i logicznych etap wykonania zajmuje jeden cykl, jednak w przypadku ładowania z pamięci na etapie wykonania zużywanych jest 5 cykli.

Jeśli mamy prosty potok skalarny, w którym jednak dozwolona jest zmiana kolejności wykonywania, to możemy zbudować następującą tabelę wykonywania pierwszych siedmiu rozkazów:

Rozkaz	Pobieranie	Dekodowanie	Wykonanie	Zapis
0	0	1	2	3
1	1	2	4	9
2	2	3	5	6
3	3	4	10	11
4	4	5	6	7
5	5	6	8	10
6	6	7	9	12

Zapisy w kolumnach oznaczających kolejne cztery etapy potoku wskazują cykl zegara, przy którym każdy z rozkazów rozpoczyna daną fazę. W tym programie drugi rozkaz ADD (rozkaz 3) zależy od rozkazu LOAD (rozkaz 1) ze względu na jeden z argumentów, r6. Ponieważ rozkaz LOAD zajmuje 5 cykli zegara, a układy logiczne wydawania napotykają rozkaz zależny ADD po 2 cyklach zegara, układy te muszą opóźnić rozkaz ADD o 3 cykle zegara. Dysponując możliwością zmiany kolejności, procesor może zatrzymać rozkaz 3 w cyklu zegara 4, a następnie przejść do wydawania następnych trzech rozkazów niezależnych, których wykonywanie rozpocznie się w 6, 8 i 9 cyklu zegara. Zakończenie wykonywania LOAD nastąpi w cyklu 9, dzięki czemu rozkaz zależny ADD może być wprowadzony do wykonywania w cyklu 10.

- (a) Uzupełnij powyższą tabelę.
- (b) Zbuduj od nowa tabelę zakładając, że zmiana kolejności nie jest dozwolona.

- (c) Zbuduj tabelę od nowa zakładając rozwiązanie superskalarne, w którym na każdym etapie można przetwarzać 2 rozkazy.
- 13.3. W kolejce rozkazów jednostki dyspozytorskiej procesora PowerPC 601 rozkazy mogą być wydawane w zmienionej kolejności jednostkom przetwarzania rozgałęzień i zmiennopozycyjnej, jednak rozkazy przeznaczone dla jednostki całkowitoliczbowej muszą być wydawane tylko z dołu kolejki. Skąd to ograniczenie?
- 13.4. Wykonaj rysunek podobny do 13.9 dla następujących przypadków:
- (a) przewidywanie rozgałęzienia: nastąpi; przewidywanie właściwe - rozgałęzienie nastąpiło;
 - (b) przewidywanie rozgałęzienia: nastąpi; przewidywanie błędne - rozgałęzienie nie nastąpiło.
- 13.5. Wyraź słowami regułę przewidywania implikowaną przez rys. 13.12. Bądź zwięzły.

JEDNOSTKA STERUJĄCA

Jednostka sterująca jest tą częścią procesora, która w rzeczywistości wywołuje zdarzenia. Jednostka sterująca wysyła sygnały sterowania na zewnątrz procesora w celu spowodowania wymiany danych z pamięcią i modułami wejścia-wyjścia. Jednostka sterująca wysyła również wewnętrzne sygnały sterowania procesorem w celu przenoszenia danych między rejestrami, spowodowania wykonania określonej funkcji przez ALU oraz w celu regulowania innych operacji wewnętrznych. Na wejście jednostki sterującej są podawane sygnały z rejestru rozkazu, znaczników stanu i sygnały sterujące ze źródeł zewnętrznych (np. sygnały przerwania).

W rozdziale 14 przeanalizujemy działanie jednostki sterującej, wyjaśniając pod względem funkcjonalnym jej czynności. Zobaczymy, że podstawową rolą jednostki sterującej jest powodowanie występowania sekwencji operacji elementarnych zwanych *mikrooperacjami* podczas trwania cyklu rozkazu. Następnie w rozdziale 15 powiemy, w jaki sposób koncepcja mikrooperacji prowadzi do eleganckiego i skutecznego wdrożenia jednostki sterującej.

Działanie jednostki sterującej

W rozdziale 9 stwierdziliśmy, że lista rozkazów maszynowych w znacznym stopniu określa procesor. Jeśli znamy listę rozkazów maszynowych, rozumiemy wyniki każdego kodu operacji, rozumiemy tryby adresowania i znamy zbiór rejestrów widzialnych dla użytkownika, znamy tym samym funkcje, które musi realizować procesor. Nie jest to jeszcze obraz całkowity. Musimy znać interfejsy zewnętrzne, dostępne zwykle za pośrednictwem magistrali, oraz sposób przetwarzania przerwań. Zgodnie z tym sposobem rozumowania, do określenia działania procesora muszą być sprecyzowane:

1. Operacje (kody operacji).
2. Tryby adresowania.
3. Rejestry.
4. Interfejs modułu wejścia-wyjścia.
5. Interfejs modułu pamięci.
6. Struktura przetwarzania przerwań.

Lista ta, chociaż ogólna, jest raczej kompletna. Punkty od 1 do 3 są definiowane poprzez określenie listy rozkazów. Punkty 4 i 5 są zwykle definiowane przez określenie magistrali systemowej. Punkt 6 jest definiowany częściowo przez magistralę systemową, a częściowo przez sposób, w jaki procesor wspiera system operacyjny.

Powyższa lista może definiować wymagania funkcjonalne w stosunku do procesora. Poszczególne punkty określają, co musi robić procesor. Tego właśnie dotyczyły części II i III. Teraz zajmiemy się problemem, jak funkcje te są wykonywane, lub - dokładniej - w jaki sposób różne elementy procesora są sterowane w celu wykonywania tych funkcji. Tak więc przechodzimy do analizowania jednostki sterującej, która kontroluje działanie procesora.

Mikrooperacje są elementarnymi operacjami wykonywanymi przez procesor. W tym podrozdziale przeanalizujemy mikrooperacje, aby lepiej zrozumieć, w jaki sposób zdarzenia składające się na cykl rozkazu mogą być opisane jako sekwencja takich mikrooperacji. Wykorzystamy prosty przykład pochodzący z [MANO93]. W pozostałej części rozdziału pokażemy, w jaki sposób koncepcja mikrooperacji może posłużyć jako przewodnik do projektowania jednostki sterującej.

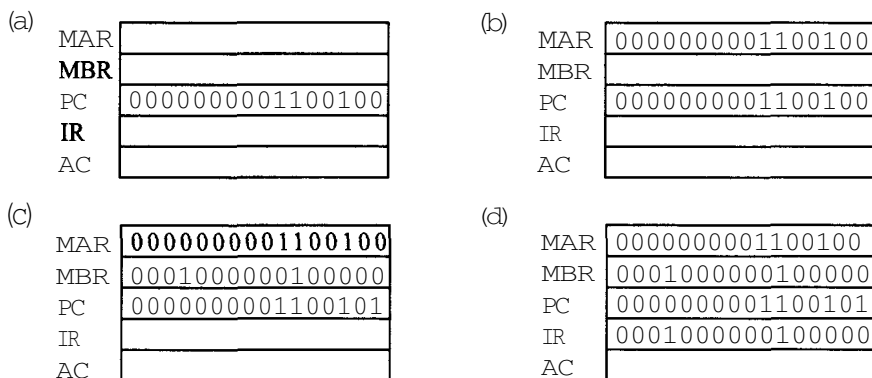
Cykl pobierania

Rozpocznijmy analizę od cyklu pobierania, który występuje na początku każdego cyklu rozkazu i w którym następuje pobranie rozkazu z pamięci. Dla celów dyskusji przyjmijmy schemat pokazany na rys. 11.8. Występują tam cztery rejestry:

- a **Rejestr adresowy pamięci (MAR).** Jest on podłączony do linii adresu magistrali systemowej. Określa adres w pamięci dla operacji odczytu lub zapisu.
- a **Rejestr buforowy pamięci (MBR).** Jest on podłączony do linii danych magistrali systemowej. Zawiera wartość, która ma być zapisana w pamięci lub która została ostatnio odczytana z pamięci.
- a **Licznik programu (PC).** Zawiera adres następnego rozkazu, który ma być wykonywany.
- a **Rejestr rozkazu (IR).** Zawiera ostatnio pobrany rozkaz.

Spójrzmy na sekwencję zdarzeń cyklu pobierania z punktu widzenia jej wpływu na rejestry procesora. Przykład widać na rys. 14.2. Na początku cyklu pobierania adres następnego rozkazu przewidzianego do wykonania znajduje się w liczniku programu (PC); w tym przypadku adresem jest 1100100. Pierwszym krokiem jest przeniesienie tego adresu do rejestru adresowego pamięci (MAR), ponieważ jest to jedyny rejestr podłączony do linii adresu magistrali systemowej. Żądany adres (zawarty w MAR) jest umieszczany na magistrali adresu, jednostka sterująca wydaje rozkaz READ (czytaj) za pomocą szyny sterowania, wynik pojawia się na szynie danych i jest kopiowany do rejestru buforowego pamięci (MBR). Musi także nastąpić inkrementacja rejestru PC w celu przygotowania się do pobrania następnego rozkazu. Ponieważ te dwa działania (odczyt słowa z pamięci i dodanie 1 do PC) nie kolidują ze sobą, dla zaoszczędzenia czasu możemy je wykonywać jednocześnie. Trzecim krokiem jest przeniesienie zawartości MBR do rejestru rozkazu (IR). Uwalnia to MBR, umożliwiając jego użycie w ewentualnym cyklu adresowania pośredniego.

W rzeczywistości prosty cykl pobierania składa się więc z 3 kroków i z 4 mikrooperacji. Każda mikrooperacja obejmuje przenoszenie danych do rejestru lub z rejestru. Tak długo, jak te przesunięcia danych nie kolidują ze sobą, możliwe jest oszczędzanie czasu przez wykonywanie ich w jednym kroku. Symbolicznie możemy zapisać tę sekwencję zdarzeń następująco:



RYСУNEK 14.2. Sekwencja zdarzeń w cyklu pobierania: (a) początek; (b) krok pierwszy; (c) krok drugi; (d) krok trzeci

$t_1 : \text{MAR} \leftarrow (\text{PC})$
 $t_2 : \text{MBR} \leftarrow \text{Pamięć}$
 $\text{PC} \leftarrow (\text{PC}) + 1$
 $t_3 : \text{IR} \leftarrow (\text{MBR})$

Koniecznych jest kilka komentarzy dotyczących tej sekwencji. Zakładamy, że do taktowania dostępny jest zegar wysyłający regularne impulsy. Każdy impuls zegara określa jednostkę czasu. Wobec tego wszystkie jednostki czasu mają tę samą długość. Każda **mikrooperacja** może być wykonana w ciągu jednej jednostki czasu. Zapisy (t_1 , t_2 , t_3) reprezentują kolejne jednostki czasu. Można to wyrazić słownie:

- *Pierwsza jednostka czasu (t_1).* Przeniesienie zawartości PC do MAR.
- a *Druga jednostka czasu (t_2).* Przeniesienie zawartości lokacji pamięci określonej w MAR do MBR. Zwiększenie zawartości PC o 1.
- a *Trzecia jednostka czasu (t_3).* Przeniesienie zawartości MBR do IR.

Zauważmy, że **mikrooperacje** druga i trzecia mają miejsce podczas drugiej jednostki czasu. Trzecia mikrooperacja mogłaby być połączona (zgrupowana) z czwartą bez naruszania operacji pobierania:

$t_1 : \text{MAR} \leftarrow (\text{PC})$
 $t_2 : \text{MBR} \leftarrow \text{Pamięć}$
 $t_3 : \text{PC} \leftarrow (\text{PC}) + 1$
 $\text{IR} \leftarrow (\text{MBR})$

Grupowanie mikrooperacji musi być zgodne z dwiema prostymi **regułami**:

1. Musi być zachowana właściwa sekwencja zdarzeń. Dlatego ($\text{MAR} \leftarrow (\text{PC})$) musi poprzedzać ($\text{MBR} \leftarrow \text{Pamięć}$), ponieważ operacja odczytu pamięci używa adresu zawartego w MAR.

2. Nie można dopuścić do konfliktów. Nie powinno się próbować odczytu i zapisu w tym samym rejestrze w tej samej jednostce czasu, ponieważ wyniki mogą być nieprzewidywalne. Na przykład mikrooperacje ($MBR \leftarrow \leftarrow \text{Pamięć}$) i ($IR \leftarrow MBR$) nie powinny następować w tej samej jednostce czasu.

Na zakończenie warto zauważyć, że jedna z mikrooperacji zawiera dodawanie. Aby uniknąć dublowania układów, dodawanie to mogłoby być wykonywane przez ALU. Jednak użycie ALU może spowodować wprowadzenie dodatkowych mikrooperacji, zależnie od funkcjonalności ALU i organizacji procesora. Odłożymy dyskusję na ten temat do dalszej części rozdziału.

Pożyteczne jest porównanie zdarzeń opisanych w tym i w następnym punkcie z rys. 3.5. Na tym rysunku mikrooperacje zostały zignorowane, nasza dyskusja wskazuje mikrooperacje potrzebne do przeprowadzenia podcykli składających się na cykl rozkazu.

Cykl adresowania pośredniego

Gdy rozkaz został już pobrany, następnym krokiem jest pobranie argumentów źródłowych. Kontynuując nasz prosty przykład, założmy jednoadresowy format rozkazu z dozwolonym adresowaniem bezpośrednim i pośrednim. Jeśli rozkaz określa adres pośredni, to cykl wykonywania musi być poprzedzony cyklem adresowania pośredniego. Przepływ danych jest pokazany na rys. 11.9 i obejmuje następujące mikrooperacje:

$t_1 : MAR \leftarrow (IR(\text{Adres}))$
 $t_2 : MBR \leftarrow \text{Pamięć}$
 $t_3 : IR(\text{Adres}) \leftarrow (MBR(\text{Adres}))$

Zawartość pola adresowego rozkazu jest przenoszona do rejestru MAR. Następnie jest używana do pobrania adresu argumentu. Na zakończenie pole adresu IR jest aktualizowane za pomocą zawartości rejestru MBR, dzięki czemu obecnie zawiera adres bezpośredni zamiast pośredniego.

Stan rejestru IR jest obecnie taki sam, jak gdyby adresowanie pośrednie nie było stosowane, i jest on gotowy do cyklu wykonywania. Pomińmy chwilowo ten cykl i rozpatrzmy cykl przerwania.

Cykl przerwania

Na zakończenie cyklu wykonywania przeprowadzany jest test mający na celu sprawdzenie, czy nie wystąpiły jakiekolwiek dozwolone przerwania. Jeśli wystąpiły, to następuje cykl przerwania. Natura tego cyklu różni się znacznie w zależności od komputera. Przedstawiamy bardzo prostą sekwencję zdarzeń zilustrowaną na rys. 11.10. Mamy:

$t_1 : \text{MBR} \leftarrow (\text{PC})$
 $t_2 : \text{MAR} \leftarrow \text{Adres zachowywany}$
 $\text{PC} \leftarrow \text{Adres programu obsługi}$
 $t_3 : \text{Pamięć} \leftarrow (\text{MBR})$

W ramach pierwszego kroku zawartość licznika PC jest przenoszona do rejestru MBR, dzięki czemu może ona być zachowana do powrotu z przerwania. Następnie do rejestru MAR jest ładowany adres, pod którym ma być zachowana zawartość PC, a do PC jest ładowany adres początkowy programu obsługi przerwania. Każde z tych dwóch działań może być pojedynczą mikrooperacją. Ponieważ jednak większość procesorów umożliwia wiele rodzajów i (lub) poziomów przerwania, dostęp do obu adresów przed ich przeniesieniem do MAR i PC może wymagać jednej lub wielu dodatkowych mikrooperacji. W każdym przypadku, gdy jest to już wykonane, krokiem końcowym jest zapis w pamięci starej wartości PC zawartej w MBR. Procesor jest teraz gotowy do rozpoczęcia następnego cyklu rozkazu.

Cykl wykonywania

Cykle pobierania, adresowania pośredniego i przerwania są proste i przewidywalne. Każdy z nich obejmuje niewielką, ustaloną sekwencję mikrooperacji i w każdym przypadku powtarzane są te same mikrooperacje.

Nie dotyczy to cyklu wykonywania. W przypadku maszyny o N różnych kodach operacji, może wystąpić N różnych sekwencji mikrooperacji. Opiszemy kilka hipotetycznych przykładów.

Najpierw rozważmy rozkaz dodawania:

ADD R1, X

który powoduje dodanie zawartości lokacji X do rejestru R1. Podczas wykonywania tego rozkazu mogłaby wystąpić następująca sekwencja mikrooperacji:

$t_1 : \text{MAR} \leftarrow (\text{IR}(\text{Adres}))$
 $t_2 : \text{MBR} \leftarrow \text{Pamięć}$
 $t_3 : \text{R1} \leftarrow (\text{R1}) + (\text{MBR})$

Na początku rejestr IR zawiera rozkaz dodawania. Podczas pierwszego kroku część adresowa rejestru IR jest ładowana do rejestru MAR. Następnie jest odczytywana odpowiednia lokacja w pamięci. Na zakończenie ALU dodaje zawartości R1 i MBR. Mogą być wymagane dodatkowe mikrooperacje w celu wydobycia odniesienia do rejestru z rejestru IR i być może w celu zatrzymania danych wejściowych i wyjściowych ALU w pewnych pośrednich rejestrach.

Zapoznajmy się teraz z dwoma bardziej złożonymi przykładami. Powszechnie spotykanym rozkazem jest rozkaz inkrementacji i pominięcia w przypadku zera (ang. *increment and skip if zero*):

ISZ X

Zawartość lokacji X jest zwiększana o 1. Jeśli wynikiem jest O, następny rozkaz jest pomijany. Możliwa jest następująca sekwencja mikrooperacji:

$t_1 : \text{MAR} \leftarrow (\text{IR}(\text{Adres}))$

$t_2 : \text{MBR} \leftarrow \text{Pamięć}$

$t_3 : \text{MBR} \leftarrow (\text{MBR}) + 1$

$t_4 : \text{Pamięć} \leftarrow (\text{MBR})$

Jeśli $(\text{MBR} = 0)$ to $(\text{PC} \leftarrow (\text{PC}) + 1)$

Nową wprowadzoną tutaj właściwością jest działanie warunkowe. Stan licznika PC jest zwiększany, jeśli $\text{MBR} = 0$. Ten test oraz towarzyszące mu działanie mogą być wdrożone w postaci jednej mikrooperacji. Zauważmy także, że ta mikrooperacja może być przeprowadzona w ciągu tej samej jednostki czasu, podczas której zaktualizowana zawartość rejestru MBR jest zapisywana w pamięci.

Na zakończenie rozważmy rozkaz wywołania podprogramu. Jako przykład posłuży nam rozkaz rozgałęziania i zachowania adresu (ang. *branch-and-save-address*):

BSA X

Adres rozkazu, który następuje po rozkazie BSA, jest zachowywany w lokacji X, a wykonywanie jest kontynuowane począwszy od lokacji $X + 1$. Zachowany adres będzie następnie użyty do powrotu. Jest to prosta metoda wywołania podprogramów. Wystarczy następująca sekwencja mikrooperacji:

$t_1 : \text{MAR} \leftarrow (\text{IR}(\text{Adres}))$

$\text{MBR} \leftarrow (\text{PC})$

$t_2 : \text{PC} \leftarrow (\text{IR}(\text{Adres}))$

$\text{Pamięć} \leftarrow (\text{MBR})$

$t_3 : \text{PC} \leftarrow (\text{PC}) + 1$

Adres znajdujący się w liczniku PC na początku rozkazu jest adresem następnego rozkazu w sekwencji. Jest on zachowywany pod adresem wyznaczonym przez rejestr IR. Następuje również zwiększenie tego ostatniego adresu w celu określenia adresu rozkazu związanego z następnym cyklem rozkazu.

Cykl rozkazu

Stwierdziliśmy, że każda faza cyklu rozkazu może być traktowana jako sekwencja elementarnych mikrooperacji. W naszym przykładzie występuje po jednej sekwencji na cykle pobierania, adresowania pośredniego i przerwania, oraz w odniesieniu do cyklu wykonywania - po jednej sekwencji mikrooperacji dla każdego kodu operacji.

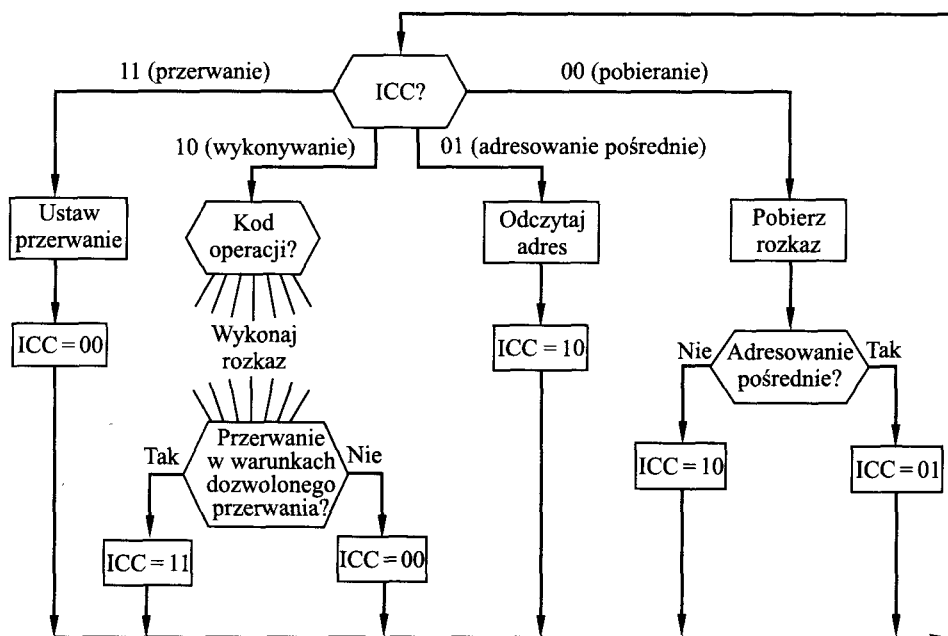
W celu uzupełnienia obrazu musimy powiązać ze sobą te sekwencje mikrooperacji, co zostało pokazane na rys. 14.3. Zakładamy, że istnieje nowy,

2-bitowy rejestr nazywany kodem cyklu rozkazu (ang. *instruction cycle code* - ICC). Rejestr ICC wyznacza stan procesora określając, w której części cyklu się on znajduje:

- 00: Pobieranie
- 01: Adresowanie pośrednie
- 10: Wykonywanie
- 11: Przerwanie

Na końcu każdego z czterech cykli następuje odpowiednie ustawienie rejestru ICC. Po cyklu adresowania pośredniego następuje zawsze cykl wykonywania. Po cyklu przerwania następuje zawsze cykl pobierania (patrz rys. 11.6). W przypadku cykli wykonywania i pobierania następny cykl zależy od stanu systemu.

Sieć działań na rys. 14.3 określa więc całkowitą sekwencję mikrooperacji, zależną jedynie od sekwencji rozkazów i przebiegu przerw. Oczywiście jest to przykład uproszczony. Sieć działań dla rzeczywistego procesora byłaby bardziej złożona. W naszej dyskusji osiągnęliśmy jednak taki stan, w którym działanie procesora jest określone jako wykonywanie sekwencji mikrooperacji. Rozpatrzmy teraz, w jaki sposób jednostka sterująca powoduje realizowanie tej sekwencji.



RYSUNEK 14.3. Sieć działań cyklu rozkazu

14.2

STEROWANIE JEDNOSTKĄ CENTRALNĄ**Wymagania funkcjonalne**

W wyniku analizy dokonanej w poprzednim podrozdziale zdekomponowaliśmy zachowanie (lub funkcjonowanie) procesora na operacje elementarne, zwane **mikrooperacjami**. Powodem naszego postępowania była chęć określenia natury jednostki sterującej. Poprzez zredukowanie działania procesora do najbardziej podstawowego poziomu, jesteśmy w stanie dokładnie zdefiniować, jakie działania musi powodować jednostka sterująca. Możemy więc określić *wymagania funkcjonalne* odnoszące się do jednostki sterującej: te funkcje, które musi realizować jednostka sterująca. Określenie tych wymagań funkcjonalnych jest podstawą projektowania i wdrażania jednostki sterującej.

Na podstawie tych informacji możemy przeprowadzić trzy etapowy proces zmierzający do scharakteryzowania jednostki sterującej:

1. Określenie podstawowych elementów procesora.
2. Opisanie mikrooperacji wykonywanych przez procesor.
3. Określenie funkcji, które musi realizować jednostka sterująca w celu spowodowania przeprowadzenia tych mikrooperacji.

Mamy już za sobą kroki 1 i 2. Podsumujmy ich wyniki. Po pierwsze, podstawowymi elementami funkcjonalnymi procesora są:

- ☐ ALU,
- ☐ rejestry,
 - a wewnętrzne ścieżki danych,
 - Q zewnętrzne ścieżki danych,
- ☐ jednostka sterująca.

Odrobina przemyśleń powinna przekonać czytelnika, że lista ta jest kompletna. ALU jest funkcjonalną istotą komputera. Rejestry są używane do przechowywania danych wewnętrznych w procesorze. Niektóre rejestry zawierają informacje o stanie potrzebne do zarządzania porządkowaniem rozkazów (np. słowo stanu programu). Pozostałe zawierają dane przeznaczone dla ALU, pamięci i modułów wejścia-wyjścia lub przekazane przez te jednostki. Wewnętrzne ścieżki danych są używane do przenoszenia danych między rejestrami oraz między rejestrem a ALU. Zewnętrzne ścieżki danych łączą rejestry z pamięcią i modułami wejścia-wyjścia, często za pomocą magistrali systemowej. Jednostka sterująca powoduje wykonywanie **operacji** wewnątrz procesora.

Wykonywanie programu składa się z operacji **angażujących** te elementy procesora. Jak widzieliśmy, operacje te składają się z sekwencji mikrooperacji. Przeglądając podrozdz. 14.1, powinno się zauważyć, że wszystkie mikrooperacje dzielą się na następujące kategorie:

- *Transfer danych z jednego rejestru do drugiego.*
- a *Transfer danych z rejestru do interfejsu zewnętrznego* (np. do magistrali systemowej).
- Q *Transfer danych z interfejsu zewnętrznego do rejestru.*
- Q *Wykonywanie operacji arytmetycznej lub logicznej*, przy czym do przechowania danych wejściowych i wyjściowych służą rejestry.

Wszystkie operacje wymagane do zrealizowania jednego cyklu rozkazu, łącznie ze wszystkimi mikrooperacjami potrzebnymi do wykonania każdego rozkazu z listy rozkazów, kwalifikują się do jednej z tych kategorii.

Możemy już teraz w pewnym stopniu wyjaśnić sposób funkcjonowania jednostki sterującej. Realizuje ona dwa podstawowe zadania:

- *Porządkowanie.* Jednostka sterująca powoduje, że procesor wykonuje ciąg mikrooperacji we właściwej kolejności (sekwencji) opartej na wykonywanym programie.
- Q *Wykonywanie.* Jednostka sterująca powoduje wykonanie każdej mikrooperacji.

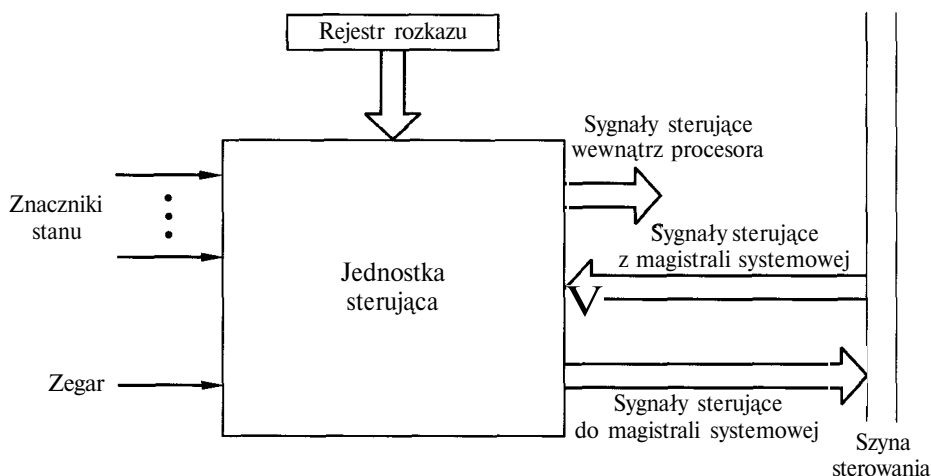
Przedstawiliśmy opis funkcjonalny tego, co wykonuje jednostka sterująca. Kluczem do jej funkcjonowania jest użycie sygnałów sterujących.

Sygnały sterujące

Określiliśmy elementy procesora (ALU, rejestry, ścieżki danych) oraz wykonywane mikrooperacje. Aby jednostka sterująca mogła realizować swoje funkcje, musi dysponować danymi wejściowymi pozwalającymi jej na określenie stanu systemu i wyjścia umożliwiające sterowanie zachowaniem systemu. Są to właściwości zewnętrzne jednostki sterującej. Wewnątrz musi ona dysponować układami logicznymi wymaganymi do pełnienia funkcji porządkowania i wykonywania. Rozważania dotyczące wewnętrznego działania jednostki sterującej odłożymy do podrozdz. 14.3 i rozdz. 15. W pozostałej części tego podrozdziału zajmiemy się oddziaływaniem między jednostką sterującą a pozostałymi elementami procesora.

Na rysunku 14.4 jest pokazany ogólny model jednostki sterującej wraz ze wszystkimi wejściami i wyjściami. Źródłami danych wejściowych jednostki sterującej są:

- *Zegar.* Dzięki niemu działanie jednostki sterującej jest związane z przebiegiem czasu. Jednostka sterująca powoduje wykonywanie jednej mikrooperacji (lub zbioru jednoczesnych mikrooperacji) po każdym impulsie zegarowym. Jest to czasami określane jako czas cyklu procesora lub czas cyklu zegara.
- *Rejestr rozkazu.* Do określenia, które mikrooperacje mają być wykonane podczas cyklu wykonywania, używany jest kod operacji bieżącego rozkazu.



RYSUNEK 14.4. Model jednostki sterującej

- a *Znaczniki stanu.* Są one potrzebne jednostce sterującej do określania stanu procesora i wyników poprzednich operacji ALU. Na przykład, wykonując rozkaz zwiększenia stanu i pominięcia w przypadku zera (ISZ), jednostka sterująca zwiększy stan licznika PC, jeśli jest ustawiony znacznik stanu 0.
- ❑ *Magistrala sterowania.* Magistrala sterowania stanowiąca część magistrali systemowej przekazuje sygnały jednostce sterującej, takie jak sygnały przezwania i potwierdzenia.

Natomiast na wyjściu pojawiają się:

- a *Wewnętrzne sygnały sterujące procesora.* Istnieją dwa ich rodzaje: te, które powodują przenoszenie danych z jednego rejestru do drugiego, oraz te, które aktywują określone funkcje ALU.
- Q *Sygnały sterujące do magistrali sterowania.* Również istnieją dwa rodzaje tych sygnałów: sygnały sterujące pamięcią i sygnały sterujące modułami wejścia-wyjścia.

Nowym elementem wprowadzonym na rysunku jest sygnał sterujący. Używane są trzy rodzaje sygnałów sterujących: aktywujące działanie ALU, aktywujące ścieżkę danych oraz te, które znajdują się na zewnętrznej magistrali systemowej lub w innym interfejsie zewnętrznym. Wszystkie te sygnały są z zasady doprowadzane bezpośrednio jako wejściowe sygnały binarne do pojedynczych bramek logicznych.

Ponownie rozważmy cykl pobierania w celu zorientowania się w sposobie sterowania realizowanym przez jednostkę sterującą. Jednostka sterująca śledzi etapy cyklu rozkazu. W danej chwili wie ona na przykład, że jako następny będzie realizowany cykl pobierania. Pierwszym krokiem jest przeniesienie zawartości licznika PC do rejestru MAR. Jednostka sterująca dokonuje tego przez podanie sygnału sterującego, który otwiera bramki między bitami licznika PC

a bitami rejestru MAR. Następnym krokiem jest wczytanie słowa z pamięci do MBR i zwiększenie stanu PC. Jednostka sterująca dokonuje tego przez jednocześnie wysłanie następujących sygnałów sterujących:

1. Sygnału sterującego, który otwiera bramki, pozwalając na przekazanie zawartości MAR do magistrali adresu.
2. Sygnału sterującego odczytem pamięci do magistrali sterowania.
3. Sygnału sterującego, który otwiera bramki, pozwalając na przekazanie zawartości magistrali danych do rejestru MBR.
4. Sygnału sterującego do układów logicznych, które dodają 1 do zawartości PC i zapisują wynik w PC.

Następnie jednostka sterująca wysyła sygnał sterujący, który otwiera bramki między MBR a IR.

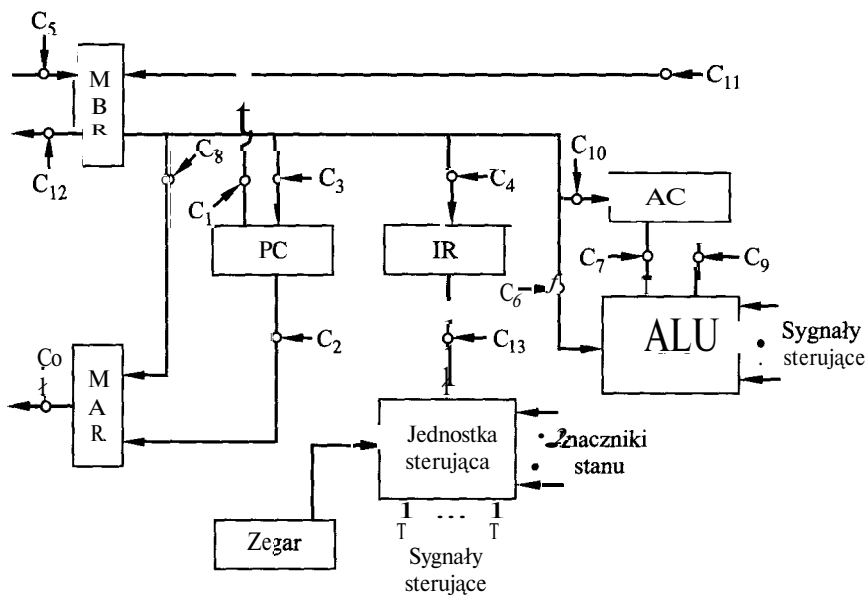
Kończy to cykl pobierania z wyjątkiem jednej rzeczy: jednostka sterująca musi zdecydować, czy jako następny ma być realizowany cykl adresowania pośredniego, czy cykl wykonywania. W celu podjęcia tej decyzji jednostka sterująca analizuje zawartość rejestru IR, aby stwierdzić, czy ma miejsce pośrednie odniesienie do pamięci.

Cykle adresowania pośredniego i przerwania są realizowane podobnie. W przypadku cyklu wykonywania jednostka sterująca rozpoczyna od analizy kodu operacji i na tej podstawie decyduje, która sekwencja mikrooperacji ma być realizowana w tym cyklu.

Przykład sygnałów sterujących

Aby zilustrować funkcjonowanie jednostki sterującej, przeanalizujmy prosty przykład, zaadaptowany z [ANDR80]. Przykład ten widać na rys. 14.5. Jest to prosty procesor z pojedynczym akumulatorem. Na rysunku są pokazane ścieżki danych między elementami. Ścieżki sterowania sygnałów wysyłanych przez jednostkę sterującą nie zostały pokazane, jednak końcówki sygnałów sterowania oznaczono przez C_i i wyróżniono za pomocą kółek. Jednostka sterująca otrzymuje sygnały wejściowe z zegara, z rejestru rozkazu oraz znaczniki stanu. W każdym cyklu zegara jednostka sterująca odczytuje wszystkie swoje wejścia i wysyła zbiór sygnałów sterujących. Sygnały sterujące są kierowane do:

- Q *Ścieżki danych.* Jednostka sterująca steruje wewnętrznym przepływem danych. Na przykład, w czasie pobierania rozkazu zawartość rejestru buforowego pamięci jest przenoszona do rejestru rozkazu. W każdej kontrolowanej ścieżce istnieje bramka (reprezentowana przez kółko na rysunku). Sygnał sterujący z jednostki sterującej okresowo otwiera bramkę, umożliwiając przepływ danych.
- a *ALU.* Jednostka sterująca kontroluje pracę ALU za pomocą zbioru sygnałów sterujących. Sygnały te aktywują różne elementy logiczne i bramki wewnątrz ALU.



RYSUNEK 14.5. Ścieżki danych i sygnały sterujące

a *Magistrali systemowej.* Jednostka sterująca wysyła sygnały sterujące na zewnątrz za pośrednictwem linii sterowania magistrali systemowej (np. przy odczycie pamięci).

Jednostka sterująca musi wciąż wiedzieć, jaki **fragment** cyklu rozkazu jest aktualnie realizowany. Dysponując tą wiedzą i odczytując wszystkie swoje wejścia, jednostka sterująca wysyła sekwencję sygnałów sterujących, które powodują wykonywanie mikrooperacji. Używa ona impulsów zegarowych do taktowania sekwencji zdarzeń, rezerwując czas między zdarzeniami na stabilizowanie się poziomów sygnałów. W tabeli 14.1 są podane sygnały sterujące potrzebne dla

TABELA 14.1. Mikrooperacje i sygnały sterujące

Mikrooperacje		Aktywne sygnały sterujące
Pobieranie	$t_1: MAR \leftarrow (PC)$	C_2
	$t_2: MBR \leftarrow \text{Pamięć}$	C_5, C_R
	$PC \leftarrow (PC) + 1$	
	$t_3: IR \leftarrow (MBR)$	C_4
Cykl pośredni	$t_1: MAR \leftarrow (IR(adres))$	C_8
	$t_2: MBR \leftarrow \text{Pamięć}$	C_5, C_R
	$t_3: IR(adres) \leftarrow (MBR(adres))$	C_4
Przerwanie	$t_1: MBR \leftarrow (PC)$	C_1
	$t_2: MAR \leftarrow \text{adres zachowywany}$ $PC \leftarrow \text{adres podprogramu}$	
	$t_3: \text{Pamięć} \leftarrow (MBR)$	C_2, C_w

CR - sygnał sterujący odczytu do magistrali systemowej
Cw - sygnał sterujący zapisu do magistrali systemowej

wcześniej opisanych sekwencji mikrooperacji. Dla uproszczenia pominięto ścieżki danych i sterowania służące do zwiększania stanu licznika PC i do ładowania ustalonych adresów do rejestrów PC i MAR.

Godna zastanowienia jest minimalistyczna natura jednostki sterującej. Jednostka ta jest „silnikiem” napędzającym cały komputer. Dokonuje tego tylko na podstawie znajomości przewidzianych do wykonania rozkazów oraz natury wyników operacji arytmetycznych i logicznych (np. dodatni, przepełnienie itd.). Nigdy nie zna ona przetwarzanych danych lub rzeczywistych wyników. Steruje tym wszystkim za pomocą niewielu sygnałów sterujących kierowanych do pewnych punktów w procesorze oraz niewielu sygnałów kierowanych do magistrali systemowej.

Wewnętrzna organizacja procesora

Na rysunku 14.5 jest zilustrowane zastosowanie wielu różnych ścieżek danych. Złożoność tego typu organizacji jest oczywista. Częściej używana jest pewna odmiana wewnętrznej magistrali, co już pokazaliśmy na rys. 11.2.

Jeśli zastosuje się wewnętrzną magistralę procesora, to rys. 14.5 można przekształcić, otrzymując rys. 14.6. ALU i wszystkie rejestry procesora są połączone za pomocą jednej magistrali wewnętrznej. Przewidziane zostały bramki i sygnały sterujące, które kontrolują ruch danych między każdym rejestrem a magistralą. Dodatkowe sygnały sterujące kontrolują przesyłanie danych do (i z) magistrali systemowej (zewnętrznej) oraz działanie ALU.

Do tej struktury dodano dwa nowe rejestry, oznaczone Y i Z. Są one potrzebne do właściwego działania ALU. Gdy wykonywana jest operacja wymagająca dwóch argumentów, jeden z nich może być uzyskany z magistrali wewnętrznej, jednak drugi musi być uzyskany z innego źródła. Do tego celu mógłby być użyty akumulator (AC), jednak ograniczałoby to elastyczność systemu i nie działałoby w przypadku procesora z wieloma rejestrami roboczymi. Rejestr Y służy do czasowego przechowywania drugiego argumentu. ALU jest układem kombinacyjnym (patrz dodatek do książki) bez wewnętrznej pamięci. Wobec tego, gdy sygnały sterujące aktywują pewną funkcję ALU, dane wejściowe ALU są przekształcane i podawane na wyjście. Wyjście ALU nie może więc być bezpośrednio dołączone do magistrali, ponieważ istniałoby wówczas sprzężenie zwrotne wyjścia z wejściem. Rejestr Z umożliwia czasowe przechowywanie wartości wyjściowych. Przy takiej organizacji operacja dodawania wartości z pamięci do AC przebiegałaby w następujących krokach:

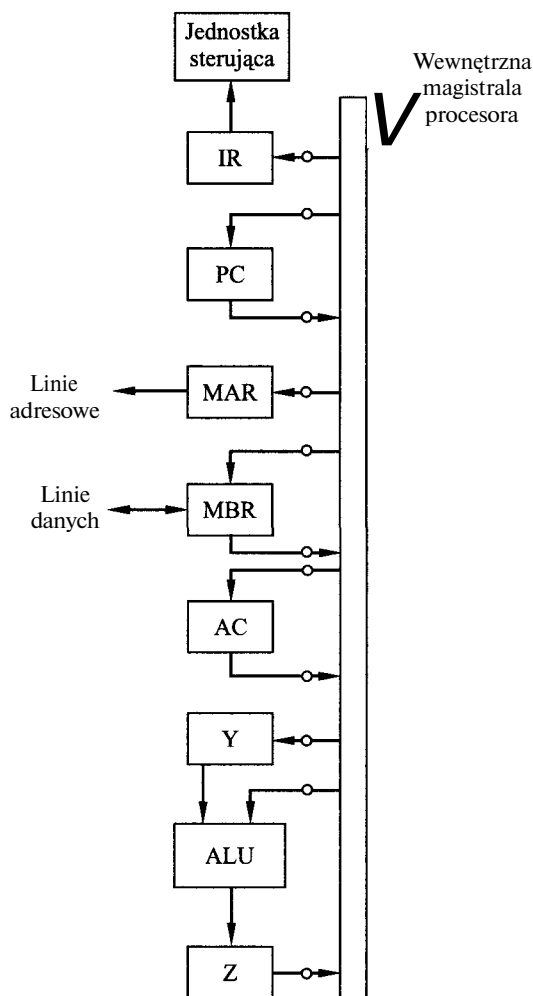
$t_1 : \text{MAR} \leftarrow (\text{IR}(\text{Adres}))$

$t_2 : \text{MBR} \leftarrow \text{Pamięć}$

$t_3 : Y \leftarrow (\text{MBR})$

$t_4 : Z \leftarrow (\text{AC}) + (Y)$

$t_5 : \text{AC} \leftarrow (Z)$

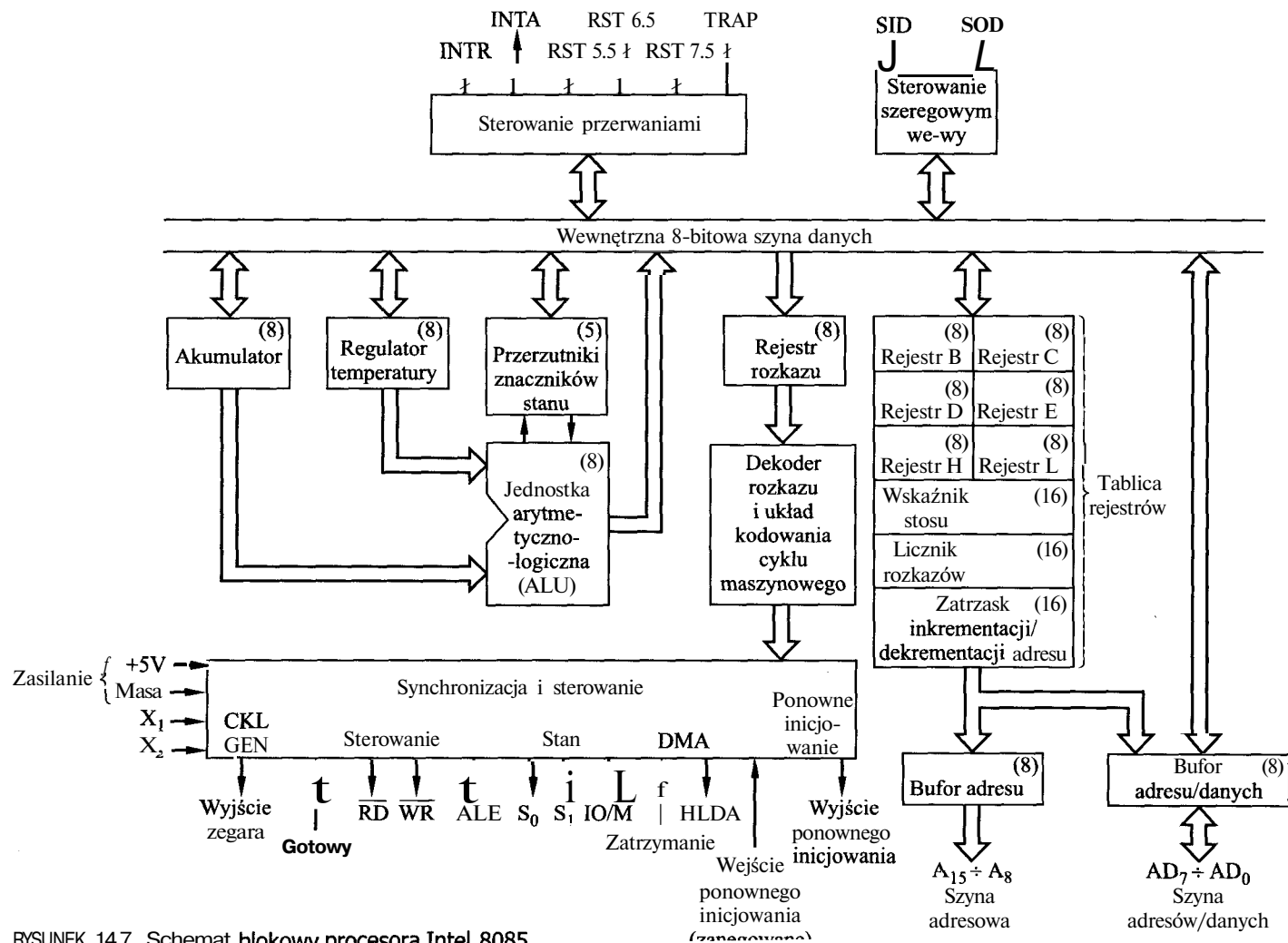


RYSUNEK 14.6. Procesor z magistralą wewnętrzną

Możliwe są inne organizacje, jednak na ogół jest używany pewien rodzaj magistrali wewnętrznej lub zespół magistrali wewnętrznych. Użycie wspólnej ścieżki danych upraszcza geometrię połączeń i sterowanie procesorem. Innym praktycznym powodem stosowania wewnętrznej magistrali jest oszczędność miejsca. Zwłaszcza w przypadku mikroprocesorów, które mogą zajmować kawałek krzemu o powierzchni zaledwie 1/4 cala kwadratowego, przestrzeń zajmowana przez połączenia międzyrejestrowe musi być minimalizowana.

Procesor Intel 8085

W celu zilustrowania koncepcji wprowadzonych dotychczas w tym rozdziale, rozważmy procesor Intel 8085. Jego organizację widać na rys. 14.7. Do kluczowych składników, których rola może nie być oczywista, należą:



RYSUNEK 14.7. Schemat blokowy procesora Intel 8085

- a **Przerzutnik zatraskowy inkrementacji/dekrementacji adresu.** Układ logiczny, który może dodawać lub odejmować 1 od stanu wskaźnika stosu lub licznika programu. Zapobiega to używaniu do tego celu ALU, co oszczędza czas.
- **Układ sterowania przerw.** Moduł ten przetwarza wiele poziomów sygnałów przerw.
- a **Układ sterowania szeregowym wejściem-wyjściem.** Moduł ten stanowi interfejs z przyrządami, które przekazują dane bit po bicie.

W tabeli 14.2 są opisane sygnały zewnętrzne kierowane do (i z) procesora 8085. Są one wprowadzane do zewnętrznej magistrali systemowej. Stanowią one interfejs między procesorem 8085 a resztą systemu (rys. 14.8).

TABELA 14.2. Sygnały zewnętrzne procesora Intel 8085

<i>Sygnały adresów i danych</i>	
Najbardziej znaczący adres ($A_{15} + A_8$)	Najbardziej znaczące 8 bitów 16-bitowego adresu
Adres/Dane ($AD_7 + AD_0$)	Najmniej znaczące 8 bitów 16-bitowego adresu lub 8 bitów danych. Multipleksowanie to umożliwia ograniczenie liczby końcówek.
Szeregowe dane wejściowe (SID)	Jednobitowe wejście dostosowane do przyrządów transmitujących szeregowo (po jednym bicie).
Szeregowe dane wyjściowe (SOD)	Jednobitowe wyjście dostosowane do przyrządów odbierających szeregowo.
<i>Sygnały taktujące i sterujące</i>	
CLK (OUT)	Zegar systemowy. Każdy cykl reprezentuje jeden stan T. Sygnał CLK jest kierowany do mikroukładów peryferyjnych i synchronizuje je.
X_1, X_2	Sygnały te pochodzą z zewnętrznego rezonatora lub innego przyrządu napędzającego generator zegara wewnętrznego.
Zezwolenie zatrasku adresu (ALE)	Następuje podczas pierwszego stanu zegara cyklu maszynowego i powoduje zapisanie linii adresowych przez mikroukłady peryferyjne. Pozwala to modułowi adresowanemu (np. pamięci, wejściu-wyjściu) na rozpoznanie, że jest adresowany.
Stan (S_0, S_1)	Sygnały sterujące używane do wskazywania, czy ma miejsce operacja odczytu, czy też zapisu.
I/O/M	Używany do zezwolenia albo modułowi wejścia-wyjścia, albo pamięci na operacje odczytu i zapisu.
Sterowanie odczytu (RD)	Wskazuje, że wybrany moduł wejścia-wyjścia lub pamięci ma być odczytywany i że szyna danych jest dostępna do przesyłania danych.
Sterowanie zapisu (WR)	Wskazuje, że dane na szynie danych mają być zapisane w wybranej lokacji pamięci lub wejścia-wyjścia.

TABELA 14.2 cd.

*Sygnały inicjowane przez pamięć i wejście-wyjście***Zatrzymanie (hold)**

Wymaga od procesora zwolnienia sterowania i używania zewnętrznej magistrali systemowej. Procesor zakończy wykonywanie rozkazu znajdującego się w rejestrze **IR**, po czym przejdzie do stanu zatrzymania, podczas którego nie umieszcza on żadnych sygnałów na szynach sterowania, adresu i danych. Podczas stanu zatrzymania magistrala może być używana do operacji DMA.

Potwierdzenie zatrzymania (HOLDA)

Sygnał wyjściowy jednostki sterującej, potwierdzający sygnał **HOLD** i wskazujący, że magistrala jest dostępna.

READY

Używany do synchronizowania procesora z wolniejszymi przyrządami pamięci i wejścia-wyjścia. Gdy adresowany przyrząd potwierdza **READY**, procesor może kontynuować operację wejściową (**DBIN**) lub wyjściową (**WR**). W przeciwnym przypadku wchodzi on w stan oczekiwania, aż przyrząd będzie gotowy.

*Sygnały związane z przerwaniem***TRAP**

Wznawia przerwanie (**RST 7.5, 6.5, 5.5**).

Zapotrzebowanie przerwania (INTR)

Pięć linii używanych przez przyrząd zewnętrzny do przerwania procesora. Procesor nie honoruje żądań, jeśli znajduje się w stanie zatrzymania lub jeśli przerwanie jest zablokowane. Przerwanie jest honorowane tylko po zakończeniu rozkazu. Przerwania następują w kolejności według malejącego priorytetu.

Potwierdzenie przerwania

Potwierdza przerwanie.

*Inicjowanie procesora***RESET IN**

Powoduje, że zawartość licznika PC jest zerowana. Procesor wznawia wykonywanie od lokacji 0.

RESET OUT

Potwierdza, że procesor został ponownie zainicjowany. Sygnał ten może być używany do ponownego inicjowania reszty systemu.

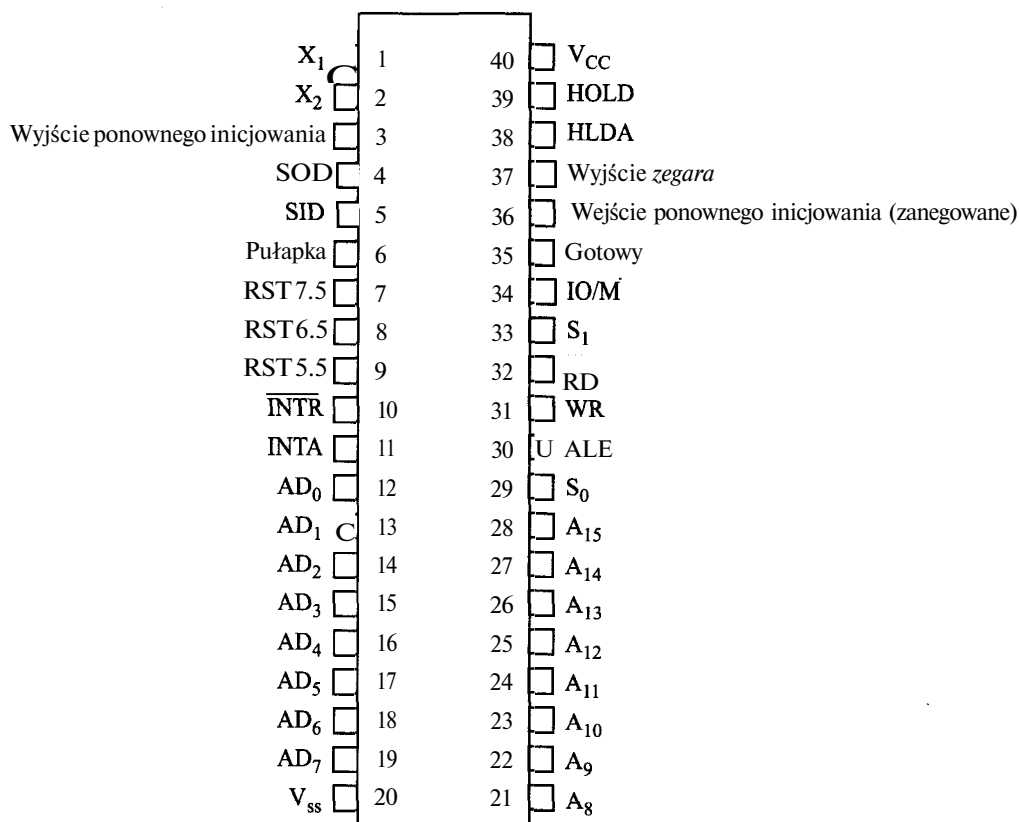
*Zasilanie***V_{cc}**

Zasilanie +5 V.

V_{ss}

Masa (0 V).

Pokazana jednostka sterująca składa się z dwóch modułów: (1) dekodera rozkazów i układu kodowania cyklu maszynowego oraz (2) układu taktowania i sterowania. Dyskusję na temat pierwszego modułu odłożymy do następnego podrozdziału. Jądem jednostki sterującej jest moduł taktowania i sterowania. Zawiera on zegar; na jego wejście są podawane: bieżący rozkaz i pewne zewnętrzne sygnały sterujące. Na wyjściu pojawiają się sygnały sterujące kierowane do innych składników CPU oraz sygnały sterujące kierowane do zewnętrznej magistrali systemowej.

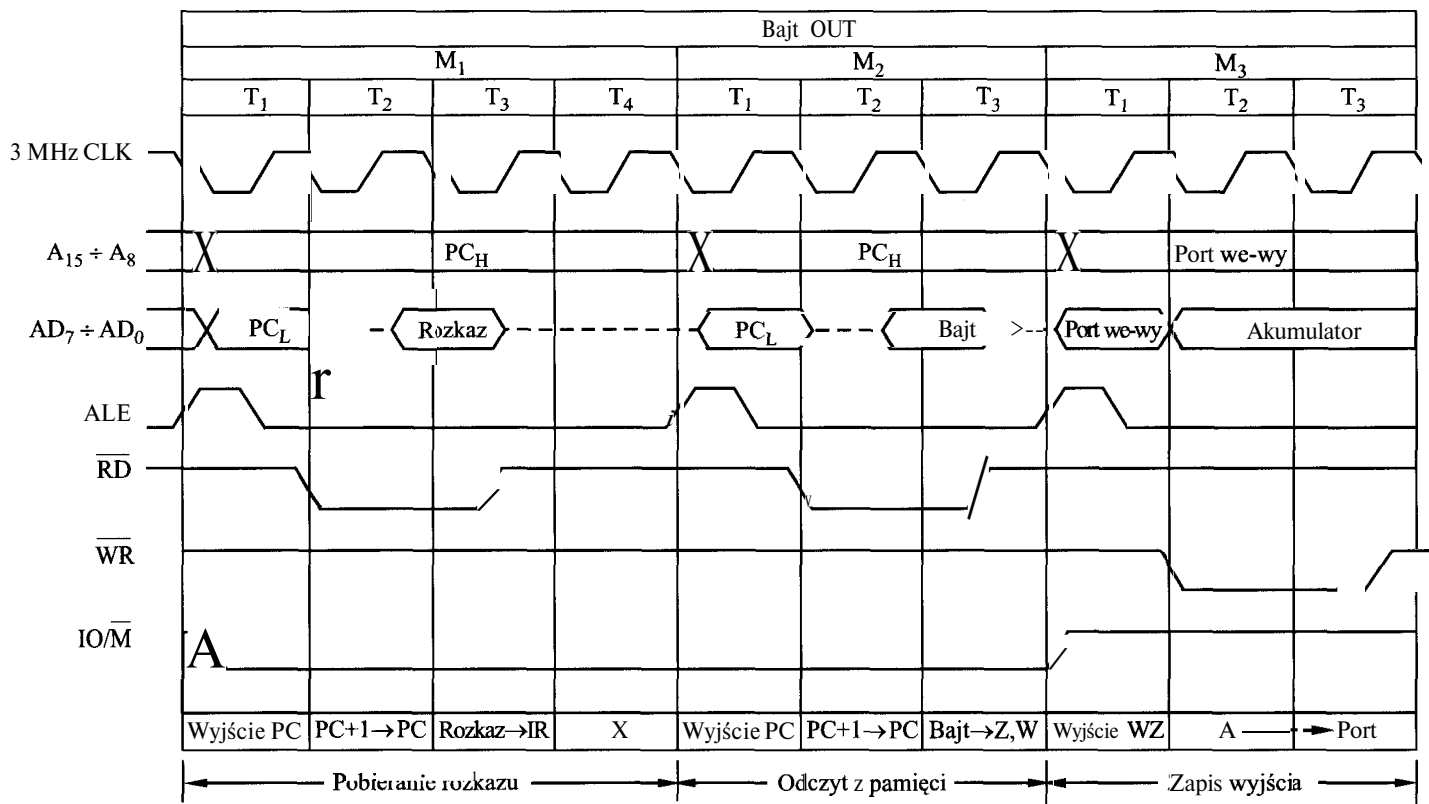


RYSUNEK 14.8. Rozkład wyprowadzeń procesora Intel 8085

Taktowanie operacji procesora jest synchronizowane za pomocą zegara i sterowane przez jednostkę sterującą za pomocą sygnałów sterujących. Każdy cykl rozkazu jest dzielony na 1 do 5 cykli maszynowych. Każdy cykl maszynowy jest z kolei dzielony na 3 do 5 stanów. Każdy stan trwa jeden cykl zegara. Podczas jednego stanu procesor wykonuje **jedną mikrooperację** lub zbiór jednoczesnych **mikrooperacji**, zgodnie z sygnałami sterującymi.

Liczba cykli maszynowych jest stała dla danego rozkazu, jednak zmienia się od rozkazu do rozkazu. Cykle maszynowe są definiowane jako równoważne dostępom do magistrali. Wobec tego liczba cykli maszynowych dla określonego rozkazu zależy od tego, ile razy procesor musi się komunikować z urządzeniami zewnętrznymi. Jeśli na przykład rozkaz składa się z dwóch porcji 8-bitowych, to do pobrania rozkazu są wymagane dwa cykle maszynowe. Jeśli rozkaz obejmuje 1-bajtową operację dostępu do pamięci lub wejścia-wyjścia, to do jego wykonania jest potrzebny trzeci cykl maszynowy.

Na rysunku 14.9 widać przykład taktowania procesora 8085, z uwzględnieniem wartości zewnętrznych sygnałów sterujących. Oczywiście, w tym samym czasie przez jednostkę sterującą są również generowane wewnętrzne sygnały



RYSUNEK 14.9. Przebiegi czasowe rozkazu OUT procesora Intel 8085

sterujące w celu kontrolowania wewnętrznych transferów danych. Na wykresie jest pokazany cykl rozkazu OUT. Potrzebne są trzy cykle maszynowe (M_1 , M_2 , M_3). Podczas pierwszego z nich pobierany jest rozkaz OUT. Podczas drugiego cyklu maszynowego pobierana jest druga połowa rozkazu zawierająca numer urządzenia wejścia-wyjścia wybranego jako wyjściowe. Podczas trzeciego cyklu zawartość AC jest kierowana do wybranego urządzenia poprzez magistralę danych.

Początek każdego cyklu maszynowego jest sygnalizowany przez impuls zezwolenia przerzutnika zatraskowego adresu (ang. *Address Latch Enabled* - ALE) pochodzący z jednostki sterującej. Impuls ALE wywołuje stan gotowości układów zewnętrznych. Podczas stanu T_1 cyklu maszynowego M_1 jednostka sterująca ustawia sygnał IO/M w celu pokazania, że jest to operacja pamięci. Jednostka sterująca powoduje również, że zawartość licznika PC jest umieszczana na szynie adresowej ($A_{15}+A_8$) oraz na szynie adresów/danych (AD_7+AD_0). Podczas opadającego zbocza impulsu ALE pozostałe moduły na magistrali zapisują adres.

Podczas stanu T_2 zaadresowany moduł pamięci umieszcza zawartość zaadresowanej lokacji na szynie adresów/danych. Jednostka sterująca ustawia sygnał sterowania odczytem (ang. *Read Control* - RD) w celu zasygnalizowania odczytu, jednak skopiowanie danych z szyny jest opóźnione do T_3 . Daje to modułowi pamięci czas na umieszczenie danych na magistrali oraz na ustabilizowanie poziomów sygnałów. Ostatni stan, T_4 , jest stanem *nieaktywnej szyny*, podczas którego procesor dekoduje rozkaz. Pozostałe cykle maszynowe przebiegają podobnie.

14.3

ROZWIĄZANIE UKŁADOWE

Analizowaliśmy jednostkę sterującą, rozważając jej wejścia, wyjścia i funkcje. Przyszedł czas na zajęcie się problemem wdrożenia jednostki sterującej. Używano do tego wielu różnorodnych metod. Większość z nich można przyporządkować do dwóch kategorii:

- a wdrożenie układowe,
- a wdrożenie mikroprogramowane.

W przypadku rozwiązania układowego jednostka sterująca jest w zasadzie układem kombinacyjnym. Jej wejściowe sygnały logiczne są transformowane na zbiór wyjściowych sygnałów logicznych, które są sygnałami sterującymi. To właśnie rozwiązanie opiszemy w tym podrozdziale. Realizację w postaci mikroprogramu omówimy w rozdziale 15.

Wejścia jednostki sterującej

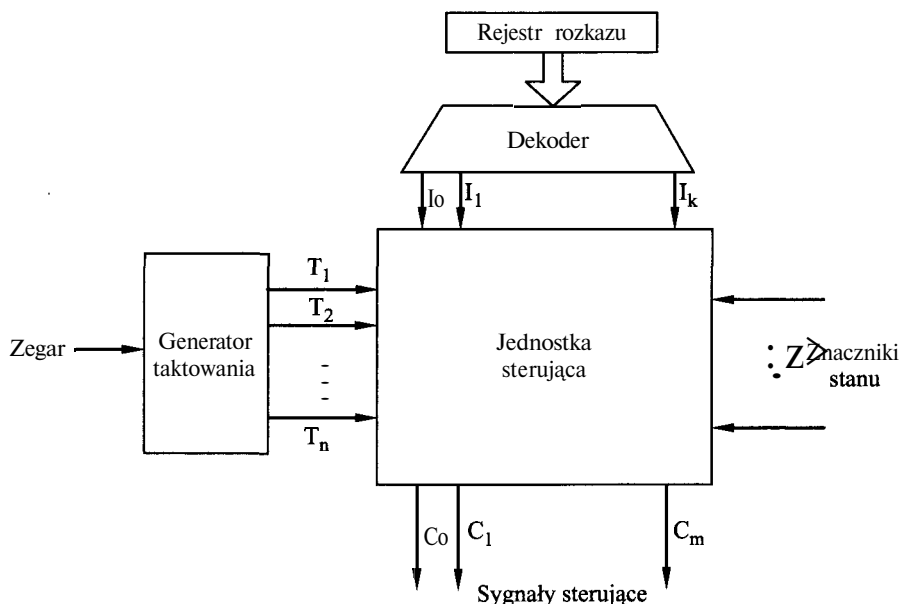
Na rysunku 14.4 jest pokazana jednostka sterująca w ujęciu takim, jakie dotąd omawialiśmy. Najważniejsze dane wejściowe pochodzą z rejestru rozkazów, zegara, znaczników stanu oraz linii sterujących magistrali. W przypadku znaczników stanu i sygnałów sterujących magistrali każdy pojedynczy bit ma zwykle pewne określone znaczenie (np. sygnalizuje przepełnienie). Pozostałe dwa wejścia nie są jednak bezpośrednio użyteczne dla jednostki sterującej.

Rozważmy najpierw rejestr rozkazów. Jednostka sterująca rozpoznaje kod operacji i wykonuje różne działania (generuje różne kombinacje sygnałów sterujących) w przypadku różnych rozkazów. W celu uproszczenia układów logicznych jednostki sterującej powinny istnieć unikatowe wejściowe sygnały logiczne odpowiadające każdemu kodowi operacji. Zapewnia to dekodery, który przyjmuje zakodowane sygnały wejściowe i dostarcza pojedynczy sygnał wyjściowy. Dekoder ma n wejść binarnych i 2^n wyjść binarnych. Każda z 2^n różnych kombinacji sygnałów wejściowych aktywuje unikatowy sygnał wyjściowy. Przykład jest pokazany w tabeli 14.3. Dekoder jednostki sterującej jest zwykle bardziej złożony ze względu na kody operacji o różnych długościach. Przykład cyfrowych układów logicznych używanych przy realizacji dekodera jest przedstawiony w dodatku A.

TABELA 14.3. Dekoder o czterech wejściach i szesnastu wyjściach

I ₁	I ₂	I ₃	I ₄	O ₁	O ₂	O ₃	O ₄	O ₅	O ₆	O ₇	O ₈	O ₉	O ₁₀	O ₁₁	O ₁₂	O ₁₃	O ₁₄	O ₁₅	O ₁₆
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Część zegarowa jednostki sterującej wysyła powtarzalną sekwencję impulsów. Jest to przydatne do mierzenia czasu trwania mikrooperacji. Konieczne jest, aby okres impulsów zegarowych był dostatecznie długi, aby umożliwić propagację sygnałów w ścieżkach danych i w układach procesora. Jak jednak widzieliśmy, jednostka sterująca wysyła różne sygnały sterujące w różnych jednostkach czasowych w czasie trwania pojedynczego cyklu rozkazu. Wobec tego



RYSUNEK 14.10. Jednostka sterująca ze zdekodowanymi wejściami

na wejściu jednostki sterującej potrzebny jest licznik dostarczający różnych sygnałów kontrolnych odpowiadających T_1 , T_2 itd. Na końcu cyklu rozkazu* jednostka sterująca musi uzyskać sygnał zwrotny z licznika w celu zainicjowania się w jednostce czasu T_1 .

Jednostka sterująca zawierająca oba omówione udoskonalenia została przedstawiona na rys. 14.10.

Rozwiązania logiczne jednostki sterującej

Do zdefiniowania sprzętowego rozwiązania jednostki sterującej pozostaje jedynie omówić wewnętrzne układy logiczne tej jednostki, które tworzą wyjściowe sygnały sterujące będące funkcją sygnałów wejściowych.

W tym celu należy obliczyć wyrażenie Boole'a dla każdego sygnału sterującego jako funkcję sygnałów wejściowych. Najlepiej wyjaśnić to na przykładzie. Rozważmy ponownie nasz prosty przykład zilustrowany na rys. 14.5. W tabeli 14.1 zapoznaliśmy się z sekwencjami **mikrooperacji** i sygnałami sterującymi niezbędnymi do sterowania trzema spośród czterech faz cyklu rozkazu.

Rozważmy pojedynczy sygnał sterujący C_5 . Sygnał ten **powoduje** wczytanie danych z zewnętrznej magistrali danych do rejestru MBR. Widzimy, że w tabeli 14.1 został on użyty dwukrotnie. Zdefiniujemy dwa nowe sygnały sterujące, P i Q, interpretowane następująco:

* W rzeczywistości inicjowanie musi następować na zakończenie każdego podcyklu (pobierania, wykonywania, adresowania pośredniego i przerwania) (*przyp.tlum.*).

PQ = 00	cykl pobierania
PQ = 01	cykl adresowania pośredniego
PQ = 10	cykl wykonywania
PQ = 11	cykl przerywania

Wobec tego następujące wyrażenie Boole'a definiuje C_5 :

$$C_5 = \bar{P} \cdot \bar{Q} \cdot T_2 + \bar{P} \cdot Q \cdot T_2$$

Oznacza to, że sygnał C_5 będzie potwierdzony podczas drugiej jednostki czasowej cyklu pobierania i cyklu adresowania pośredniego.

Wyrażenie to nie jest kompletne. Sygnał C_5 jest również potrzebny podczas cyklu wykonywania. W naszym prostym przykładzie załóżmy, że istnieją tylko trzy rozkazy odczytywane z pamięci: LDA, ADD i AND. Możemy teraz zdefiniować C_5 jako:

$$C_5 = \bar{P} \cdot \bar{Q} \cdot T_2 + \bar{P} \cdot Q \cdot T_2 + P \cdot \bar{Q} \cdot (LDA + ADD + AND) \cdot T_2$$

Podobny proces mógłby być powtórzony dla każdego sygnału sterującego generowanego przez procesor. Wynikiem byłby zbiór równań Boole'a definiujący zachowanie jednostki sterującej i tym samym procesora.

Aby to wszystko powiązać, jednostka sterująca musi kontrolować stan cyklu rozkazu. Jak już wspomnieliśmy, na zakończenie każdego podcyklu (pobierania, adresowania pośredniego, wykonywania, przerywania) jednostka sterująca wysyła sygnał, który powoduje inicjowanie generatora taktowania i podanie sygnału TV. Jednostka sterująca musi także ustawić odpowiednie wartości P i Q celu określenia następnego podcyklu.

Zauważmy, że w przypadku nowoczesnego, złożonego procesora liczba równań Boole'a wymagana do zdefiniowania jednostki sterującej jest bardzo duża. Zadanie wdrożenia układu kombinacyjnego spełniającego te wszystkie równania staje się ekstremalnie trudne. W rezultacie używa się zwykle daleko prostszego rozwiązania, znanego jako mikroprogramowanie. Jest ono przedmiotem następnego rozdziału.

POLECANA LITERATURA

W wielu książkach zawarto podstawy działania jednostki sterującej, w tym w [HENN91], [WARD90], [HAYE88] i [MANO93].

HAYE88 Hayes J.: *Computer Architecture and Organization*. Wyd. II. New York, McGraw-Hill, 1988.

HENN91 Hennessy J., Jouppi N.: *Computer Technology and Architecture: An Evolving Interaction*. *Computer*, September 1991.

MANO93 Mano M.: *Computer System Architecture*. Englewood Cliffs, Prentice Hall, 1993.

WARD90 Ward S., Halstead R.: *Computation Structures*. Cambridge, MIT Press, 1990.

PROBLEMY DO ROZWIĄZANIA

- 14.1. Twoja ALU jest w stanie dodawać zawartość swoich dwóch rejestrów wejściowych i może tworzyć logiczne uzupełnienia bitów każdego z rejestrów, nie może jednak odejmować. Liczby są przechowywane w postaci reprezentacji uzupełnienia do dwóch. Wymień mikrooperacje, jakie musi spowodować jednostka sterująca w celu wykonania odejmowania.
- 14.2. Pokaż mikrooperacje i sygnały sterujące w ujęciu takim, jak w tabeli 14.1 dla CPU z rys. 14.5 i dla następujących rozkazów:
- ☐ ładuj akumulator,
 - Q zapisz zawartość akumulatora,
 - ☐ dodaj do akumulatora,
 - ☐ I do akumulatora,
 - ☐ skocz,
 - Q skocz, jeśli $AC = 0$,
 - a dopełnij akumulator.
- 14.3. Załóż, że opóźnienie propagacji w magistrali i w ALU przedstawionej na rys. 14.6 wynosi odpowiednio 20 i 100 ns. Czas wymagany do skopiowania przez rejestr danych z magistrali wynosi 10 ns. Ile czasu musi być przewidziane na:
- (a) przeniesienie danych z rejestru do rejestru?
 - (b) inkrementowanie licznika programu?
- 14.4. Napisz sekwencję mikrooperacji wymaganą w przypadku struktury magistralowej z rys. 14.6 dla dodania liczby do AC, jeśli liczba ta jest:
- (a) argumentem natychmiastowym;
 - (b) argumentem o adresie pośrednim;
 - (c) argumentem o adresie bezpośrednim.
- 14.5. Sporządź wykres podobny do rys. 14.9 dla następujących rozkazów procesora 8085:
- (a) MOV reg1, reg2 $reg1 \leftarrow (reg2)$
 - (b) MOV M, reg $M \leftarrow (reg)$
- 14.6. Zasugeruj wewnętrzną ścieżkę danych i organizację sygnałów sterujących dla tablicy rejestrów procesora Intel 8085.
- 14.7. Stos jest wdrożony w postaci przedstawionej na rys. 9.14. Pokaż sekwencję mikrooperacji dla operacji:
- (a) zdejmowania ze stosu;
 - (b) umieszczania na stosie.

Sterowanie za pomocą mikroprogramu

Termin **mikroprogram** został po raz pierwszy zaproponowany przez M.V. Wilkesa we wczesnych latach pięćdziesiątych. Wilkes zaproponował pewien rodzaj uporządkowanego i systematycznego podejścia do projektowania jednostki sterującej, w którym unika się złożoności charakterystycznej dla wdrożenia układowego. Idea ta zaintrygowała wielu badaczy, jednak wydawała się niemożliwa do zrealizowania, ponieważ wymagałaby szybkiej i stosunkowo taniej pamięci sterującej.

Stan sztuki **mikroprogramowania** przedstawiono w lutowym wydaniu *Data-mation* z roku 1964. Żaden system **mikroprogramowany** nie był w tym czasie szeroko stosowany, a w jednym z artykułów [HILL64] następująco podsumowano popularny pogląd na przyszłość mikroprogramowania: „Jest ona nieco mglista. Żaden z dużych producentów nie wykazał zainteresowania tą **metodą**, chociaż najprawdopodobniej wszyscy ją **zbadali**.”

Sytuacja ta zmieniła się drastycznie w ciągu niewielu miesięcy. W kwietniu zaanonsowano **System/360 IBM**, w którym prawie wszystkie największe modele były **mikroprogramowane**. Chociaż seria 360 wyprzedziła dostępność półprzewodnikowych pamięci ROM, zalety mikroprogramowania okazały się na tyle istotne, że firma IBM zdecydowała się na ten krok. Od tego czasu popularność mikroprogramowania w wielu zastosowaniach wzrastała, przy czym jednym z nich było **mikroprogramowe** rozwiązanie jednostki sterującej procesora. To właśnie zastosowanie przeanalizujemy w tym rozdziale.

15.1

KONCEPCJE PODSTAWOWE**Mikrorozkazy**

W ujęciu dotychczas opisanym jednostka sterująca wydaje się być urządzeniem rozsądnie prostym. Jednak realizacja jednostki sterującej w postaci połączenia podstawowych elementów logicznych jest zadaniem niełatwym. Projekt musi uwzględniać rozwiązania logiczne służące do porządkowania mikrooperacji, do ich wykonywania, do interpretowania kodów operacji i do podejmowania decyzji na podstawie znaczników stanu ALU. Trudno jest zaprojektować i przetestować takie urządzenie. Ponadto rozwiązanie takie jest stosunkowo nieelastyczne. Trudno jest na przykład zmienić projekt, jeśli ktoś życzy sobie dodania nowego rozkazu maszynowego.

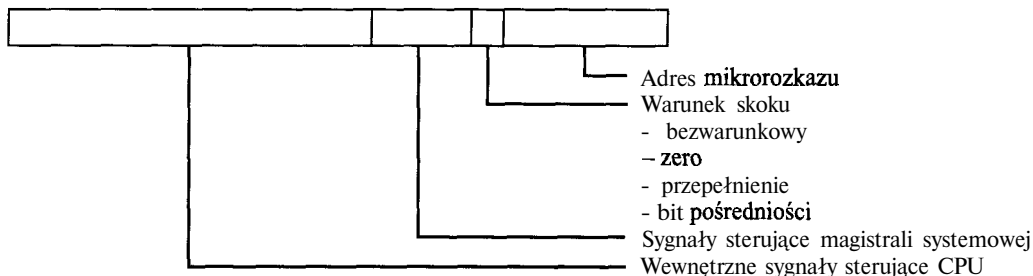
Istnieje jednak rozwiązanie alternatywne, całkiem powszechne w produkowanych obecnie komputerach, a mianowicie implementacja mikroprogramowanej jednostki sterującej.

Rozważmy ponownie tabelę 14.1. Obok wymienionych sygnałów sterujących każda mikrooperacja została opisana w notacji symbolicznej. Zauważmy, że notacja ta wygląda całkiem jak język programowania! W istocie to jest język, nazywa się go **językiem mikroprogramowania**. Każdy wiersz opisuje zbiór jednocześnie następujących mikrooperacji i jest określany jako **mikrorozkaz**. Sekwencja rozkazów jest nazywana **mikroprogramem lub oprogramowaniem układowym** (ang. *firmware*). Ten ostatni termin odzwierciedla to, że mikroprogram znajduje się pośrodku między sprzętem a oprogramowaniem. Łatwiej jest zaprojektować oprogramowanie układowe niż sprzęt, jednak trudniej jest napisać program należący do oprogramowania układowego niż do zwykłego oprogramowania.

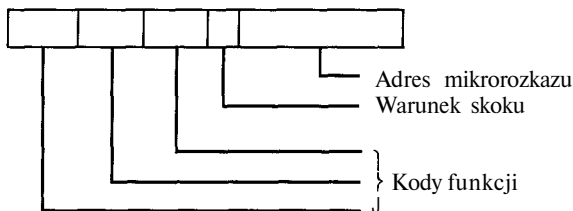
Jak można zastosować koncepcję mikroprogramowania do wdrożenia jednostki sterującej? Zauważmy, że dla każdej mikrooperacji wszystkim, co wykonuje jednostka sterująca, jest generowanie zbioru sygnałów sterujących. Wobec tego dla dowolnej mikrooperacji każda linia sterowania wyprowadzona z jednostki sterującej jest albo w stanie włączenia, albo wyłączenia. Stan ten może być oczywiście reprezentowany przez cyfrę binarną związaną z każdą linią sterowania. Moglibyśmy więc utworzyć słowo sterujące, w którym każdy bit reprezentuje jedną linię sterowania. Wobec tego każda mikrooperacja byłaby reprezentowana przez różny układ jedynek i zer w słowie sterującym.

Zaczyna to wyglądać obiecująco. Załóżmy, że połączymy w łańcuch sekwencję słów sterujących w celu reprezentowania sekwencji mikrooperacji wykonywanych przez jednostkę sterującą. Jesteśmy już prawie „w domu”. Musimy następnie uprzytomnić sobie, że sekwencja mikrooperacji nie jest ustalona. Czasem występuje cykl adresowania pośredniego, a czasem nie. Zapiszmy więc nasze słowa sterujące w pamięci, przy czym każde słowo niech ma unikatowy ad-

(a)



(b)



RYСУNEK 15.1. Typowe formaty mikrorozkazów: (a) mikrorozkaz poziomy; (b) mikrorozkaz pionowy

res. Dodajmy teraz pole adresowe do każdego słowa sterującego, wskazując w ten sposób lokację następnego słowa przewidzianego do wykonania, jeśli będzie spełniony pewien warunek (np. gdy bit adresowania pośredniego w rozkazie odnoszącym się do pamięci jest równy 1). Dodajmy również kilka bitów precyzujących warunek.

Rezultat jest określany jako **mikrorozkaz poziomy** (rys. 15.1a). Format mikrorozkazu lub słowa sterującego jest następujący. Występuje w nim po jednym bicie na każdą wewnętrzną linię sterowania procesora i po jednym bitem na każdą linię sterowania magistrali systemowej. Występuje też pole warunku wskazujące warunek rozgałęzienia oraz pole adresu następnego mikrorozkazu, który ma być wykonywany, jeśli nastąpi rozgałęzienie.

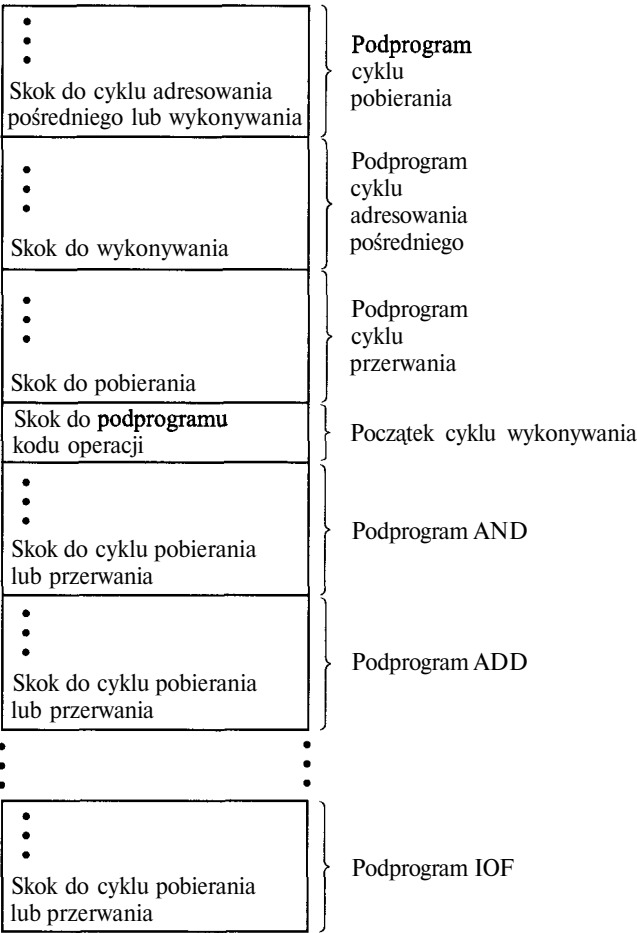
Taki mikrorozkaz jest interpretowany następująco:

1. W celu wykonania tego mikrorozkazu należy włączyć wszystkie linie sterowania wskazane przez bity 1 i pozostawić wyłączone wszystkie linie sterowania wskazane przez 0. Wynikające stąd sygnały sterujące spowodują wykonanie jednej lub wielu **mikrooperacji**.
2. Jeśli warunek wskazany przez bity warunku nie jest spełniony, to należy wykonać następny mikrorozkaz w sekwencji.
3. Jeśli warunek wskazany przez bity warunku jest spełniony, to następny mikrorozkaz przewidziany do wykonania jest wskazany w polu adresowym.

Na rysunku 15.2 widać, w jaki sposób te słowa sterujące lub **mikrorozkazy** mogą być zorganizowane w postaci **pamięci sterującej**. Mikrorozkazy w każ-

dym programie standardowym mają być realizowane po kolei. Każdy program standardowy kończy się rozkazem rozgałęzienia lub skoku wskazującym, do jakiego punktu należy przejść. Istnieje specjalny program standardowy cyklu wykonywania, którego jedynym celem jest zaznaczenie tego programu związanego z rozkazem maszynowym (AND, ADD itd.), który ma być wykonywany jako następny, zależnie od kodu operacji.

Na rysunku 15.2 jest przedstawiona pamięć sterująca. Rysunek ten jest jednocześnie zwięzłym opisem funkcjonowania jednostki sterującej. W pamięci jest określana **sekwencja mikrooperacji**, która ma być wykonywana podczas każdego cyklu (pobierania, adresowania pośredniego, wykonywania, przerwania), jest także określony porządek tych cykli. Jeśli pamięć ta nie byłaby niczym więcej, ta notacja byłaby pożyteczna do dokumentowania działania jednostki sterującej określonego komputera. Jest ona jednak czymś więcej. Jest także sposobem wdrażania jednostki sterującej.



RYSUNEK 15.2. Organizacja pamięci sterującej

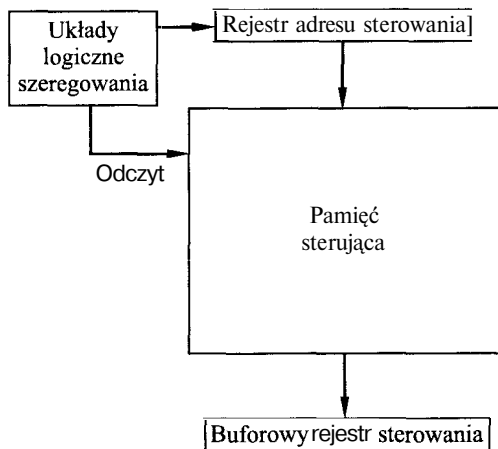
Mikroprogramowana jednostka sterująca

Pamięć sterująca pokazana na rys. 15.2 zawiera program opisujący zachowanie jednostki sterującej. Wynika stąd, że moglibyśmy wdrożyć jednostkę sterującą po prostu wykonując ten program.

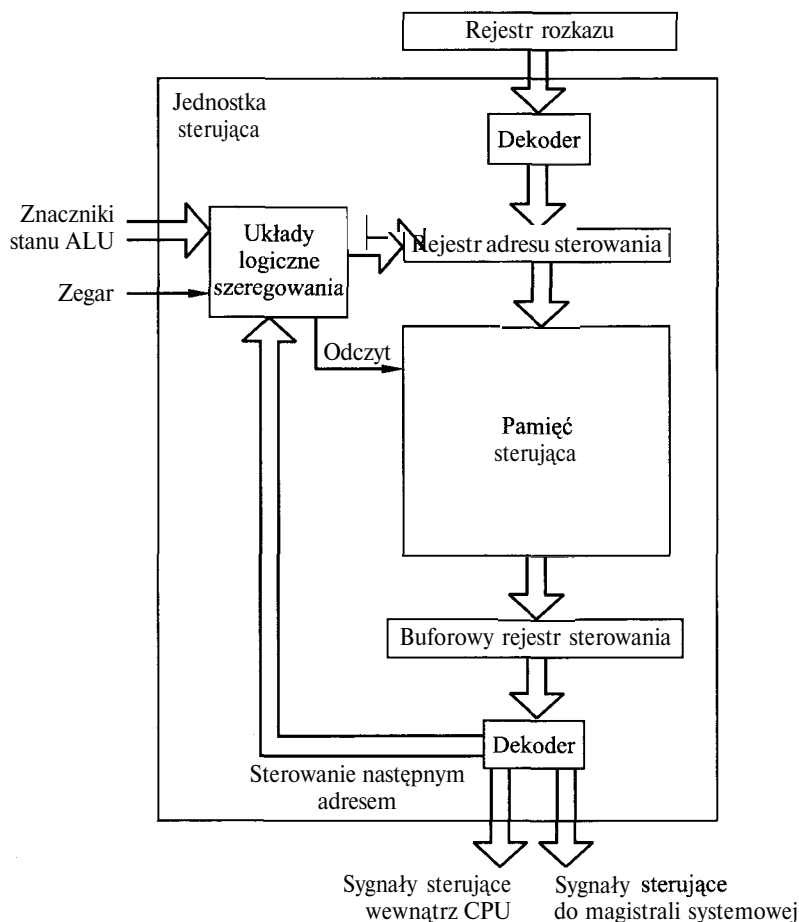
Na rysunku 15.3 są pokazane podstawowe elementy takiej implementacji. Zbiór mikrorozkazów jest przechowywany w *pamięci sterującej*. **Rejestr adresu sterowania** zawiera adres następnego mikrorozkazu, który ma być odczytany. Po odczytaniu mikrorozkazu z pamięci sterującej jest on przenoszony do **buforowego rejestru sterowania**. Lewa część tego rejestru (patrz rys. 15.1a) jest połączona z liniami sterowania wychodzącymi z jednostki sterującej. *Odczytanie* mikrorozkazu z pamięci sterującej jest więc tym samym, co *wykonanie* tego mikrorozkazu! Trzecim elementem pokazanym na rysunku jest jednostka szeregująca, która ładuje rejestr adresu sterowania i wydaje rozkaz odczytu.

Przeanalizujmy tę strukturę bardziej szczegółowo, wykorzystując do tego celu rys. 15.4. Porównując ją z rys. 14.4 widzimy, że jednostka sterująca ma nadać takie same wejścia (IR, znaczniki stanu ALU, zegar) i wyjścia (sygnały sterujące). Jednostka sterująca działa następująco:

1. W celu wykonania rozkazu szeregująca jednostka logiczna wydaje rozkaz READ (odczyt) odnoszący się do pamięci sterującej.
2. Słowo, którego adres jest określony w rejestrze adresu sterowania, jest wczytywane do buforowego rejestru sterowania.
3. Zawartość buforowego rejestru sterowania generuje sygnały sterujące oraz informację o następnym adresie skierowaną do logicznej jednostki szeregowania.
4. Logiczna jednostka szeregowania ładuje nowy adres do rejestru adresu sterowania na podstawie informacji o następnym adresie uzyskanej z buforowego rejestru sterowania i na podstawie znaczników stanu ALU.



RYСУNEK 15.3. Mikroarchitektura jednostki sterującej



RYSUNEK 15.4. Funkcjonowanie mikroprogramowanej jednostki sterującej

Wszystko to następuje podczas jednego cyklu zegara.

Ostatni z wymienionych kroków wymaga szczegółowego zbadania. Na zakończenie każdego mikrorozkazu logiczna jednostka szeregowania ładuje nowy adres do rejestru adresu sterowania. Zależnie od wartości znaczników stanu ALU i buforowego rejestru sterowania podejmowana jest jedna z trzech decyzji:

- a *Przejsięcie do następnego rozkazu.* Dodanie 1 do rejestru adresu sterowania.
- a *Skok do następnego programu standardowego w wyniku mikrorozkazu skoku.* Ładowanie pola adresu buforowego rejestru sterowania do rejestru adresu sterowania.
- *Skok do programu standardowego kodu maszynowego.* Ładowanie rejestru adresu sterowania na podstawie kodu operacji w IR.

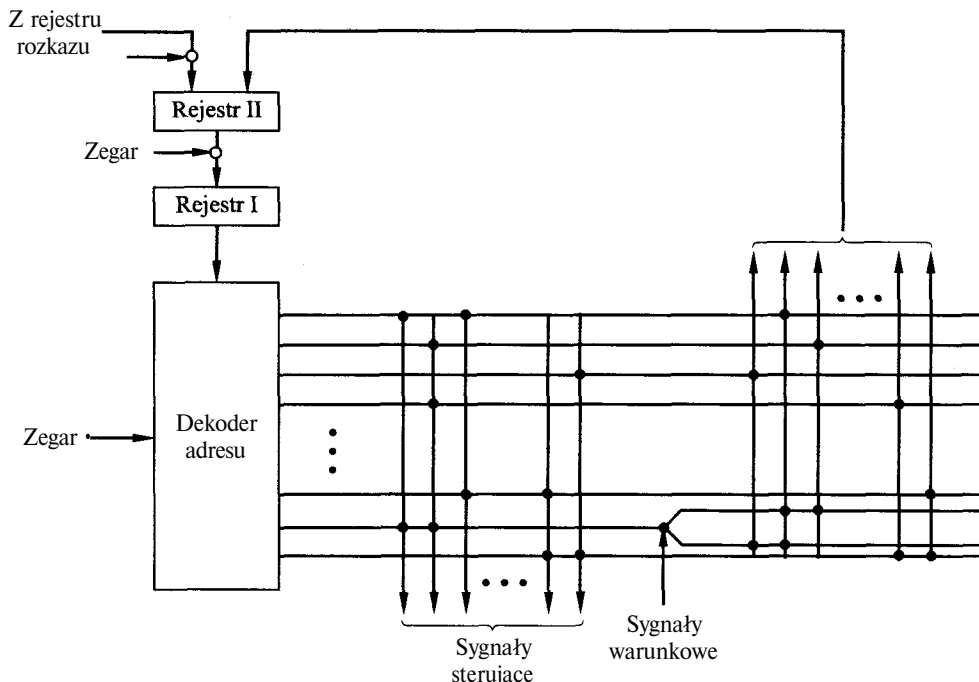
I ostatnia sprawa: na rys. 15.4 są pokazane dwa moduły oznaczone jako *dekodery*. Górny dekodery przekształca kod operacji z rejestru IR na adres w pamięci sterującej. Dolny dekodery nie jest używany w przypadku mikrorozkazów

poziomych, jest natomiast potrzebny w przypadku mikrorozkazów pionowych (rys. 15.1b). Jak już wspomnieliśmy, w mikrorozkazie poziomym każdy bit pola sterowania jest związany z linią sterowania. W mikrorozkazie pionowym dla każdego działania (np. $MAR \leftarrow (PC)$) jest używany kod, a dekodery przekształca ten kod na indywidualne sygnały sterujące. Zaletą mikrorozkazów pionowych jest to, że są one bardziej zwarte (zawierają mniej bitów) niż mikrorozkazy poziome, kosztem niewielkiej ilości dodatkowych układów logicznych oraz opóźnień.

Sterowanie wg Wilkesa

Jak już wspomnieliśmy, Wilkes jako pierwszy zaproponował w roku 1951 użycie mikroprogramowanej jednostki sterującej [WILK51]. Propozycja ta została następnie opracowana w postaci bardziej szczegółowego projektu [WILK53]. Pouczające jest przeanalizowanie tej źródłowej propozycji.

Wilkes skoncentrował się na opracowaniu systematycznego podejścia do projektowania jednostki sterującej. Konfiguracja, jaką zaproponował, jest przedstawiona na rys. 15.5. Sercem systemu jest matryca wypełniona częściowo diodami. Podczas cyklu maszynowego jeden wiersz matrycy jest wzbudzany za pomocą impulsu. Powoduje to pojawienie się sygnałów w tych punktach, w których są umieszczone diody (wskazane na rysunku za pomocą kropki). Pierwsza



RYСУNEK 15.5. Mikroprogramowana jednostka sterująca Wilkesa

część wiersza generuje sygnały sterujące działaniem procesora. Druga część generuje adres wiersza, który ma być wzbudzany w następnym cyklu maszynowym. Wobec tego każdy wiersz matrycy jest jednym mikrorozkazem, a matryca jest pamięcią sterującą.

Na początku cyklu adres wiersza, który ma być pobudzany, znajduje się w rejestrze I. Adres ten jest wprowadzany do dekodera, który, jeśli jest aktywowany przez impuls zegara, pobudza jeden wiersz matrycy. Zależnie od sygnałów sterowania albo kod operacji pochodzący z rejestru rozkazu, albo druga część wzbudzonego wiersza są podczas tego cyklu kierowane do rejestru II. Zawartość rejestru II jest następnie bramkowana do rejestru I za pomocą impulsu zegara. Kolejne impulsy zegara są używane do wzbudzania wiersza matrycy i do przenoszenia zawartości rejestru II do rejestru I. Układ dwurejestrowy jest konieczny, ponieważ dekodery jest po prostu układem kombinacyjnym. Przy zastosowaniu tylko jednego rejestru, wartość wejściowa stałaby się wartością wyjściową w ciągu tego samego cyklu, co spowodowałoby niestabilność.

Schemat ten jest bardzo podobny do mikroprogramowania poziomego opisanego wcześniej (rys. 15.1a). Główna różnica jest następująca. W układzie opisanym poprzednio stan rejestru adresu sterowania powinien być zwiększony o 1 w celu otrzymania następnego adresu. Natomiast w schemacie Wilkesa następny adres jest zawarty w mikrorozkazie. Aby umożliwić rozgałęzienia, wiersz musi zawierać dwie części adresowe sterowane za pomocą sygnału warunkowego (np. znacznika stanu), co widać na rysunku.

Po zaproponowaniu tego układu Wilkes podał przykład jego użycia w postaci projektu jednostki sterującej prostej maszyny. Przykład ten będący pierwszym znanym projektem mikroprogramowanego procesora, wart jest przedstawienia tutaj, ponieważ ilustruje wiele współczesnych zasad mikroprogramowania.

Procesor hipotetycznej maszyny zawiera następujące rejestry:

- A - mnożna,
- B - akumulator (najmniej znacząca połowa),
- C - akumulator (najbardziej znacząca połowa),
- D - rejestr przesuwany.

Ponadto występują trzy rejestry i dwa 1-bitowe znaczniki stanu dostępne wyłącznie dla jednostki sterującej:

- E - służy zarówno jako rejestr adresu pamięci (MAR), jak i rejestr do czasowego przechowywania,
- F - licznik programu,
- G - drugi rejestr do czasowego przechowywania, używany do liczenia.

W tabeli 15.1 znajduje się lista rozkazów maszynowych dla tego przykładu. W tabeli 15.2 jest kompletna lista mikrorozkazów wyrażona w formie symbolicznej i stanowiąca wdrożenie jednostki sterującej. W celu pełnego zdefiniowania systemu potrzeba więc zaledwie 38 mikrorozkazów.

TABELA 15.1. Lista rozkazów maszynowych w przykładzie Wilkesa

Notacja:	Acc - akumulator Acc_1 - najbardziej znacząca połowa bitów akumulatora Acc_2 - najmniej znacząca połowa bitów akumulatora n - lokacja pamięci n $C(X)$ - zawartość X (X - rejestr lub lokacja pamięci)
Rozkaz	Wynik
$A\ n$	$C(Acc) + C(n)$ do Acc_1
$S\ n$	$C(Acc) - C(n)$ do Acc_1
$H\ n$	$C(n)$ do Acc_2
$V\ n$	$C(Acc_2) \times C(n)$ do Acc , gdzie $C(n) > 0$
$T\ n$	$C(Acc_1)$ do n , 0 do Acc
$U\ n$	$C(Acc_1)$ do n
$R\ n$	$C(Acc) \times 2^{-(n+1)}$ do Acc
$L\ n$	$C(Acc) \times 2^{n+1}$ do Acc
$G\ n$	Jeśli $C(Acc) < 0$, to przenieś sterowanie do n ; jeśli $C(Acc) > 0$, to ignoruj (tzn. kontynuuj szeregowo)
$/\ n$	Wczytaj następny znak na wejściu do n
$O\ n$	Wyślij $C(n)$ do wyjścia

W pierwszej kolumnie jest podany adres (numer wiersza) każdego mikrorozkazu. Adresy te odpowiadają wymienionym kodom operacji. Gdy więc jest napotkany kod operacji rozkazu dodawania (A), to wykonywany jest mikrorozkaz zawarty w lokacji 5. W kolumnach 2 i 3 są przedstawione działania realizowane odpowiednio przez ALU i jednostkę sterującą. Każde wyrażenie symboliczne musi być przekształcone na zbiór sygnałów sterujących (bitów mikrorozkazu). Kolumny 4 i 5 dotyczą ustawiania i używania dwóch znaczników stanu (przerzutników). W kolumnie 4 jest określony sygnał, który powoduje ustawienie znaczników stanu. Na przykład $(1)C_s$ oznacza, że znacznik stanu numer 1 jest ustawiany przez bit znaku liczby w rejestrze C. Jeśli kolumna 5 zawiera identyfikator znacznika stanu, to kolumny 6 i 7 zawierają dwa alternatywne adresy mikrorozkazów, które mają być użyte. W przeciwnym razie kolumna 6 określa adres następnego mikrorozkazu do pobrania.

Rozkazy od O do 4 składają się na cykl pobierania. Mikrorozkaz 4 przekazuje kod operacji do dekodera, który generuje adres mikrorozkazu odpowiadający przewidzianemu do pobrania rozkazowi maszynowemu. Czytelnik, na podstawie dokładnego przestudiowania tabeli 15.2, powinien być w stanie zrozumieć działanie jednostki sterującej.

Zalety i wady

Główną zaletą użycia mikroprogramowania do implementacji jednostki sterującej jest uproszczenie projektowania tej jednostki. Rozwiązanie takie jest zarówno tańsze, jak i bardziej odporne na błędy. Układowa jednostka sterująca musi zawierać skomplikowane układy logiczne w celu szeregowania wielu mikroope-

TABELA 15.2. Mikrorozkazy w przykładzie Wilkesa

Notacja: $A, B, C, \dots$ reprezentują różne rejestry w jednostce arytmetycznej i w jednostce sterującej. ' C do D ' wskazuje, że układy przełączające łączą wyjście rejestru C z rejestrem wejściowym D ; ' $(D + A)$ do C ' wskazuje, że rejestr wyjściowy A jest połączony z jednym z wejść sumatora (wyjście D jest na stałe dołączone do innego wejścia), a wyjście sumatora z rejestrem C .

Symbol numeryczny n w cudzysłowie (np. ' n ') reprezentuje źródło, którego wyjściem jest liczba n wyrażona w jednostkach najmniej znaczącej cyfry.

Jednostka arytmetyczna		Re estrowa jednostka sterująca	Przerzutnik warunkowy		Następny mikrorozkaz	
			Ustawienie	Użycie	0	1
0		F do G i E			1	
1		$(G$ do ' 1 ') do F			2	
2		Pamięć do G			3	
3		G do E			4	
4		E do dekodera			-	
A 5	C do D				16	
S 6	C do D				17	
H 7	Pamięć do B				0	
V 8	Pamięć do A				27	
T 9	C do pamięci				25	
U 10	C do pamięci				0	
tf 11	B do D	E do G			19	
L 12	C do D	E do G			22	
G 13		E do G	(1) C_5		18	
/ 14	Wejście do pamięci				0	
O 15	Pamięć do wyjścia				0	
16	$(D + \text{pamięć})$ do C				0	
17	$(D - \text{pamięć})$ do C				0	
18				1	0	1
19	D do B (R)*	$(G - '1')$ do E			20	
20	C do D		(1) E_5		21	
21	D do C (R)			1	11	0
22	D do C (L)**	$(G - '1')$ do E			23	
23	B do D		(1) E_5		24	
24	D do B (L)			1	12	0
25	'0' do B				26	
26	B do C				0	
27	'0' do C	'18' do E			28	
28	B do D	E do G	(1) B_1		29	
29	D do B (R)	$(G - '1')$ do E			30	
30	C do D (R)		(2) E_5	1	31	32
31	D do C			2	28	33
32	$(D + A)$ do C			2	28	33
33	B do D		(1) B_1		34	
34	D do B (R)				35	
35	C do D (R)			1	36	37
36	D do C				0	
37	$(D - A)$ do C				0	

*Przesunięcie w prawo. Układy przełączające w jednostce arytmetycznej są połączone tak, że najmniej znacząca cyfra rejestru C jest umieszczana podczas mikrooperacji przesuwania w prawo w najbardziej znaczącym miejscu rejestru B , a najbardziej znacząca cyfra rejestru C (cyfra znaku) jest powtarzana (dzięki temu wykonywana jest poprawka w przypadku liczb ujemnych).

**Przesunięcie w lewo. Układy przełączające są połączone tak, aby podczas mikrooperacji przesunięcia w lewo przepuścić najbardziej znaczącą cyfrę rejestru B do najmniej znaczącego miejsca rejestru C .

racji w cyklu rozkazu. Jednak dekodery i logiczna jednostka szeregowania w mikroprogramowanej jednostce sterującej są bardzo prostymi układami logicznymi.

Główną wadą mikroprogramowanej jednostki sterującej jest to, że jest ona nieco wolniejsza od jednostki układowej wykonanej w porównywalnej technologii. Mimo to mikroprogramowanie jest dominującą metodą wdrażania jednostek sterujących we współczesnych komputerach, przede wszystkim z powodu łatwości stosowania. Teraz przeanalizujemy rozwiązanie mikroprogramowane bardziej szczegółowo.

15.2

SZERELOWANIE MIKROROZKAZÓW

Dwoma podstawowymi zadaniami realizowanymi przez mikroprogramowaną jednostkę sterującą są:

- a *Szeregowanie mikrorozkazów.* Odczytywanie następnego mikrorozkazu z pamięci sterującej.
- a *Wykonywanie mikrorozkazu.* Generowanie sygnałów sterujących potrzebnych do wykonania mikrorozkazu.

Podczas projektowania jednostki sterującej obydwa te zadania muszą być rozpatrywane łącznie, ponieważ oba mają wpływ na format mikrorozkazu i taktowanie jednostki sterującej. W tym podrozdziale skupimy się na szeregowaniu, w minimalnym stopniu zajmując się zagadnieniami formatu i taktowania. Zagadnienia te zostaną przeanalizowane bardziej szczegółowo w następnym podrozdziale.

Rozważania projektowe

Dwa następujące problemy występują przy projektowaniu metody szeregowania mikrorozkazów: rozmiar mikrorozkazu i czas generowania adresu. Pierwszy problem jest oczywisty; minimalizowanie rozmiaru pamięci sterującej zmniejsza koszt tego zespołu. Drugi problem polega na dążeniu do tak szybkiego wykonywania mikrorozkazów, jak tylko jest to możliwe.

Przy wykonywaniu mikroprogramu adres następnego mikrorozkazu przewidzianego do wykonania należy do jednej z trzech kategorii:

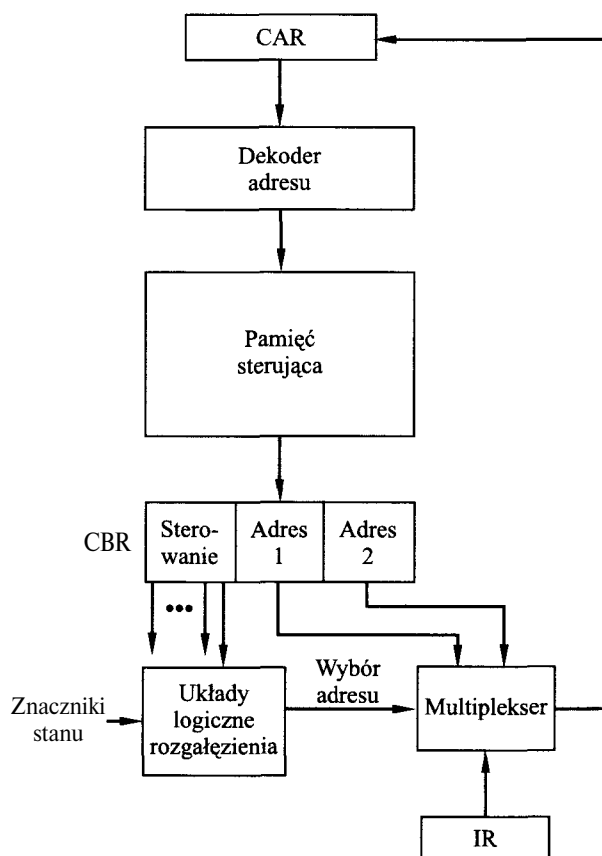
- a określany przez rejestr rozkazu,
- następny adres w sekwencji,
- a rozgałęzienie.

Pierwsza kategoria występuje tylko raz w cyklu rozkazu, tuż po pobraniu rozkazu. Druga kategoria jest najbardziej powszechna w większości projektów.

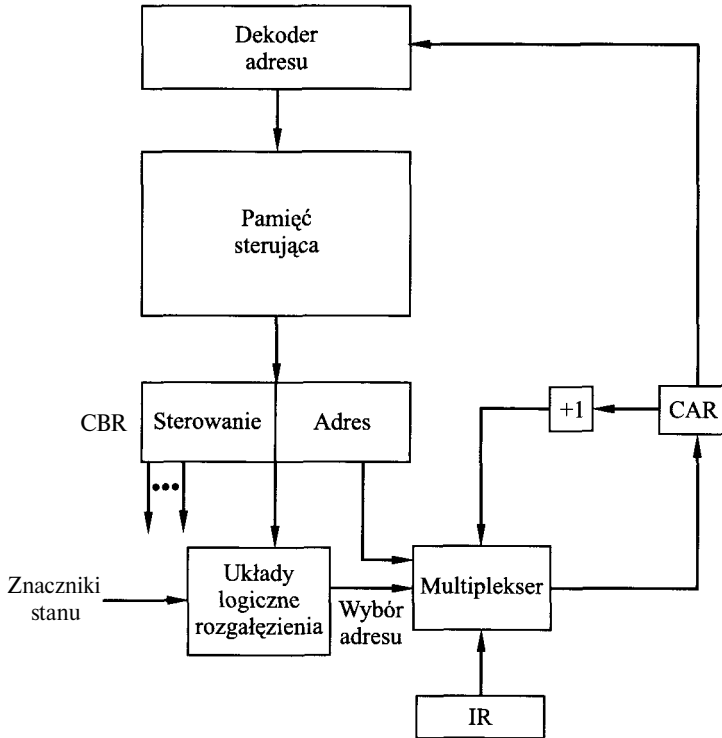
Nie jest jednak możliwa taka optymalizacja projektu, żeby występowały tylko dostępy sekwencyjne. Rozgałęzienia, zarówno warunkowe, jak i bezwarunkowe, są niezbędną częścią mikroprogramu. Ponadto dąży się do używania krótkich sekwencji mikrorozkazów; jeden z każdego trzech lub czterech mikrorozkazów powinien dotyczyć rozgałęzienia [SIEW82]. Ważne jest więc projektowanie zwartych, wydajnych czasowo metod wykonywania mikrorozkazów rozgałęziania.

Metody szeregowania

Adres następnego mikrorozkazu w pamięci sterującej musi być generowany na podstawie bieżącego mikrorozkazu, warunkowych znaczników stanu zawartości rejestru rozkazu. Używa się do tego celu wielu różnych metod. Możemy je pogrupować na trzy ogólne kategorie zilustrowane na rys. 15.6÷15.8 [CLIN81]. Kategorie te są oparte na formacie informacji adresowej w mikrorozkazie:



RYSUNEK 15.6. Układy logiczne sterowania rozgałęzieniami, dwa pola adresowe

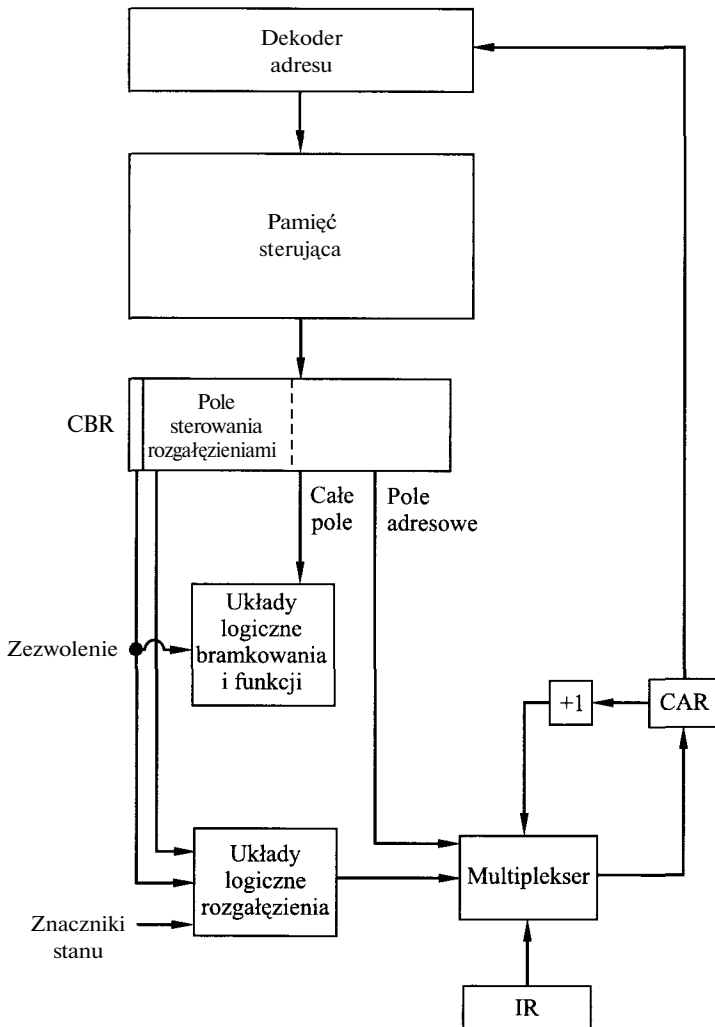


RYСУNEK 15.7. Układy logiczne sterowania rozgałęzieniami, jedno pole adresowe

- a dwa pola adresowe,
- a jedno pole adresowe,
- a format zmienny.

Najprostszym rozwiązaniem jest uwzględnienie dwóch pól adresowych w każdym mikrorozkazie. Na rysunku 15.6 zasugerowano sposób użycia tej informacji. Występuje tu multiplexer służący jako miejsce docelowe obu pól adresowych i zawartości rejestru rozkazu. Na podstawie wejścia wyboru adresu multiplexer przekazuje do rejestru adresu sterowania (CAR) albo kod operacji, albo jeden z dwóch adresów. Zawartość rejestru CAR jest następnie dekodowana w celu utworzenia adresu następnego mikrorozkazu. Sygnały wyboru adresu są doprowadzane przez moduł logiczny rozgałęziania, na którego wejście są podawane znaczniki stanu jednostki sterującej oraz bity pochodzące z części sterowania w mikrorozkazie.

Chociaż rozwiązanie dwuadresowe jest proste, wymaga ono większej ilości bitów w mikrorozkazie niż inne rozwiązania. Oszczędności mogą być poczynione za pomocą pewnych dodatkowych układów logicznych. Powszechnym rozwiązaniem jest stosowanie jednego pola adresowego (rys. 15.7). W tym rozwiązaniu opcje dotyczące pobierania następnego adresu są następujące:



RYSUNEK 15.8. Układy logiczne sterowania rozgałęzieniami, format zmienny

- a pole adresowe,
- a kod rejestru rozkazu,
- następny adres w sekwencji.

Sygnały wyboru adresu określają, która z tych opcji zostaje wybrana. Rozwiązanie to powoduje zredukowanie liczby pól adresowych do jednego. Zauważmy jednak, że często pole to nie będzie używane. Schemat kodowania mikrorozkazu wykazuje więc pewną nieefektywność.

Innym rozwiązaniem jest przyjęcie dwóch całkowicie różnych formatów mikrorozkazu (rys. 15.8). Za pomocą jednego bitu określa się, który format będzie używany. W jednym formacie pozostałe bity są używane do aktywowania

sygnałów sterujących. W drugim formacie niektóre bity sterują modułem logicznym rozgałęziania, a pozostałe dostarczają adres. W przypadku pierwszego formatu następny adres jest albo następnym adresem w sekwencji, albo adresem wyprowadzanym z rejestru rozkazu. W przypadku drugiego formatu określane jest albo rozgałęzienie warunkowe, albo bezwarunkowe. Wadą tego rozwiązania jest to, że jeden **mikrorozkaz** rozgałęzienia pochłania cały cykl. W przypadku pozostałych rozwiązań generowanie adresu następuje jako część tego samego cyklu, co generowanie sygnałów sterujących. Powoduje to zminimalizowanie liczby odniesień do pamięci sterującej.

Oczywiście, opisane rozwiązania mają charakter ogólny. Konkretnie wdrożenia często obejmują odmiany lub kombinacje tych metod.

Generowanie adresu

Spoglądaliśmy na problem szeregowania z punktu widzenia rozważań nad formatem i ogólnych wymagań dotyczących logiki. Innym punktem widzenia jest rozważenie różnych sposobów wyprowadzania lub obliczania następnego adresu.

W tabeli 15.3 wymieniono różne metody generowania adresu. Mogą one być podzielone na metody jawne, w których adres jest jawnie osiągalny w **mikrorozkazie**, oraz metody domyślne wymagające dodatkowych układów logicznych do wygenerowania adresu.

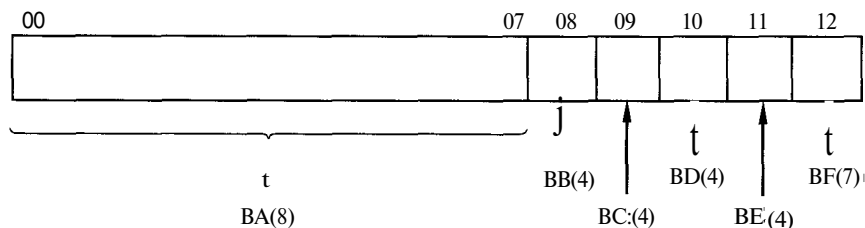
TABELA 15.3. Metody generowania adresu mikrorozkazu

Jawne	Niejawne
Dwupolowa	Odwzorowanie
Rozgałęzienie bezwarunkowe	Dodawanie
Rozgałęzienie warunkowe	Sterowanie szcztąkowe

W zasadzie zajmowaliśmy się metodami jawnymi. W przypadku rozwiązania dwuadresowego w każdym **mikrorozkazie** osiągalne są dwa alternatywne adresy. Przy zastosowaniu pojedynczego pola adresowego lub formatu zmiennego mogą być wdrożone różne rozkazy rozgałęziania. Rozkaz rozgałęzienia warunkowego zależy od następujących rodzajów informacji:

- a znaczników stanu ALU,
- a części pól kodu operacji lub trybu adresowania w rozkazie maszynowym,
- a części wybranego rejestru, takiej jak bit znaku,
- Q bitów stanu wewnątrzjednostki sterującej.

Kilka metod domyślnych również znajduje powszechne zastosowanie. Jedną z nich, **odwzorowywanie**, jest wymagana we wszystkich praktycznie projektach. Część kodu operacji rozkazu maszynowego musi być odwzorowana w adresie mikrorozkazu. Rozwiązanie to zostało przyjęte w rodzinie IBM S/360 [TUCK67] oraz w wielu modelach S/370. Jako przykładu użyjemy IBM 3033.



RYSUNEK 15.9. Rejestr adresu sterowania IBM 3033

Rejestr adresu sterowania w IBM 3033 ma długość 13 bitów. Został pokazany na rys. 15.9. Można wyróżnić dwie części adresu. Ośiem najstarszych bitów (00-07) normalnie nie ulega zmianie przy przejściu od jednego cyklu mikrorozkazu do następnego. Podczas wykonywania mikrorozkazu tych 8 bitów kopiuje się bezpośrednio z 8-bitowego pola mikrorozkazu (pole BA) do 8 najstarszych bitów rejestru adresu sterowania. W ten sposób definiuje się blok 32 mikrorozkazów w pamięci sterowania. Pozostałych 5 bitów rejestru adresu sterowania jest ustawianych w celu określenia specyficznego adresu mikrorozkazu, który ma być pobrany jako następny. Każdy z tych bitów jest określany przez 4-bitowe pole (z wyjątkiem jednego, który jest związany z polem 7-bitowym) w bieżącym mikrorozkazie; pole określa warunek ustawiania odpowiedniego bitu. Na przykład, bit w rejestrze adresu sterowania może być ustawiony jako 1 lub 0 zależnie od tego, czy wystąpiło przeniesienie w ostatniej operacji ALU.

Ostatnie rozwiązanie wymienione w tabeli 15.3 nosi nazwę **sterowania resztkowego** (ang. *residual control*). Obejmuje ono użycie adresu mikrorozkazu, który został uprzednio zachowany w tymczasowym miejscu wewnątrz jednostki sterującej. Na przykład, niektóre listy mikrorozkazów umożliwiają stosowanie programów standardowych. Do przechowywania adresów powrotnych jest używany rejestr wewnętrzny lub stos rejestrów. Przykładem takiego rozwiązania jest LSI-11, który teraz przeanalizujemy.

Szeregowanie mikrorozkazów w LSI-11

LSI-11 jest mikrokomputerową wersją PDP-11, przy czym główne podzespoły systemu zostały umieszczone na jednej płycie drukowanej. W LSI-11 zastosowano mikroprogramowaną jednostkę sterującą [SEBE76, CLIN81].

Mikrokomputer LSI-11 używa 22-bitowego mikrorozkazu i pamięci sterowania o pojemności 2K słów 22-bitowych. Adres następnego mikrorozkazu jest wyznaczany na jeden z pięciu sposobów:

- ❑ *Jako następny, kolejny adres.* Gdy nie występują inne rozkazy, zawartość rejestru adresu sterowania w jednostce sterującej jest zwiększana o 1.
- Q *Przez odwzorowanie kodu operacji.* Na początku każdego cyklu rozkazu adres następnego mikrorozkazu jest określany przez kod operacji.
- Q *Jako możliwość podprogramu standardowego.* Wyjaśniona poniżej.

- *W wyniku testowania przerwania.* Pewne mikrorozkazy precyzują sprawdzanie przerw. Jeśli występuje przerwanie, określa to adres następnego mikrorozkazu.
- a *W wyniku rozgałęzienia.* Używane są mikrorozkazy rozgałęzienia warunkowego i bezwarunkowego.

Przewidziana jest możliwość jednopoziomowego podprogramu standardowego. W każdym mikrorozkazie poświęcono temu jeden bit. Jeśli jest on ustawiony, to do 11-bitowego rejestru powrotu jest ładowana zaktualizowana zawartość rejestru adresu sterowania. Następny mikrorozkaz określający powrót spowoduje, że rejestr adresu sterowania zostanie załadowany zawartością rejestru powrotu.

Rozkaz powrotu jest jedną z form rozkazu rozgałęzienia bezwarunkowego. Inna z form takiego rozkazu powoduje, że do rejestru adresu sterowania jest ładowanych 11 bitów mikrorozkazu. Rozkaz rozgałęzienia warunkowego wykorzystuje 4-bitowy kod testowy w ramach mikrorozkazu. Kod ten określa testowanie różnych kodów warunkowych ALU w celu stwierdzenia decyzji rozgałęzienia. Jeśli warunek nie jest spełniony, to wybierany jest następny adres w kolejności. Jeśli natomiast jest spełniony, to do 8 najmniej znaczących pozycji bitowych rejestru adresu sterowania jest ładowanych 8 bitów mikrorozkazu. Pozwala to na rozgałęzianie w ramach strony pamięci o pojemności 256 słów.

Jak można zauważyć, jednostka sterująca LSI-11 ma wielkie możliwości szeregowania adresów. Daje to programiście znaczną elastyczność i może ułatwić zadanie mikroprogramowania. Z drugiej strony takie rozwiązanie wymaga większej liczby układów logicznych w jednostce sterującej niż rozwiązania prostsze.

15.3

WYKONYWANIE MIKROROZKAZÓW

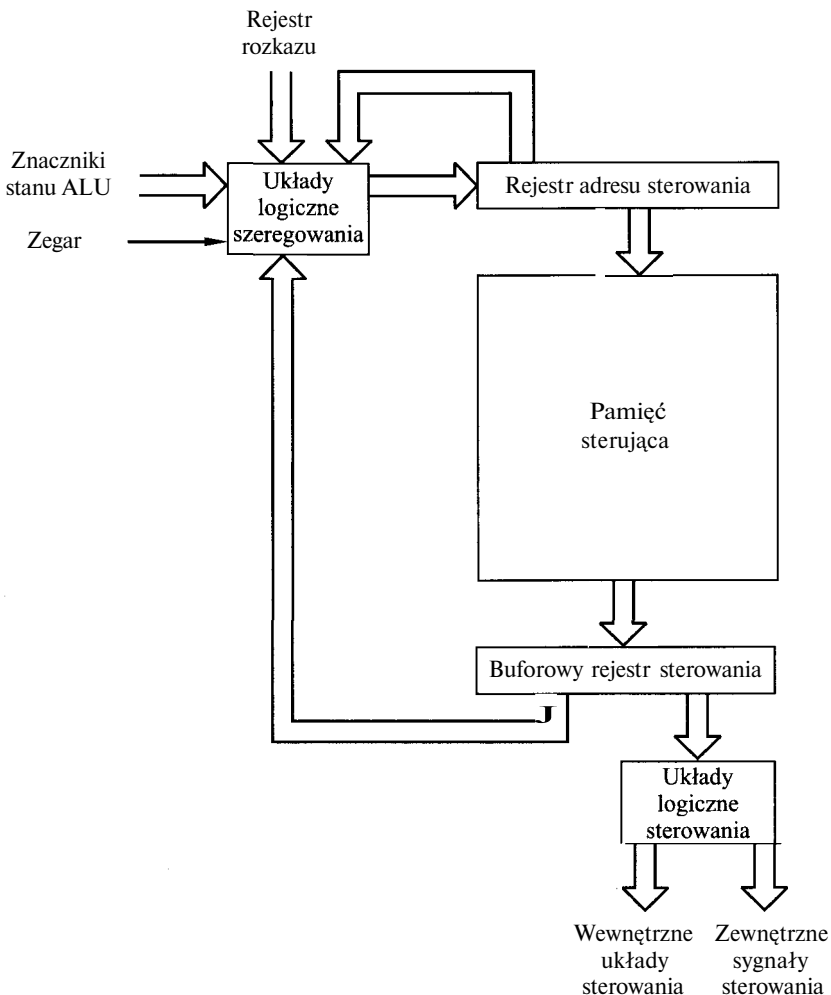
Cykl mikrorozkazu jest podstawowym zdarzeniem w mikroprogramowanym procesorze. Każdy cykl składa się z dwóch części: pobierania i wykonywania. Część pobierania jest określona przez generowanie adresu mikrorozkazu i o tym mówiliśmy w poprzednim podrozdziale. W tym podrozdziale zajmiemy się wykonywaniem mikrorozkazu.

Przypomnijmy sobie, co się dzieje w wyniku wykonania mikrorozkazu. W istocie rezultatem wykonania mikrorozkazu są sygnały sterujące. Niektóre z tych sygnałów sterujących są rozsyłane wewnątrz procesora. Pozostałe są kierowane do zewnętrznej magistrali sterowania lub do zewnętrznego interfejsu. Oprócz tego jest określany adres następnego mikrorozkazu.

Z tego opisu wynika organizacja jednostki sterującej, jaką pokazano na rys. 15.10. Ta nieco zrewidowana wersja rys. 15.4 stanowi nasz główny punkt

zainteresowania w tym podrozdziale. Rola podstawowych modułów na tym rysunku powinna teraz być oczywista. Logiczny moduł szeregowania zawiera układy logiczne wykonujące funkcje omówione w poprzednim podrozdziale. Generuje on adres następnego mikrorozkazu na podstawie rejestru rozkazu, znaczników stanu ALU, rejestru adresu sterowania (w celu zwiększania jego stanu) oraz buforowego rejestru sterowania. Ten ostatni może dostarczać rzeczywistego adresu, bitów sterowania lub jednego i drugiego. Moduł jest sterowany zegarem, który określa taktowanie cykli mikrorozkazów.

Moduł logiczny sterowania generuje sygnały sterujące jako funkcję niektórych bitów mikrorozkazu. Powinno być jasne, że format i zawartość mikrorozkazu określa złożoność logicznego modułu sterowania.



RYSUNEK 15.10. Organizacja jednostki sterującej

Taksonomia mikrorozkazów

Mikrorozkazy mogą być klasyfikowane na różne sposoby. W literaturze można znaleźć następujące sposoby podziału mikrorozkazów:

- pionowe/poziome,
- a upakowane/nieupakowane,
- a mikroprogramowane układowo/programowo,
- a kodowane bezpośrednio/pośrednio.

Wszystkie te cechy wpływają na format mikrorozkazu. Żaden z tych terminów nie był używany w literaturze w sposób spójny i precyzyjny. Jednak przeanalizowanie tych rozróżnień jakościowych może posłużyć do wyjaśnienia możliwości występujących przy projektowaniu mikrorozkazów. W następnych punktach rozpatrzmy najpierw istotne problemy projektowania leżące u podstaw kolejnych par własności, po czym przedstawimy koncepcje wynikające z każdej pary.

W oryginalnej propozycji Wilkesa [WILK51] każdy bit mikrorozkazu albo bezpośrednio tworzył sygnał sterujący, albo bezpośrednio określał bit następnego adresu. Zobaczyliśmy w poprzednim podrozdziale, że możliwe są bardziej złożone schematy szeregowania adresów, przy użyciu mniejszej liczby bitów mikrorozkazu. Schematy te wymagają bardziej złożonego logicznego modułu szeregującego. Podobny dylemat występuje w odniesieniu do tej części mikrorozkazu, która dotyczy sygnałów sterujących. Można zaoszczędzić bity słowa sterowania przez zakodowanie informacji sterowania, a następnie jej zdekodowanie w celu utworzenia sygnałów sterujących.

Jak można przeprowadzić to kodowanie? Żeby odpowiedzieć na to pytanie, weźmy pod uwagę K różnych, wewnętrznych i zewnętrznych sygnałów sterujących dostarczanych przez jednostkę sterującą. W schemacie Wilkesa do tego celu służy K bitów mikrorozkazu. Pozwala to na generowanie 2^K możliwych kombinacji sygnałów sterujących podczas dowolnego cyklu rozkazu. Możemy to jednak poprawić zauważając, że nie wszystkie możliwe kombinacje zostaną użyte. Na przykład:

- a Dwa źródła nie mogą być bramkowane do tego samego miejsca docelowego (np. C_2 i C_8 na rys. 14.5).
- a Rejestr nie może jednocześnie być źródłem i miejscem przeznaczenia (np. C_5 i C_{12} na rys. 14.5).
- a W określonej chwili tylko jedna kombinacja sygnałów sterujących może być skierowana do ALU.
- a W określonej chwili tylko jedna kombinacja sygnałów sterujących może być doprowadzona do zewnętrznej magistrali sterowania.

Tak więc dla danego procesora można by wymienić wszystkie możliwe i dopuszczalne kombinacje sygnałów sterujących, uzyskując liczbę możliwości

logicznych; każdy bit generuje określony sygnał sterujący. W miarę jak używane są bardziej zwarte i bardziej ogólne schematy kodowania, potrzebne są coraz bardziej złożone układy logiczne dekodowania. To z kolei może niekorzystnie wpływać na wydajność. Na propagację sygnałów przez bramki bardziej złożonego logicznego modułu sterowania potrzeba więcej czasu. Wobec tego wykonywanie zakodowanych mikrorozkazów trwa dłużej niż rozkazów niezakodowanych.

Wobec tego wszystkie własności wymienione w tabeli 15.4 tworzą pewne widmo strategii projektowania. Na ogół projektowanie zbliża się do jednego końca widma tego zbioru w celu zoptymalizowania wydajności jednostki sterującej. Projekty zbliżone do drugiego krańca widma są bardziej ukierunkowane na optymalizację procesu mikroprogramowania. Istotnie, listy mikrorozkazów zbliżone do drugiego krańca widma wyglądają bardzo podobnie, jak listy rozkazów maszynowych. Dobrym tego przykładem jest projekt LSI-11, który opiszemy w tym podrozdziale. Zwykle, gdy celem jest po prostu wdrożenie jednostki sterującej, projekt będzie zbliżony do tego pierwszego krańca zbioru. Projekt IBM 3033, który również omówimy, należy do tej właśnie kategorii. Jak wykazemy w dalszym ciągu, niektóre systemy umożliwiają różnym użytkownikom budowanie różnych mikroprogramów przy użyciu takich samych rozwiązań mikrorozkazów. W tych przypadkach projekt najprawdopodobniej znajdzie się bliżej drugiego krańca widma.

Możemy teraz posługiwać się wcześniej wprowadzoną terminologią. W tabeli 15.4 jest pokazane, w jaki sposób trzy spośród wymienionych par wyrażenń wiążą się z widmem mikrorozkazów. W istocie, wszystkie te pary opisują ten sam obiekt, jednak akcentują różne cechy projektu.

Stopień upakowania wiąże się ze stopniem identyfikacji między określonym zadaniem sterowania a określonymi bitami mikrorozkazu. Gdy bity stają się bardziej upakowane, dana liczba bitów zawiera więcej informacji. Wobec tego upakowanie wiąże się znaczeniowo z kodowaniem. Terminy *poziomy* i *pionowy* od-

TABELA 15.4. Rodzaje mikrorozkazów

	<i>Własności</i>
Nie zakodowane	Wysoce zakodowane
Wiele bitów	Kilka bitów
Szczegółowy widok sprzętu	Zagregowany widok sprzętu
Trudne do programowania	Łatwe do programowania
W pełni wykorzystana współbieżność	Nie w pełni wykorzystana współbieżność
Niewiele lub brak sterujących układów logicznych	Złożone sterujące układy logiczne
Szybkie wykonywanie	Powolne wykonywanie
Optymalizacja wydajności	Optymalizacja programowania
	<i>Terminologia</i>
Nie upakowane	Upakowane
Poziome	Pionowe
Układowe	Programowe

noszą się do względnej szerokości mikrorozkazów. W [SIEW82] zasugerowano prostą regułę, zgodnie z którą mikrorozkazy pionowe mają długości w przedziale $16+40$ bitów, poziome zaś w przedziale $40+100$ bitów. Terminy *mikroprogramowanie układowe* i *programowe* są używane w celu zasugerowania stopnia zbliżenia do stanowiących podstawę sygnałów sterujących oraz do rozwiązania sprzętowego. Mikroprogramy układowe (ang. *hard microprograms*) są na ogół stałe i osadzone w pamięci stałej. Mikroprogramy programowe (ang. *soft microprograms*) można łatwiej zmieniać i przywodzą na myśl mikroprogramowanie przez użytkownika.

Inna para terminów wymienionych na początku tego podrozdziału odnosi się do kodowania bezpośredniego lub pośredniego. Zajmiemy się teraz tym właśnie rozróżnieniem.

Kodowanie mikrorozkazu

W praktyce nie projektuje się jednostek sterujących jako czysto nie kodowanych lub posługujących się poziomym formatem mikrorozkazu. Używa się przynajmniej pewnego stopnia kodowania w celu zredukowania szerokości pamięci sterowania oraz uproszczenia zadania mikroprogramowania.

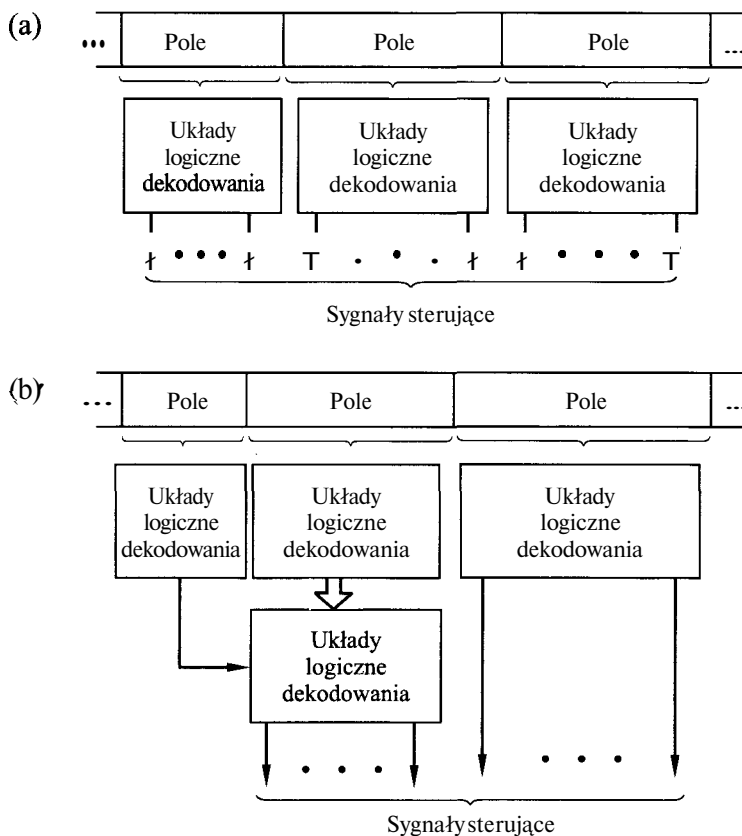
Podstawowa metoda kodowania jest przedstawiona na rys. 15.11a. Mikrorozkaz jest zorganizowany w postaci zbioru pól. Każde pole zawiera kod, który po zdekodowaniu aktywuje jeden lub więcej sygnałów sterujących.

Zastanówmy się, co wynika z tego schematu. Podczas wykonywania mikrorozkazu każde pole jest dekodowane i są generowane sygnały sterujące. Wobec tego przy liczbie N pól jednocześnie prowadzi się N działań. Wynikiem każdego działania jest wysłanie jednego lub wielu sygnałów sterujących. Na ogół, chociaż nie zawsze, chcemy zaprojektować format w ten sposób, żeby każdy sygnał sterujący był wysyłany przez nie więcej, niż jedno pole. Oczywiście musi być możliwe, aby każdy sygnał sterujący był aktywowany przez przynajmniej jedno pole.

Rozważmy teraz indywidualne pole. Pole obejmujące L bitów zawiera jeden z 2^L kodów, z których każdy może być przyporządkowany innej kombinacji sygnałów sterujących. Ponieważ w określonej chwili w polu może pojawić się tylko jeden kod, kody wzajemnie wykluczają się i wobec tego wykluczają się wzajemnie również powodowane przez nie działania.

Projektowanie kodowanego formatu mikrorozkazu może więc być wyrażone w prosty sposób:

- Podział formatu na niezależne pola. Oznacza to, że każde pole obrazuje taki zbiór działań (kombinację sygnałów sterujących), że działania powodowane przez różne pola mogą następować jednocześnie.
- a Zdefiniowanie każdego pola w taki sposób, żeby alternatywne działania określane przez to pole wzajemnie się wykluczały. Oznacza to, że tylko jedno spośród działań precyzowanych przez dane pole może następować w określonej chwili.



RYSUNEK 15.11. Kodowanie mikrorozkazu: (a) kodowanie bezpośrednie; (b) kodowanie pośrednie

Możliwe są dwa podejścia do podziału kodowanego mikrorozkazu na pola: funkcjonalne oraz według zasobów. **Funkcjonalna metoda kodowania** identyfikuje funkcje wewnątrz maszyny i przypisuje pola rodzajom funkcji. Jeśli na przykład mogą występować różne źródła danych przenoszonych do akumulatora, jedno z pól może być przypisane do tego celu, przy czym każdy kod określa inne źródło. **Kodowanie według zasobów** opiera się na postrzeganiu maszyny jako zbioru niezależnych zasobów i polega na przypisaniu każdemu z nich (np. pamięci, wejściu-wyjściu, ALU) jednego pola.

Innym aspektem kodowania jest to, czy jest ono bezpośrednie, czy pośrednie (rys. 15.11b). W przypadku kodowania pośredniego jedno pole jest używane do określania interpretacji drugiego pola. Rozważmy na przykład ALU, które może przeprowadzić osiem różnych operacji arytmetycznych i osiem różnych operacji przesuwania. Pole 1-bitowe mogłoby być użyte do wskazywania, czy ma być przeprowadzona operacja arytmetyczna, czy przesuwania; pole 3-bitowe wskazywałoby operację. Metoda ta na ogół implikuje dwa poziomy dekodowania, co zwiększa opóźnienia propagacji.

(a) Proste transfery rejestrowe

0	0	0	0	0	0		

 $\text{MDR} \leftarrow \text{Rejestr}$

0	0	0	0	0	1		

 $\text{Rejestr} \leftarrow \text{MDR}$

0	0	0	0	1	0		

 $\text{MAR} \leftarrow \text{Rejestr}$

Wybór rejestru

Operacje pamięci

0	0	1	0	0	0	0	0

Odczyt

0	0	1	0	0	1	0	0

Zapis

Specjalne operacje szeregowania

0	1	0	0	0	0	0	0

 $\text{CSAR} \leftarrow \text{Zdekodowany MDR}$

0	1	0	0	0	1	0	0

 $\text{SCAR} \leftarrow \text{Stała (w następnym bajcie)}$

0	1	0	0	1	0	0	0

Pominięcie

Operacje ALU

0	1	1	0	0	0		

 $\text{ACC} \leftarrow \text{ACC} + \text{Rejestr}$

0	1	1	0	0	1		

 $\text{ACC} \leftarrow \text{ACC} - \text{Rejestr}$

0	1	1	0	1	0		

 $\text{ACC} \leftarrow \text{Rejestr}$

0	1	1	0	1	1		

 $\text{Rejestr} \leftarrow \text{ACC}$

0	1	1	1	0	0		

 $\text{ACC} \leftarrow \text{Rejestr} + 1$

Wybór rejestru

(b) 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

1	2	3	4	5	6

Definicja pola:

- 1 - transfer rejestrowy
- 2 - operacja pamięci
- 3 - operacja szeregowania
- 4 - operacja ALU
- 5 - wybór rejestru
- 6 - stała

RYSUNEK 15.12. Alternatywne formaty mikrorozkazu prostego komputera: (a) repertuar mikrorozkazów pionowych; (b) format mikrorozkazu poziomego

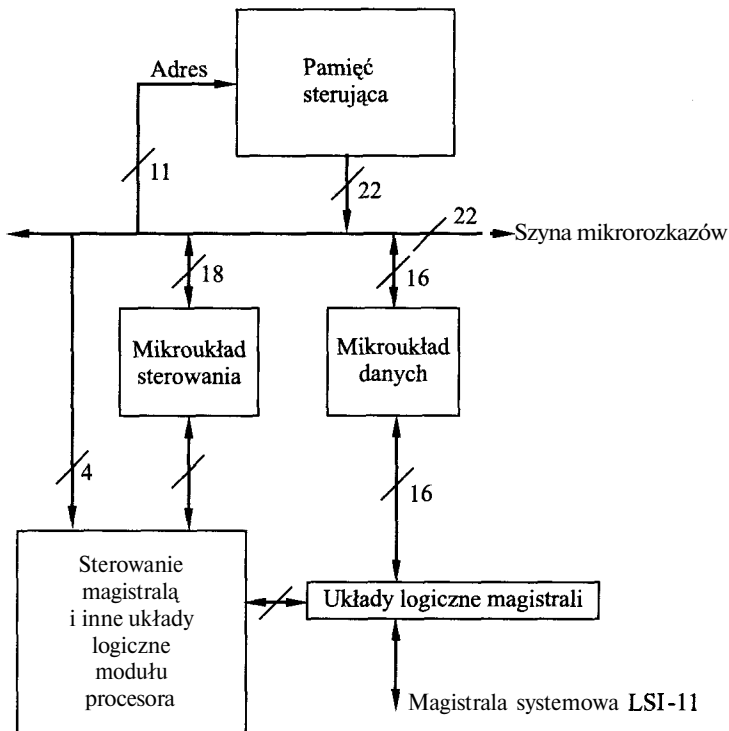
Na rysunku. 15.12, pochodzącym z [RAUS80], jest podany prosty przykład tych koncepcji. Założmy istnienie procesora z jednym akumulatorem i kilkoma rejestrami wewnętrznymi, takimi jak licznik programu i rejestr tymczasowy wyjścia ALU. Na rysunku 15.12a jest pokazane rozwiązanie pionowe. Pierwsze 3 bity wskazują rodzaj operacji, następne 3 są kodem operacji, a ostatnie 2 służą do wybierania rejestru wewnętrznego. Na rysunku 15.12b są przedstawione rozwiązania bardziej poziome, chociaż w dalszym ciągu jest stosowane kodowanie. W tym przypadku różne pola dotyczą różnych funkcji.

Wykonywanie mikrorozkazów w LSI-11

Procesor LSI-11 [SEBE76, CLIN81] stanowi dobry przykład pionowego formatu mikrorozkazu. Najpierw zapoznamy się z organizacją jednostki sterującej a następnie z formatem mikrorozkazu.

Organizacja jednostki sterującej LSI-11

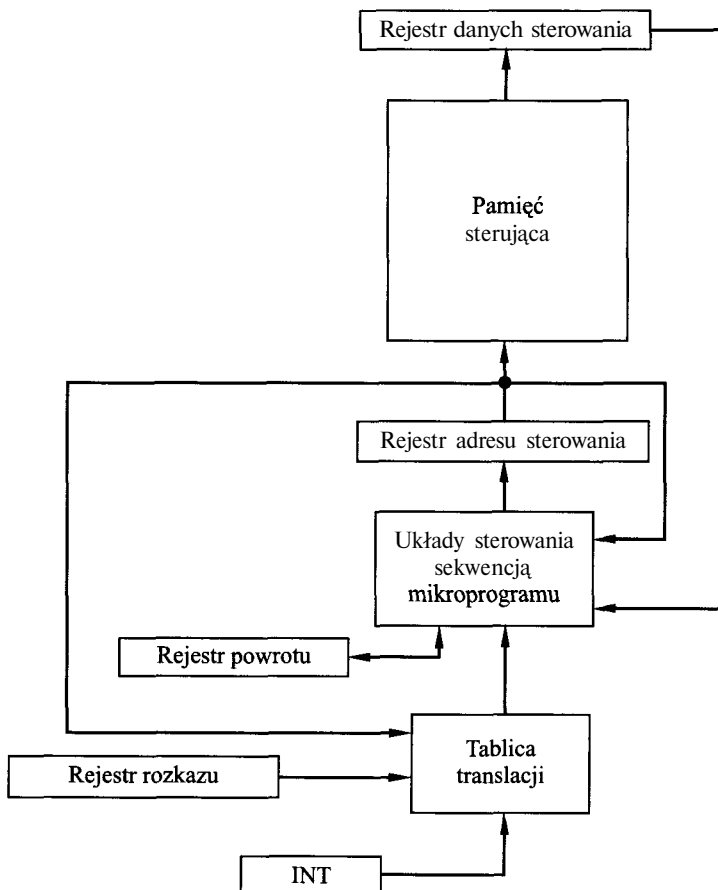
Procesor LSI-11 jest pierwszym modelem rodziny PDP-11 oferowanym jako procesor na pojedynczej płycie drukowanej. Na płycie znajdują się trzy mikroukłady LSI, magistrala wewnętrzna określana jako **magistrala mikrorozkazu**, oraz pewne dodatkowe układy logiczne interfejsu.



RYSUNEK 15.13. Uproszczony schemat blokowy procesora LSI-11

Na rysunku 15.13 widać organizację procesora LSI-11 w postaci uproszczonej. Występują tu trzy mikroukłady: mikroukład danych, mikroukład sterowania i pamięć sterowania. Mikroukład danych zawiera 8-bitową ALU, 26 rejestrów 8-bitowych i pamięć kilku kodów warunkowych. Szesnaście rejestrów posłużyło do wdrożenia 8 16-bitowych rejestrów ogólnego przeznaczenia PDP-11. Pozostałe obejmują rejestr słowa stanu programu, rejestr adresu pamięci (MAR) i buforowy rejestr pamięci. Ponieważ ALU przeprowadza w określonej chwili operacje jedynie na 8 bitach, do wdrożenia 16-bitowej operacji arytmetycznej procesora PDP-11 są wymagane dwa przejścia przez ALU. Jest to sterowane mikroprogramem.

Mikroukład (lub mikroukłady) pamięci sterowania zawiera pamięć sterowania o szerokości 22 bitów. Mikroukład sterowania zawiera też układy logiczne do szeregowania i wykonywania mikrorozkazów oraz rejestr adresu sterowania, rejestr danych sterowania i kopię rejestru rozkazu maszynowego.



RYСУNEK 15.14. Organizacja jednostki sterującej LSI-11

Magistrala MIB łączy wszystkie podzespoły. Podczas pobierania mikrorozkazu mikroukład sterowania generuje 11-bitowy adres i umieszcza go na magistrali MIB. Z pamięci sterowania jest pobierany 22-bitowy mikrorozkaz, który jest lokowany na magistrali MIB. Do mikroukładu danych jest kierowanych 16 najmniej znaczących bitów, podczas gdy 18 najmniej znaczących bitów trafia do mikroukładu sterowania. Specjalne funkcje na płycie procesora są sterowane za pomocą 4 najbardziej znaczących bitów.

Na rysunku 15.14 jest przedstawiony nadal uproszczony, jednak bardziej szczegółowy obraz jednostki sterującej LSI-11 (na rysunku pominięto granice poszczególnych mikroukładów). Schemat szeregowania adresów, opisany w podrozdz. 15.2, został wdrożony w postaci dwóch modułów. Ogólne sterowanie sekwencją zapewnia moduł sterowania sekwencją mikrorozkazów, który ma możliwość zwiększania stanu rejestru adresu mikrorozkazu i przeprowadzania rozgałęzień bezwarunkowych. Inne formy obliczania adresu są przeprowadzane przez oddzielną tablicę translacji. Tablica ta to układ kombinacyjny, który generuje adres oparty na: mikrorozkazie, rozkazie maszynowym, stanie licznika mikroprogramu i stanie rejestru przerwań.

Tablica translacji jest używana w następujących okolicznościach:

- a gdy kod operacji jest używany do rozpoczęcia podprogramu standardowego;
- a gdy bity trybu adresowania w mikrorozkazie są testowane w celu przeprowadzenia właściwego adresowania;
- a gdy okresowo są testowane warunki przerwania;
- a gdy analizowane są mikrorozkazy rozgałęzienia warunkowego.

Format mikrorozkazu LSI-11

Format mikrorozkazu LSI-11 można określić jako krańcowo pionowy. Rozkaz ma szerokość zaledwie 22 bitów. Lista mikrorozkazów ściśle zastępuje listę rozkazów maszynowych PDP-11. Intencją projektu była optymalizacja wydajności jednostki sterującej w ramach ograniczeń stwarzanych przez rozwiązanie pionowe, łatwe do programowania. W tabeli 15.5 są wymienione niektóre mikrorozkazy LSI-11.

Na rysunku 15.15 jest pokazany 22-bitowy format mikrorozkazu LSI-11. Specjalnymi funkcjami sterują 4 najbardziej znaczące bity. Bit translacji zezwala tablicy translacji na sprawdzenie przerwań zawieszonych. Bit ładowania rejestru powrotu jest używany na zakończenie podprogramu standardowego do powodowania ładowania adresu następnego mikrorozkazu z rejestru powrotu.

Pozostałych 16 bitów używa się do zakodowanych mikrorozkazów. Format jest tu podobny do rozkazu maszynowego, z kodem operacji o zmiennej długości i z jednym lub z wieloma argumentami.

TABELA 15.5. Wybrane mikrorozkazy LSI-11

Operacje arytmetyczne

Dodaj słowo (bajt, stałą znakową)
 Zbadaj słowo (bajt, stałą znakową)
 Inkrementuj słowo (bajt) o 1
 Inkrementuj słowo (bajt) o 2
 Zaneguj słowo (bajt)
 Warunkowo inkrementuj (dekrementuj)
 Warunkowo dodaj słowo (bajt)
 Dodaj słowo (bajt) z przeniesieniem
 Warunkowo dodaj cyfry
 Odejmij słowo (bajt)
 Porównaj słowo (bajt, stałą znakową)
 Odejmij słowo (bajt) z przeniesieniem
 Dekrementuj słowo (bajt) o 1

Operacje logiczne

Wykonaj AND na słowie (bajcie, stałej znakowej)
 Zbadaj słowo (bajt)
 Wykonaj OR na słowie (bajcie)
 Wykonaj EXCLUSIVE-OR na słowie (bajcie)
 Zeruj bit słowa (bajtu)

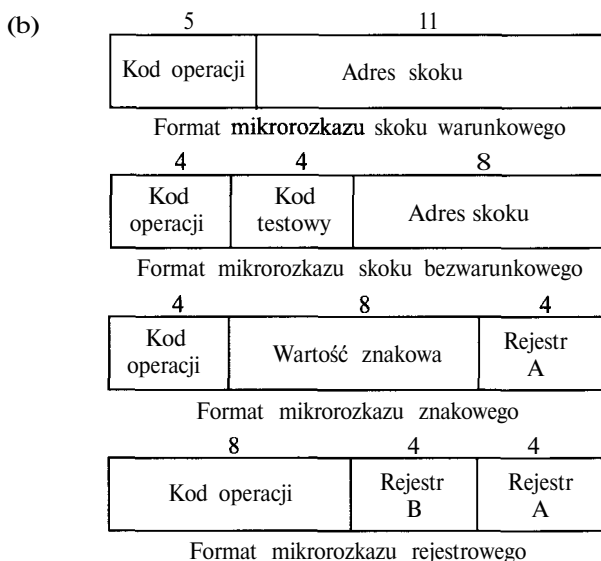
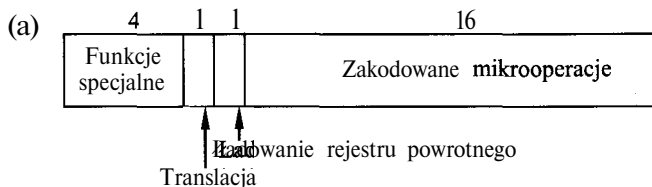
Przesuń słowo (bajt) w prawo (w lewo)
 z przeniesieniem (bez przeniesienia)
 Uzupełnij słowo (bajt)

Operacje ogólne

Przenieś słowo (bajt)
 Skocz
 Powrót
 Skocz warunkowo
 Ustaw (wyzeruj) znaczniki stanu
 Ładuj mniej znaczącą część do G
 Warunkowo przenieś słowo (bajt)

Operacje wejścia-wyjścia

Wprowadź słowo (bajt)
 Wprowadź słowo stanu (bajt)
 Odczytaj
 Odczytaj (zapisz) i inkrementuj słowo (bajt) o 1
 Odczytaj (zapisz) i inkrementuj słowo (bajt) o 2
 Odczytaj (zapisz) potwierdzenie
 Wyprowadź słowo (bajt, stan)



RYSUNEK 15.15. Format mikrorozkazu LSI-11: (a) format pełnego mikrorozkazu LSI-11;
 (b) format zakodowanej części mikrorozkazu LSI-11

Wykonywanie mikrorozkazów w IBM 3033

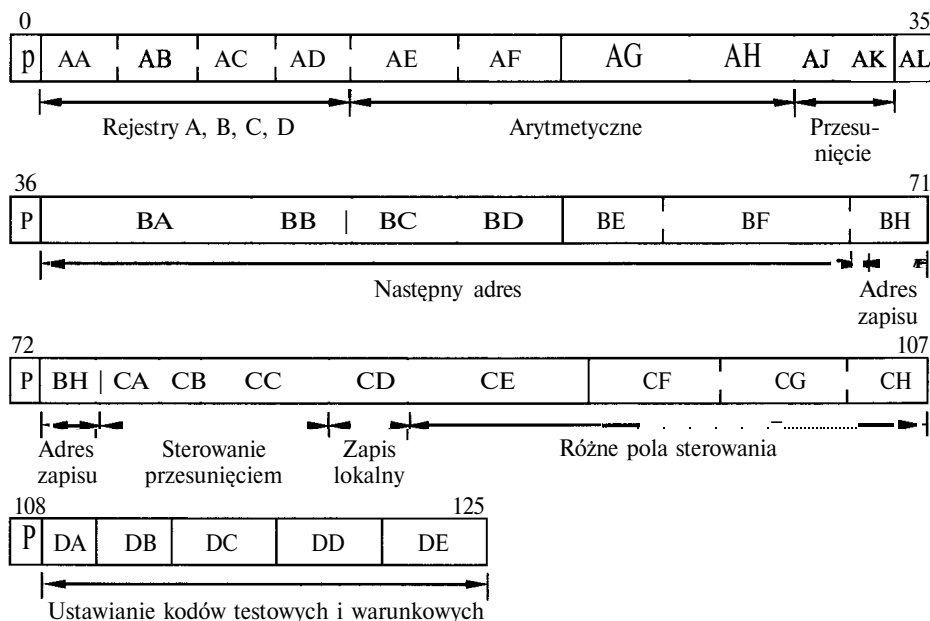
Standardowa pamięć sterowania IBM 3033 składa się z 4K słów. Pierwsza połowa z nich (0000+07FF) zawiera mikrorozkazy 108-bitowe, podczas gdy pozostałe (0800+0FFF) są używane do przechowywania mikrorozkazów 126-bitowych. Format jest przedstawiony na rys. 15.16. Chociaż jest on raczej poziomy, nadal szeroko jest stosowane kodowanie. Główne pola tego formatu są podane w tabeli 15.6.

Dane na wejścia ALU są podawane z czterech wyspecjalizowanych, niewidzialnych dla użytkownika rejestrów (A, B, C i D). Format mikrorozkazu zawiera pola służące do ładowania tych rejestrów z rejestrów widzialnych dla użytkownika, służących do wykonywania działań ALU oraz określania rejestru widzialnego dla użytkownika służącego do przechowania wyniku. Istnieją również pola służące do ładowania i zapisu danych między rejestrami a pamięcią.

Mechanizm szeregowania w IBM 3033 omówiliśmy w podrozdz. 15.2.

TABELA 15.6. Pola sterujące mikrorozkazów IBM 3033

<i>Pola sterujące ALU</i>	
AA(3)	Ładuj rejestr A z jednego spośród rejestrów danych
AB(3)	Ładuj rejestr B z jednego spośród rejestrów danych
AC(3)	Ładuj rejestr C z jednego spośród rejestrów danych
AD(3)	Ładuj rejestr D z jednego spośród rejestrów danych
AE(4)	Skieruj określone bity A do ALU
AF(4)	Skieruj określone bity B do ALU
AG(5)	Określa operację arytmetyczną ALU na wejściu A
AH(4)	Określa operację arytmetyczną ALU na wejściu B
AJ(1)	Określa wejście D lub B do ALU po stronie B
AK(4)	Skieruj wyjście arytmetyczne do przesuwnika
CB(1)	Aktywuj przesuwnik
CC(5)	Określa funkcje logiczne i przeniesienia
CE(7)	Określa wielkość przesunięcia
CA(3)	Ładuj rejestr F
<i>Pola szeregowania i rozgałęziania</i>	
AL(1)	Zakończ operację i przeprowadź rozgałęzienie
BA(8)	Ustaw bity wysokiego rzędu (00-07) sterującego rejestru adresu
BB(4)	Określa warunek ustawiania bitu 8 sterującego rejestru adresu
BC(4)	Określa warunek ustawiania bitu 9 sterującego rejestru adresu
BD(4)	Określa warunek ustawiania bitu 10 sterującego rejestru adresu
BE(4)	Określa warunek ustawiania bitu 11 sterującego rejestru adresu
BF(4)	Określa warunek ustawiania bitu 12 sterującego rejestru adresu



RYSUNEK 15.16. Format mikrorozkazu IBM 3033

15.4

TI 8800

Moduł SDB 8800 (skrót od *Software Development Board*) firmy Texas Instruments jest mikroprogramowanym 32-bitowym modułem komputerowym. W skład układu wchodzi zapisywalna pamięć sterowania, wdrożona w postaci RAM zamiast ROM. Nie osiąga on szybkości ani gęstości systemu mikroprogramowanego z pamięcią sterowania typu ROM. Jest jednak użyteczny do opracowywania prototypów i do celów edukacyjnych.

Moduł SDB 8800 składa się z następujących podzespołów (rys.15.17):

- pamięci mikrorozkazów,
- Q mikrosekwensera,
- a 32-bitowej ALU,
- a procesora zmiennopozycyjnego i całkowitoliczbowego,
- lokalnej pamięci danych.

Podzespoły te są połączone za pomocą dwóch magistrali. Magistrala DA dostarcza danych z pola danych mikrorozkazu do ALU, procesora zmiennopozycyjnego lub mikrosekwensera. W tym ostatnim przypadku dane obejmują adres, który ma być użyty przez rozkaz rozgałęziania. Magistrala ta może również służyć do przekazania danych z ALU lub z mikrosekwensera do pozostałych podzespołów. Magistrala systemu Y łączy ALU i procesor zmiennopozycyjny z pamięcią lokalną i z modułami zewnętrznymi poprzez interfejs PC.

Moduł jest dopasowany do komputerów macierzystych kompatybilnych z IBM PC. Komputer macierzysty stanowi odpowiednią platformę do asemblacji mikroprogramu i jego uruchomienia.

Format mikrorozkazu

Format mikrorozkazu modułu 8800 składa się ze 128 bitów podzielonych na 30 pól funkcjonalnych, co widać w tabeli 15.7. Każde pole składa się z jednego lub wielu bitów. Pola są zgrupowane w pięć głównych kategorii:

- a sterowania modułu,
- a mikroukładu procesora zmiennopozycyjnego i całkowitoliczbowego 8847,
- rejestrowej ALU 8832,
- a mikrosekwensera 8818,
- a pola danych WCS.

Jak widać na rys. 15.17, 32 bity pola danych WCS są wprowadzane do magistrali DA w celu przekazania jako dane do ALU, procesora zmiennopozycyjnego lub sekwensera. Pozostałych 96 bitów (pola 1÷27) mikrorozkazu to sygnały sterujące przekazywane bezpośrednio do odpowiedniego modułu. Dla uproszczenia te pozostałe połączenia nie zostały pokazane na rys. 15.17.

Pierwszych 6 pól dotyczy operacji, które należą do sterowania modułem, a nie pojedynczymi podzespołami. Do operacji sterowania należą:

1. Wybór kodów warunkowych do sterowania sekwenserem. Pierwszy bit pola 1 wskazuje, czy znacznik warunku powinien być ustawiony jako 1, czy jako 0. Pozostałe bity wskazują, który znacznik ma być ustawiony.
2. Wysłanie zapotrzebowania wejścia-wyjścia do PC/AT.
3. Zezwolenie operacji odczytu/zapisu w lokalnej pamięci danych.
4. Wyznaczenie jednostki napędzającej magistralę systemu Y. Wybierany jest jeden z czterech podzespołów dołączonych do magistrali (rys. 15.17).

Ostatnie 32 bity tworzą pole danych zawierające informację specyficzną dla określonego mikrorozkazu.

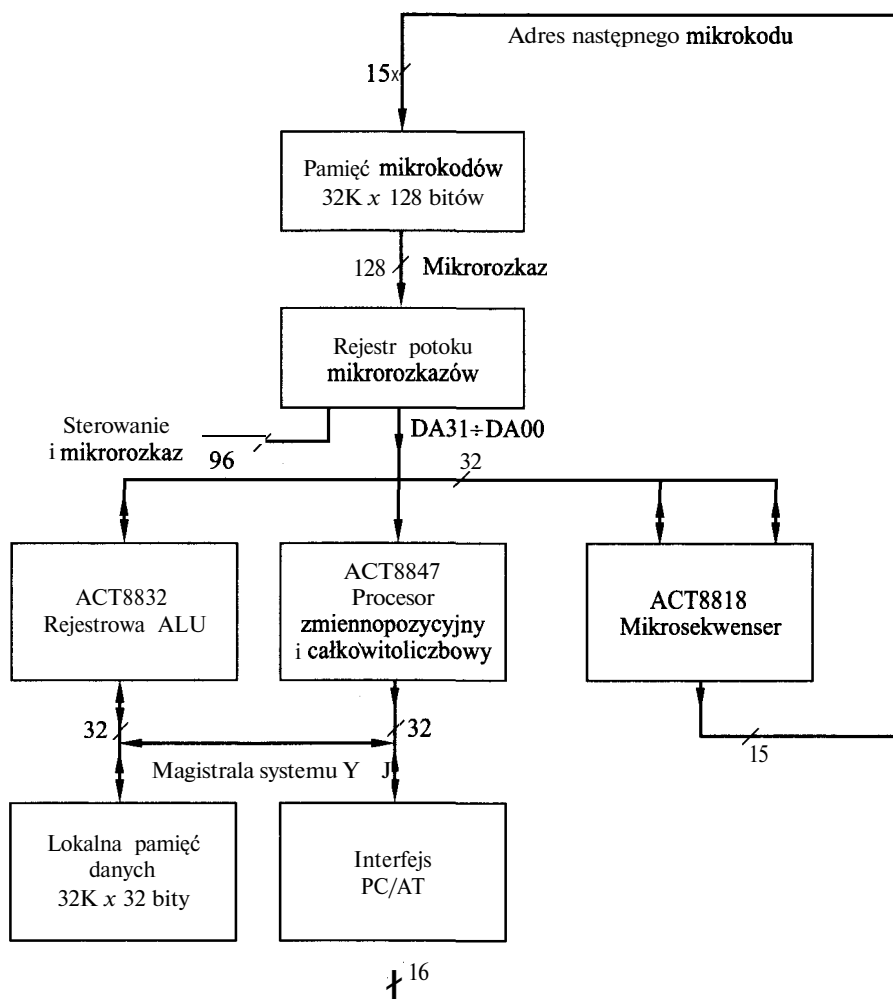
Pozostałe pola mikrorozkazu najlepiej jest omawiać w kontekście podzespołu, który podlega ich kontroli. W pozostałej części tego podrozdziału przedyskutujemy sekwenser i ALU z rejestrami. Jednostkę zmiennopozycyjną pominemy, ponieważ nie wprowadzono tu nowych koncepcji.

Mikrosekwenser

Podstawową funkcją mikrosekwensera 8818 jest generowanie adresu następnego mikrorozkazu w mikroprogramie. Do pamięci mikrorozkazów jest doprowadzany 15-bitowy adres (rys. 15.17).

TABELA 15.7. Format mikrorozkazów TI 8800

Numer pola	Liczba bitów	Opis
<i>Sterowanie modulem</i>		
1	5	Wybierz wejście kodu warunkowego
2	1	Zezwól/zablokuj zewnętrzny sygnał zapotrzebowania wejścia-wyjścia
3	2	Zezwól/zablokuj lokalne operacje odczytu/zapisu pamięci
4	1	Ładuj stan/nie ładuj stanu
5	2	Określ jednostkę sterującą magistralą Y
6	2	Określ jednostkę sterującą magistralą DA
<i>Mikroukład zmiennopozycyjny całkowitoliczbowy 8847</i>		
7	1	Sterowanie rejestrem C: taktuj/nie taktuj
8	1	Wybór najbardziej znaczących lub najmniej znaczących bitów dla magistrali Y
9	1	Źródło danych rejestru C: ALU, multiplexer
10	4	Wybierz tryb IEEE lub FAST dla ALU i MUL
11	8	Wybierz źródła argumentów: rejestry RA, rejestry RB, rejestr P, rejestr S, rejestr C
12	1	Sterowanie rejestrem RB: taktuj/nie taktuj
13	1	Sterowanie rejestrem RA: taktuj/nie taktuj
14	2	Potwierdzenie źródła danych
15	2	Zezwól/zablokuj rejestry potoku
16	11	Działanie ALU 8847
<i>Rejestrowa ALU 8832</i>		
17	2	Zezwolenie/zablokowanie zapisu wyjścia danych w wybranym rejestrze: najbardziej znacząca połowa, najmniej znacząca połowa
18	2	Wybierz źródło danych tablicy rejestrów: magistrala DA, magistrala DB, wyjście Y multiplexera ALU, magistrala systemu Y
19	3	Przesuń modyfikator rozkazu
20	1	Przeniesienie: wymuszaj/nie wymuszaj
21	2	Ustaw tryb konfiguracji ALU: 32, 16 lub 8 bitów
22	2	Wybierz wejście do multiplexera S: tablica rejestrów, magistrala DB, rejestr MQ
23	1	Wybierz wejście do multiplexera R: tablica rejestrów, magistrala DA
24	6	Wybierz rejestr w tablicy C dla zapisu
25	6	Wybierz rejestr w tablicy B dla odczytu
26	6	Wybierz rejestr w tablicy A dla zapisu
27	8	Działanie ALU
<i>Mikroekwenser 8818</i>		
28	12	Sterujące sygnały wejściowe do 8818
<i>Pole danych WCS</i>		
29	16	Najbardziej znaczące bity zapisywalnego pola danych pamięci sterującej
30	16	Najmniej znaczące bity zapisywalnego pola danych pamięci sterującej



RYSUNEK 15.17. Schemat blokowy modułu TI 8800

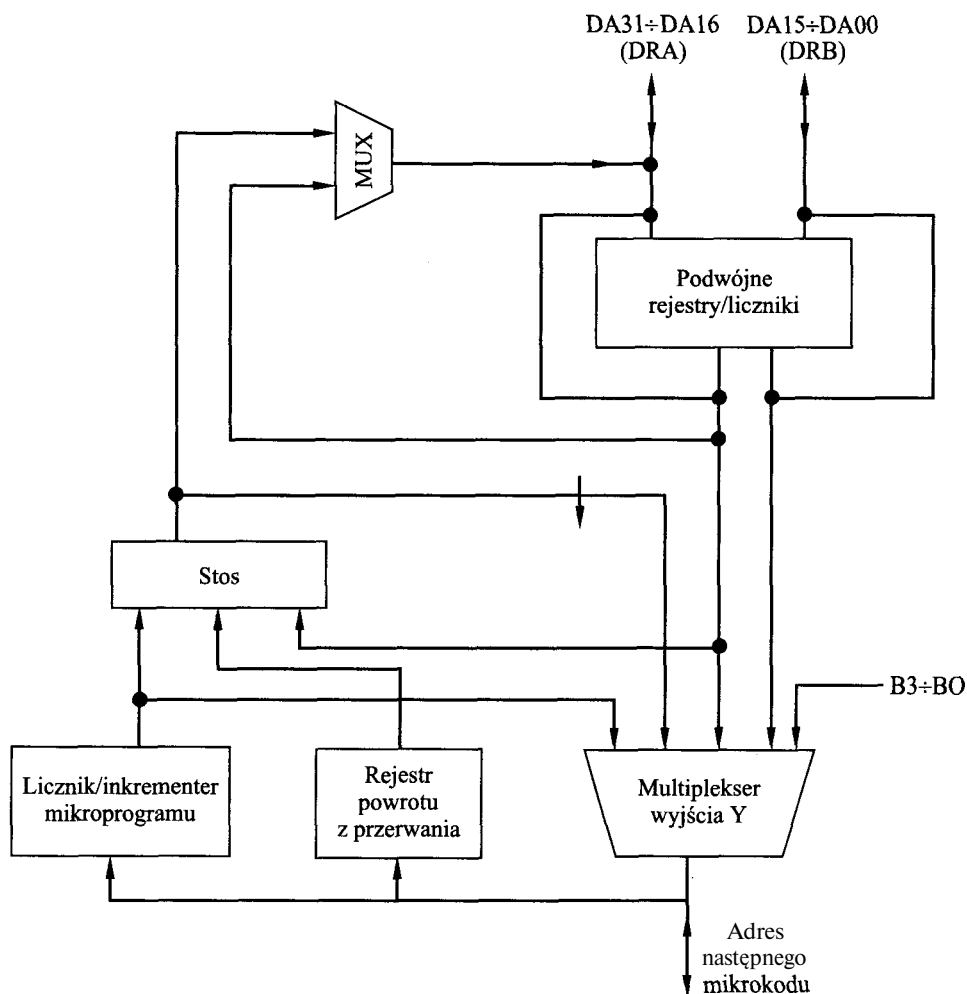
Następny adres może być wybrany z jednego z pięciu źródeł, którymi są:

1. Rejestr licznika mikroprogramu (MPC) używany do powtarzania (ponownego użycia tego samego adresu) i kontynuowania (zwiększania adresu o 1) rozkazów.
2. Stos, który wspiera wywołania podprogramów standardowych, a także pętle iteracyjne i powroty z przerw.
3. Porty DRA i DRB, stanowiące dwie dodatkowe ścieżki z urządzeń zewnętrznych, przez które mogą być generowane adresy mikroprogramu. Porty te są połączone odpowiednio z najbardziej znaczącymi i z najmniej znaczącymi bitami magistrali DA. Dzięki temu sekwenser może otrzymać adres następnego rozkazu z pola danych WCS bieżącego mikrorozkazu lub na podstawie wyniku obliczonego przez ALU.

4. Liczniki rejestrowe RCA i RCB, które mogą być użyte do przechowywania adresu.
5. Zewnętrzne wejście do dwukierunkowego portu Y, wspierające przerwania zewnętrzne.

Na rysunku 15.18 widać blokowy schemat logiczny mikrosekwensera 8818. Podzespół ten składa się z następujących głównych części funkcjonalnych:

- 16-bitowego licznika mikroprogramu (MPC) zbudowanego z rejestru i układu zwiększania stanu rejestru.
- Q Dwoch liczników rejestrowych, RCA i RCB, służących do liczenia pętli i iteracji, przechowywania adresów rozgałęzień lub sterowania przyrządami zewnętrznymi.



RYSUNEK 15.18. Mikrosekwenser TI 8818

- a Stosu o pojemności 65 słów 16-bitowych, umożliwiającego wywoływanie podprogramów standardowych i przerwania.
- a Rejestru powrotu z przerwań oraz układu zezwalania wyjścia Y, służącego do przetwarzania przerwań na poziomie mikrorozkazu.
- Multiplexera wyjścia Y, za pośrednictwem którego następny adres może być wybrany z MPC, RCA, RCB, z magistrali zewnętrznych DRA i DRB lub ze stosu.

Rejestry/liczniki

Rejestry RCA i RCB mogą być ładowane przez magistralę DA albo z bieżącego mikrorozkazu, albo z wyjścia ALU. Ich zawartości mogą być używane jako liczniki do sterowania przebiegiem wykonywania i mogą być automatycznie dekrementowane przy dostępie. Zawartości te mogą być również używane jako adresy mikrorozkazów, które mają być doprowadzone do multiplexera wyjścia Y. Wspierane jest niezależne sterowanie oboma rejestrami podczas cyklu mikrorozkazu, z wyjątkiem jednoczesnego zmniejszania zawartości obu rejestrów.

Stos

Stos umożliwia wiele poziomów zagnieżdżonych wywołań lub przerwań i może być używany do wspierania rozgałęzień i pętli. Pamiętajmy, że operacje te odnoszą się tylko do jednostki sterującej, a nie do całego procesora, i że używane adresy są adresami mikrorozkazów w pamięci sterowania.

Możliwy jest sześć operacji stosu:

1. Kasowanie, które ustawia wskaźnik stosu na zero, opróżniając stos.
2. Zdejmowanie ze stosu, które powoduje dekrementowanie wskaźnika stosu.
3. Umieszczanie na stosie zawartości MPC, rejestru powrotów z przerwań lub magistrali DRA i jednoczesne inkrementowanie wskaźnika stosu.
4. Odczyt adresu wskazanego przez wskaźnik odczytu i udostępnienie go multiplexerowi wyjścia Y.
5. Wstrzymanie, które powoduje, że adres wskaźnika stosu pozostaje niezmienny.
6. Ładowanie wskaźnika stosu za pomocą 7 najmniej znaczących bitów DRA.

Sterowanie mikrosekwenserem

Mikrosekwenser jest sterowany przede wszystkim przez 12-bitowe pole bieżącego mikrorozkazu - pole 28 (tabela 15.7). Pole to składa się z następujących pół cząstkowych:

- a OSEL (1 bit) - pole wyboru wyjścia. Określa, która wartość zostanie umieszczona na wyjściu multiplexera połączonego z magistralą DRA (górn-

ny lewy róg rys. 15.18). Wyjście jest wybierane w ten sposób, że pochodzi albo ze stosu, albo z rejestru RCA. Z magistrali DRA jest on następnie podawany albo na wejście multiplexera wyjścia Y, albo do rejestru RCA.

- Q SELDR (1 bit) - pole wyboru magistrali DR. Jeśli jest ustawione na 1, to oznacza wybór zewnętrznej magistrali DA jako wejścia do magistrali DRA/DRB. Jeśli jest ustawione na 0, to oznacza wybór wyjścia multiplexera DRA do magistrali DRA (sterowane przez OSEL) oraz skierowanie zawartości rejestru RCB do magistrali DRB.
- ZEROIN (1 bit). Używany do wskazywania rozgałęzienia warunkowego. Zachowanie mikrosekwensera będzie następnie zależało od kodu warunkowego wybranego w polu 1 (tabela 15.7).
- a RC2+RC0 (3 bity) - bity sterujące rejestrami. Określają zmianę zawartości rejestrów RCA i RCB. Zawartość rejestrów albo może pozostać bez zmian, albo może być dekrementowana, albo wreszcie rejestry mogą być ładowane z magistrali DRA/DRB.
- S2+S0 (3 bity) - bity sterujące stosem. Określają, jaka operacja stosu ma być wykonywana.
- Q MUX2+MUX0 (3 bity) - bity sterowania wyjściem. W połączeniu z kodem warunkowym (jeśli jest używany) sterują multiplexerem wyjścia Y, określając w ten sposób adres następnego mikrorozkazu. Multiplexer może wybrać swoje wyjście ze stosu, rejestrów DRA, DRB lub MPC.

Bity te mogą być ustawiane indywidualnie przez programistę. Zwykle jednak tak się nie robi. Zamiast tego programista używa mnemoników odpowiadających potrzebnej kombinacji bitów. W tabeli 15.8 jest wymienionych 15 mnemoników dla pola 28. Asembler mikroprogramu zamienia je na odpowiednie wzory bitowe.

TABELA 15.8. Bity mikrorozkazu mikrosekwensera TI 8818 (pole 28)

Mnemonik	Wartość	Opis
RST8818	00000000110	rozkaz sprowadzania do stanu wyjściowego
BRA88181	011000111000	rozgałęziaj do rozkazu DRA
BRA88180	010000111110	rozgałęziaj do rozkazu DRA
INC88181	000000111110	kontynuuj rozkaz
INC88180	001000001000	kontynuuj rozkaz
CAL88181	010000110000	skocz do podprogramu pod adresem określonym przez DRA
CAL88180	010000101110	skocz do podprogramu pod adresem określonym przez DRA
RET8818	000000011010	powrót z podprogramu
PUSH8818	000000110111	umieść adres powrotu z przerwania na stosie
POP8818	100000010000	powrót z przerwania
LOADDRA	000010111110	ładuj licznik DRA z magistrali DA
LOADDRB	000110111110	ładuj licznik DRB z magistrali DA
LOADDRAB	000110111100	ładuj DRA/DRB
DECRDRA	010001111100	dekrementuj licznik DRA i rozgałęziaj, jeśli nie zero
DECRDRB	010101111100	dekrementuj licznik DRB i rozgałęziaj, jeśli nie zero

Jako przykład niech posłuży rozkaz INC88181, który jest używany do wybrania następnego **mikrorozkazu** jako kolejnego w sekwencji, jeśli bieżący wybrany kodem warunkowym jest 1. Z tabeli 15.8 mamy:

INC88181 = 000000111110

co dekoduje się bezpośrednio na

- Q OSEL = 0. Wybranie RCA jako wyjścia doprowadzanego do wyjścia MUX; w tym przypadku wybór ten jest nieistotny.
- a SELDR = 0. Jak wyżej; jest to nieistotne w przypadku tego rozkazu.
- a ZEROIN = 0. W połączeniu z wartością MUX wskazuje, że nie nastąpi rozgałęzienie.
- R = 000. Zachowanie bieżącej wartości RA i RC.
- a S = 111. Zachowanie bieżącego stanu stosu.
- a MUX = 110. Wybranie MPC, jeśli kod warunkowy = 1; wybranie DRA, jeśli kod warunkowy = 0.

Rejestrowa ALU

Moduł 8832 jest 32-bitową ALU z 64 rejestrami, która może być **konfigurowana** w postaci czterech 8-bitowych ALU, dwóch 16-bitowych ALU lub jednej 32-bitowej ALU.

Moduł 8832 jest sterowany przez 39 bitów w polach mikrorozkazu (pola 17-27, tabela 15.7); są one doprowadzane do ALU jako sygnały sterujące. Ponadto, jak widać na rys. 15.17, moduł 8832 jest połączony z 32-bitową magistralą DA i z 32-bitową magistralą systemu Y. Wejścia z magistrali DA mogą być doprowadzone jednocześnie jako wejścia danych do tablicy rejestrów obejmującej 64 słowa oraz do modułu logicznego ALU. Wejście z magistrali systemu Y jest doprowadzone do modułu logicznego ALU. Wyniki operacji ALU oraz przesunięcia są kierowane do magistrali DA lub magistrali systemu Y. Mogą one być również dostarczone do wewnętrznej tablicy rejestrów.

Trzy 6-bitowe porty adresowe umożliwiają jednoczesne pobieranie dwóch argumentów i zapisywanie argumentu w tablicy rejestrów. Przesuwnik MQ i rejestr MQ mogą być również skonfigurowane jako działające niezależnie w celu wdrożenia operacji 8- i 16-bitowych o podwójnej dokładności oraz 32-bitowej operacji przesunięcia.

Pola od 17 do 26 każdego mikrorozkazu sterują sposobem przepływu danych wewnątrz modułu 8832 oraz między 8832 a otoczeniem. Pola te są następujące:

- 17 - *zezwoleń zapisu*. Dwa bity tego pola określają zapis 32 bitów, 16 najbardziej znaczących bitów lub 16 najmniej znaczących bitów, albo też brak zapisu w tablicy rejestrów. Rejestr docelowy jest określany w polu 24.

- 18 - *wybór źródła danych tablicy rejestrów*. Jeśli następuje zapis w tablicy rejestrów, to dwa bity tego pola określają źródło: magistralę DA, magistralę DB, wyjście ALU lub magistralę systemu Y.
- 19 - *modyfikator rozkazu przesunięcia*. Określa opcje dotyczące dostarczania bitów wypełniających oraz odczytu bitów przesuwanych podczas operacji przesunięcia.
- 20 - *przeniesienie do ALU*. Bit ten określa, czy następuje przeniesienie bitu do ALU w danej operacji.
- 21 - *rodzaj konfiguracji ALU*. Moduł 8832 może być skonfigurowany jako jedna 32-bitowa ALU, dwie 16-bitowe ALU lub cztery 8-bitowe ALU.
- 22 - *wejście S*. Wejścia modułu logicznego ALU są obsługiwane przez dwa wewnętrzne multipleksery oznaczane jako S i R. Pole to służy do wybrania wejścia obsługiwanego przez multiplekser S: z tablicy rejestrów, magistrali DB lub rejestru MQ. Rejestr źródłowy jest wybierany w polu 25.
- 23 - *wejście R*. Służy do wybrania wejścia obsługiwanego przez multiplekser R: z tablicy rejestrów lub z magistrali DA.
- 24 - *rejestr docelowy*. Określa adres rejestru w tablicy rejestrów, który ma być użyty dla argumentu docelowego.
- 25 - *rejestr źródłowy*. Określa adres rejestru w tablicy rejestrów, który ma być użyty dla argumentu źródłowego, dostarczany przez multiplekser S.
- 26 - *rejestr źródłowy*. Określa adres rejestru w tablicy rejestrów, który ma być użyty dla argumentu źródłowego, dostarczany przez multiplekser R.

Wreszcie pole 27 zawiera 8-bitowy kod operacji określający działanie arytmetyczne lub logiczne, które ma być realizowane przez ALU. W tabeli 15.9 są wymienione operacje, które mogą być realizowane.

Jako przykład kodowania pól 17+27, rozważmy rozkaz dodania zawartości rejestru 1 do rejestru 2 i umieszczenia wyniku w rejestrze 3. Symbolicznym rozkazem jest

CONT11 [17], WELH, SELRYFYMX, [24], R3, R2, R1, PASS+ADD

Asembler przekształci to na odpowiedni wzór bitowy. Poszczególne składniki rozkazu mogą być opisane następująco:

- a CONT11 jest podstawowym rozkazem NOP.
- Q Pole [17] jest zmieniane na WELH (zezwolenie zapisu, najmniej znaczące i najbardziej znaczące), dzięki czemu następuje wpis do rejestru 32-bitowego.
- Pole [18] jest zmieniane na SELRFYMX w celu wybrania wyjścia ALU Y MUX.
- a Pole [24] jest zmieniane w celu wyznaczenia rejestru R3 jako docelowego.
- Pole [25] jest zmieniane w celu wyznaczenia rejestru R2 jako jednego z rejestrów źródłowych.
- a Pole [26] jest zmieniane w celu wyznaczenia rejestru R1 jako jednego z rejestrów źródłowych.

- a Pole [27] jest zmieniane w celu określenia operacji ADD (dodawania) ALU. Rozkazem dla przesuwnika ALU jest PASS; wobec tego wartość z wyjścia ALU nie jest przesuwana przez przesuwnik.

Można tu podać kilka uwag na temat notacji symbolicznej. Nie jest konieczne określanie numeru pola w przypadku pól kolejnych. Znaczy to, że

CONT11 [17], WELH, [18], SELRFYMX

może być zapisane jako

CONT11 [17], WELH, SELRFYMX

ponieważ SELRFYMX jest w polu [18].

Rozkazy ALU z grupy 1 w tabeli 15.9 muszą zawsze być używane w połączeniu z grupą 2. Rozkazy ALU z grup 3÷5 nie mogą być używane z grupą 2.

TABELA 15.9. Pole rozkazu rejestrowej ALU TI 8832 (pole 27)

Grupa 1		Funkcja
ADD	H#01	$R + S + C_n$
SUBR	H#02	$(\text{NOT } R) + S + C_n$
SUBS	H#03	$R = (\text{NOT } S) + C_n$
INSC	H#04	$S + C_n$
INCNS	H#05	$(\text{NOT } S) + C_n$
INCR	H#06	$R + C_n$
INCNR	H#07	$(\text{NOT } R) + C_n$
XOR	H#09	$R \text{ XOR } S$
AND	H#0A	$R \text{ AND } S$
OR	H#0B	$R \text{ OR } S$
NAND	H#0C	$R \text{ NAND } S$
NOR	H#0D	$R \text{ NOR } S$
ANDNR	H#0E	$(\text{NOT } R) \text{ AND } S$
Grupa 2		Funkcja
SRA	H#00	Przesunięcie arytmetyczne w prawo o pojedynczej precyzji
SRAD	H#10	Przesunięcie arytmetyczne w prawo o podwójnej precyzji
SRL	H#20	Przesunięcie logiczne w prawo o pojedynczej precyzji
SRLD	H#30	Przesunięcie logiczne w prawo o podwójnej precyzji
SLA	H#40	Przesunięcie arytmetyczne w lewo o pojedynczej precyzji
SLAD	H#50	Przesunięcie arytmetyczne w lewo o podwójnej precyzji
SLC	H#60	Przesunięcie cykliczne w lewo o pojedynczej precyzji
SLCD	H#70	Przesunięcie cykliczne w lewo o podwójnej precyzji
SRC	H#80	Przesunięcie cykliczne w prawo o pojedynczej precyzji
SRCD	H#90	Przesunięcie cykliczne w prawo o podwójnej precyzji
MQSRA	H#A0	Arytmetyczne przesunięcie w prawo rejestru MQ
MQSRL	H#B0	Logiczne przesunięcie w prawo rejestru MQ
MQSL	H#C0	Logiczne przesunięcie w lewo rejestru MQ
MQSLC	H#D0	Cykliczne przesunięcie w lewo rejestru MQ
LOADMQ	H#E0	Ładuj rejestr MQ
PASS	H#F0	Przepuść ALU do Y (bez operacji przesuwania)

TABELA 15.9 cd.

Grupa 3		Funkcja
SET1	H#08	Ustaw bit 1
SET0	H#18	Ustaw bit 0
TB1	H#28	Zbadaj bit 1
TBO	H#38	Zbadaj bit 0
ABS	H#48	Wartość bezwzględna
SMTC	H#58	Znak - moduł / uzupełnienie do dwóch
ADDI	H#68	Dodaj natychmiastowy
SUBI	H#78	Odejmij natychmiastowy
BADD	H#88	Bajtowo dodaj R do S
BSUBS	H#98	Bajtowo odejmij S od R
BSUBR	H#A8	Bajtowo odejmij R od S
BINCS	H#B8	Bajtowo inkrementuj S
BINCNS	H#C8	Bajtowo inkrementuj ujemną S
BXOR	H#D8	Bajtowo wykonaj XOR na R i S
BAND	H#E8	Bajtowo wykonaj AND na R i S
BOR	H#F8	Bajtowo wykonaj OR na R i S
Grupa 4		Funkcja
CRC	H#00	Akumuluj cykliczny znak redundancji
SEL	H#10	Wybierz S lub R
SNORM	H#20	Normalizuj pojedynczą długość
DNORM	H#30	Normalizuj podwójną długość
DIVRF	H#40	Ustal resztę dzielenia
SDIVQF	H#50	Ustal resztę dzielenia ze znakiem
SMULI	H#60	Iteruj mnożenie ze znakiem
SMULT	H#70	Zakończ mnożenie ze znakiem
SDIVIN	H#80	Inicjuj dzielenie ze znakiem
SDIVIS	H#90	Rozpocznij dzielenie ze znakiem
SDIVI	H#A0	Iteruj dzielenie ze znakiem
UDIVIS	H#B0	Rozpocznij dzielenie bez znaku
UDIVI	H#C0	Iteruj dzielenie bez znaku
UMULI	H#D0	Iteruj mnożenie bez znaku
SDIVIT	H#E0	Zakończ dzielenie ze znakiem
UDIVIT	H#F0	Zakończ dzielenie bez znaku
Grupa 5		Funkcja
LOADFF	H#0F	Ładuj przerzutniki dzielenie/BCD
CLR	H#1F	Wyczyść
DUMPPF	H#5F	Wyjście z przerzutników dzielenie/BCD
BCDBIN	H#7F	Konwersja notacji BCD na binarną
EX3BC	H#8F	Korekta bajtu za pomocą kodu z nadmiarem 3
EX3C	H#9F	Korekta słowa za pomocą kodu z nadmiarem 3
SDIVO	H#AF	Badanie przepełnienia dzielenia ze znakiem
BINEX3	H#DF	Konwersja notacji binarnej na kod z nadmiarem 3
NOP32	H#FF	Brak operacji

ZASTOSOWANIA MIKROPROGRAMOWANIA

Od czasu wprowadzenia mikroprogramowania, a zwłaszcza od późnych lat sześćdziesiątych, zastosowania mikroprogramowania różnicowały się i upowszechniały. Już w roku 1971 można było odnotować większość (jeżeli nie wszystkie) współczesnych zastosowań mikroprogramowania [FLYN71, HAAV71]. Nowsze przeglądy obejmują w istocie ten sam zbiór zastosowań [RAVS80]. Do zbioru aktualnych zastosowań mikroprogramowania należą:

- a budowa komputerów,
- a emulacja,
- a wspieranie systemów operacyjnych,
- budowa urządzeń o specjalnym przeznaczeniu,
- a wspieranie języków wysokiego poziomu,
- Q mikrodiagnostyka,
- a dostosowywanie do potrzeb użytkownika.

Rozdział ten był poświęcony dyskusji na temat budowy komputerów. Rozwiązanie mikroprogramowane jest jedną z metod wdrażania jednostek sterujących. Spokrewnioną metodą jest *emulacja* [MALL75]. Emulacja odnosi się do użycia mikroprogramu w jednej maszynie do wykonywania programów napisanych pierwotnie dla innej. Najbardziej powszechnym zastosowaniem emulacji jest wspomaganie użytkowników przenoszących się od komputera do komputera. W ten sposób postępują często producenci, aby ułatwić swoim klientom wymianę starszych maszyn na nowsze, dzięki czemu szukanie innego producenta (dostawcy) jest dla nich nieatrakcyjne. Użytkownicy są często zaskoczeni, stwierdzając, jak długo to przejściowe narzędzie jest stosowane. Jeden z obserwatorów [MALL83] zauważył, że nawet w roku 1983 można było znaleźć System/370 IBM emulujący IBM 1401 zastąpiony fizycznie ponad 15 lat wcześniej!

Innym owocnym zastosowaniem mikroprogramowania jest *wspieranie systemów operacyjnych*. Mikroprogramy mogą być używane do implementacji funkcji pierwotnych, co zastępuje ważne części oprogramowania systemowego. Metoda ta może uprościć zadanie implementacji systemu operacyjnego i poprawić wydajność tego systemu.

Mikroprogramowanie jest również użytecznym narzędziem przy realizacji *urządzeń o specjalnym przeznaczeniu*, które mogą być wbudowane do komputera macierzystego. Dobrym tego przykładem jest moduł komunikacji danych (ang. *data Communications board*). Moduł ten zawiera własny mikroprocesor. Ponieważ jest używany do celów specjalnych, rozsądne jest wdrożenie niektórych jego funkcji w postaci oprogramowania układowego (ang. *firmware*) zamiast zwykłego (*software*), co pozwala na zwiększenie wydajności.

Wspieranie języków wysokiego poziomu jest jeszcze jednym owocnym obszarem zastosowań mikroprogramowania. Różne funkcje i rodzaje danych mogą

być zaimplementowane bezpośrednio w postaci oprogramowania układowego. W rezultacie łatwiej jest kompilować program na efektywną postać języka maszynowego. Dzięki temu język maszynowy jest dostosowywany do potrzeb języka wysokiego poziomu (jak FORTRAN, COBOL, Ada).

Mikroprogramowanie może być używane do wspierania monitorowania, wykrywania, izolowania i naprawiania błędów systemu. Możliwości te są znane jako *mikrodiagnostyka* i mogą znacznie ułatwić konserwację systemu. Rozwiązanie to umożliwia automatyczną rekonfigurację systemu po wykryciu uszkodzenia: jeśli na przykład niewłaściwie pracuje szybki układ mnożący, jego funkcję może przejąć układ mikroprogramowany.

Bardzo ogólną kategorią zastosowań jest *dostosowywanie do potrzeb użytkownika*. W wielu maszynach występuje *zapisywalna pamięć sterowania*, to znaczy pamięć sterowania zaimplementowana w postaci RAM zamiast ROM, co umożliwia użytkownikowi pisanie mikroprogramów. Na ogół dostarczana jest łatwa do użycia lista mikrorozkazów o formacie pionowym. Dzięki temu użytkownik może dostosować maszynę do określonego zastosowania.

15.6

POLECANA LITERATURA

Istnieje wiele książek poświęconych mikroprogramowaniu. Być może najbardziej obszerną jest [LYNC93]. W [SEGE91] przedstawiono podstawy mikroprogramowania i projektowania systemów mikroprogramowanych, posługując się przykładowym projektem prostego procesora 16-bitowego. W [ANDR80] i [CLIN81] wyczerpująco ujęto implementację jednostek sterujących. Do innych książek poświęconych temu tematowi oraz innym zastosowaniom mikroprogramowania należą [BANE82] i [AGRA76]. W [KRAF81] posłużono się mini-komputerem opracowanym przez AT&T jako przypadkiem do przestudiowania. W [PARK89] i [TI90] zawarto szczegółowy opis modułu rozwijania oprogramowania TI 8800.

AGRA76 Agrawala A., Rauscher T.: *Foundations of Microprogramming Architecture, Software, and Applications*. New York, Academic Press, 1976.

ANDR80 Andrews M.: *Principles of Firmware Engineering in Microprogram Control*. Silver Spring, Computer Science Press, 1980.

BANE82 Banerji D., Raymond J.: *Elements of Microprogramming*. Englewood Cliffs, Prentice-Hall, 1982.

CLIN81 Cline B.: *Microprogramming Concepts and Techniques*. New York, Petrocelli, 1981.

KRAF81 Kraff G., Toy W.: *Microprogrammed Control and Reliable Design of Small Computers*. Englewood Cliffs, Prentice-Hall, 1981.

LYNC93 Lynch M.: *Microprogrammed State Machine Design*. Boca Raton, 1993.

PARK89 Parker A., Hamblen J.: *An Introduction to Microprogramming with Exercises Designed for Texas Instruments SN74ACT8800 Software Development Board*. Dallas, Texas Instruments, 1989.

SEGE91 Segee B., Field J.: *Microprogramming and Computer Architecture*. New York, Wiley, 1991.

T190 Texas Instruments Inc.: *SN74ACT8800 Family Data Manual*. SCSS006C, 1990.

15.7

PROBLEMY DO ROZWIĄZANIA

- 15.1. Opisz realizację rozkazu mnożenia w hipotetycznej maszynie zaprojektowanej przez Wilkesa. Użyj opisu oraz sieci działań.
- 15.2. Załóż, że lista **mikrorozkazów** zawiera **mikrorozkaz** o następującej postaci symbolicznej:

$$I(AC_0 = 1) \text{ THEN } CAR \leftarrow (C_{0-6}) \text{ ELSE } CAR \leftarrow (CAR) + 1$$

gdzie AC_0 jest bitem znaku akumulatora, a C_{0-6} to pierwsze 7 bitów **mikrorozkazu**. Stosując ten mikrorozkaz, napisz **mikroprogram** wdrażający rozkaz maszynowy Branch Register Minus (BRM), który powoduje rozgałęzienie, jeśli AC jest ujemny. Załóż, że bity $C_1 + C_n$ **mikrorozkazu** określają równoległy zbiór **mikrooperacji**. Wyraż program symbolicznie.

- 15.3. Prosty procesor ma cztery główne fazy w cyklu rozkazu: pobierania, adresowania pośredniego, wykonywania i przerwania. Dwa 1-bitowe znaczniki czasu określają bieżącą fazę w odniesieniu do realizacji układowej.

- (a) Dlaczego potrzebne są te znaczniki stanu?
- (b) Dlaczego nie są one potrzebne w przypadku **mikroprogramowanej jednostki sterującej**?

- 15.4. Rozważ jednostkę sterującą z rys. 15.7. Załóż, że pamięć sterowania ma szerokość 24 bity. Część sterowania w formacie mikrorozkazu jest podzielona na 2 pola. Pole mikrooperacji złożone z 13 bitów określa **mikrooperację**, która ma być zrealizowana. Pole wyboru adresu określa warunek, oparty na znacznikach stanu, który powoduje rozgałęzienie mikrorozkazu. Znaczników stanu jest 8.

- (a) Ile bitów jest w polu wyboru adresu?
- (b) Ile bitów jest w polu adresu?
- (c) Jaki jest rozmiar pamięci sterowania?

- 15.5. Jak w warunkach z poprzedniego problemu może być realizowane rozgałęzienie bezwarunkowe? Jak można zapobiec rozgałęzieniu - to znaczy, opisz **mikrorozkaz**, który nie określa rozgałęziania ani warunkowego, ani bezwarunkowego.

- 15.6. Chcemy, żeby każdemu podprogramowi rozkazu maszynowego odpowiadało 8 słów sterowania. Kody operacji rozkazów maszynowych mają 5 bitów, a pamięć sterowania ma pojemność 1024 słów. Zaproponuj odwzorowanie rejestru rozkazu w rejestrze adresu sterowania.

- 15.7. Rozkaz maszynowy BPNZ powoduje rozgałęzienie, jeśli wartość w AC jest dodatnia i niezerowa. Adres rozgałęzienia może być wyrażony bezpośrednio lub pośrednio, zależnie od bitu pośredniości (I) w rozkazie. Zaproponuj podprogram oparty na organizacji przedstawionej na rys. 15.7 będący implementacją tego rozkazu.
- 15.8. Używany jest kodowany format **mikrorozkazu**. Pokaż, w jaki sposób 9-bitowe pole **mikrooperacji** może być podzielone na pola cząstkowe w celu określenia 49 różnych działań.
- 15.9. Procesor ma 16 rejestrów, ALU o 16 funkcjach logicznych i 16 arytmetycznych oraz przesuwnik o 8 operacjach, przy czym zespoły te są połączone za pomocą wewnętrznej magistrali. Zaprojektuj format mikrorozkazu określający różne **mikrooperacje** tego procesora.
- 15.10. Przeanalizuj każdy z mnemoników mikrosekwensera z tabeli 15.9. Wykaż, dlaczego każdy bit jest ustawiany we wskazany sposób w celu osiągnięcia pożądanego działania.

ORGANIZACJA RÓWNOLEGŁA

W ostatniej części książki przedstawimy organizację równoległą, będącą obszarem o wciąż wzrastającym znaczeniu. W rozdziale 16 omówimy dwa najwybitniejsze i najbardziej udane zastosowania organizacji równoległej: systemy wieloprocesorowe oraz organizację wektorową. W odniesieniu do systemów wieloprocesorowych w rozdziale tym zajmiemy się kluczowym problemem projektowym, jakim jest spójność pamięci podręcznej. Dokonamy również przeglądu ważnego obszaru zastosowań procesorów równoległych - użycia wielu procesorów o organizacji wieloprocesorowej. Organizacja ta powoduje powstanie nowych problemów, nie związanych z tradycyjnymi konfiguracjami wieloprocesorowymi, po prostu z powodu skali przedsięwzięcia, które może obejmować setki a nawet tysiące procesorów.

Przetwarzanie równoległe

Tradycyjnie komputer był postrzegany jako maszyna sekwencyjna. Większość komputerowych języków programowania wymaga od programisty określania algorytmów jako sekwencji rozkazów. Procesory realizują programy przez sekwencyjne wykonywanie rozkazów maszynowych, rozkaz po rozkazie. Każdy rozkaz jest wykonywany w postaci sekwencji operacji (pobieranie rozkazu, pobieranie argumentów, przeprowadzanie operacji, zapisanie wyników).

Taki obraz komputera nigdy nie był całkowicie prawdziwy. Na poziomie **mikrooperacji** wiele sygnałów sterujących jest generowanych w tym samym czasie. Potokowe przetwarzanie rozkazów, polegające przynajmniej na nakładaniu się operacji pobierania i wykonywania, jest stosowane od długiego czasu. Są to dwa przykłady równoległego realizowania funkcji.

W miarę ewoluowania techniki komputerowej i spadku kosztu sprzętu komputerowego, projektanci komputerów dostrzegali coraz więcej możliwości paralelizmu, mając na uwadze zwykle podniesienie wydajności, a czasem poprawę niezawodności. W rozdziale tym zapoznamy się z trzema najwybitniejszymi i najbardziej udanymi podejściami do organizacji równoległej. Najpierw przeanalizujemy wieloprocesorowość, jeden z najwcześniejszych i wciąż najbardziej powszechny przykład organizacji równoległej. Wieloprocesorowość polega zwykle na użyciu wielu procesorów dysponujących wspólną pamięcią. Następnie rozpatrzmy sprzętowe rozwiązania organizacyjne służące do obliczeń wektorowych. Rozwiązania te zmierzają do optymalizacji ALU pod kątem przetwarzania wektorów lub tablic liczb zmiennopozycyjnych. Używano ich do wdrażania klasy systemów znanej jako superkomputery. Na zakończenie przedstawimy bardziej ogólny obszar, określany jako organizacja procesorów równoległych.

16.1

PRZETWARZANIE WIELOPROCESOROWE

Użycie wielu procesorów jest motywowane względami wydajności i (lub) niezawodności. Klasyfikujemy takie systemy następująco:

- *Luźno powiązane systemy wieloprocesorowe.* System taki stanowi zbiór stosunkowo autonomicznych jednostek, przy czym każdy procesor ma własną pamięć główną i kanały wejścia-wyjścia. W tym kontekście jest często stosowany termin systemy wielokomputerowe.
- a *Procesory wyspecjalizowane funkcjonalnie.* Chodzi o jednostki takie, jak procesory wejścia-wyjścia. W tym przypadku istnieje główny procesor o ogólnym przeznaczeniu oraz wyspecjalizowane procesory, które są sterowane przez główny procesor i wykonują dla niego usługi.
- *Silnie powiązane systemy wieloprocesorowe.* System taki obejmuje zbiór procesorów o wspólnej pamięci głównej, działający pod zintegrowaną kontrolą systemu operacyjnego.
- *Systemy z przetwarzaniem równoległym.* Silnie powiązane systemy wieloprocesorowe, które mogą współpracować, pracując równoległe nad jednym zadaniem.

Ponieważ tematem tej książki jest organizacja i architektura pojedynczego systemu komputerowego, pierwsza kategoria na tej liście wykracza poza jej zakres; zainteresowani mogą sięgnąć do [STAL94]. Zetknęliśmy się już z koncepcją wyspecjalizowanych procesorów, głównie w rozdz. 6. Przetwarzanie równoległe jest głównie problemem projektowania oprogramowania i chociaż wykonywano i wykonuje się wiele badań, pozostaje ono poza stanem praktycznego stosowania [HABE85, FOX87, KARP87, OBER88]. Dlatego w tym podrozdziale skupimy się na silnie powiązanych systemach wieloprocesorowych.

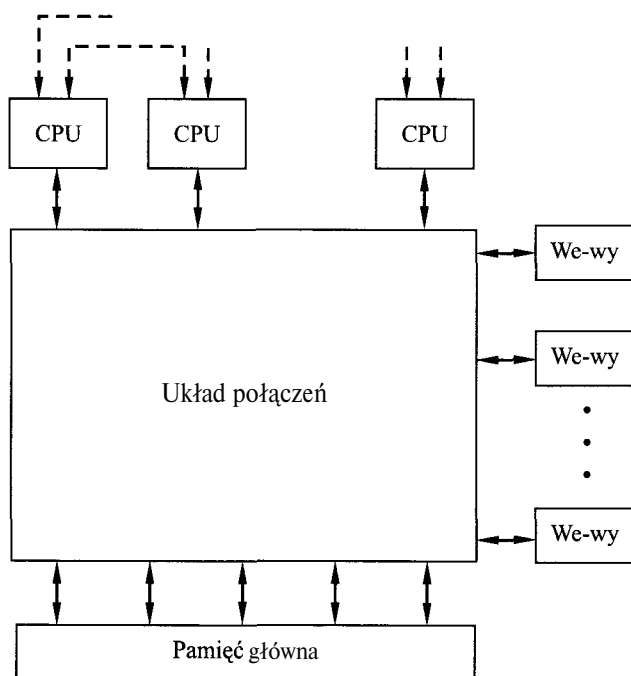
Rozpocznijmy od rozważenia samej definicji silnie powiązanego systemu wieloprocesorowego, który nazywamy **wieloprocesorem**. Zgodnie z [ENSL77], wieloprocesor można scharakteryzować następująco:

1. Obejmuje on dwa lub więcej podobnych procesorów o ogólnym przeznaczeniu i o porównywalnych możliwościach.
2. Wszystkie procesory mają dostęp do tej samej (wspólnej) pamięci globalnej. Mogą też być używane pamięci lokalne (własne).
3. Wszystkie procesory dzielą się dostępem do urządzeń wejścia-wyjścia za pośrednictwem tych samych kanałów, albo przez różne kanały umożliwiające dostęp do tych samych urządzeń.
4. System jest sterowany przez zintegrowany system operacyjny, zapewniający współpracę między procesorami i ich programami na poziomie zadania, pracy, pliku i elementów danych.

Punkty od 1 do 3 są zrozumiałe same przez się. Punkt 4 ilustruje jedną z różnic w stosunku do systemu luźno powiązanego. W przypadku tego ostatniego fizyczną jednostką współpracy jest zwykle kompletny plik. W wieloprocessorze pojedyncze elementy danych mogą stanowić poziom wzajemnego oddziaływania, a między procesami może istnieć współpraca.

Organizacja

Na rysunku 16.1 jest pokazany ogólny schemat organizacji systemu wieloprocessorowego. Występuje w nim dwa lub więcej procesorów. Każdy procesor stanowi zamkniętą całość, obejmując jednostkę sterującą, ALU, rejestry i w miarę możliwości pamięć podręczną. Każdy procesor ma dostęp do wspólnej pamięci głównej oraz do urządzeń wejścia-wyjścia za pośrednictwem pewnej formy mechanizmu łączenia. Procesory mogą porozumiewać się ze sobą poprzez pamięć (wiadomości i informacje o stanie pozostawiane we wspólnych obszarach danych). Mogą również wymieniać sygnały bezpośrednio, co zostało pokazane za pomocą linii przerywanych. Pamięć jest często zorganizowana w ten sposób, że możliwych jest wiele jednoczesnych dostępuów do oddzielnych bloków pamięci. W niektórych konfiguracjach każdy procesor może dysponować własną pamięcią główną i kanałami wejścia-wyjścia jako uzupełnieniem zasobów wspólnych.



RYSUNEK 16.1. Ogólny schemat blokowy silnie powiązanego systemu mikroprocesorowego

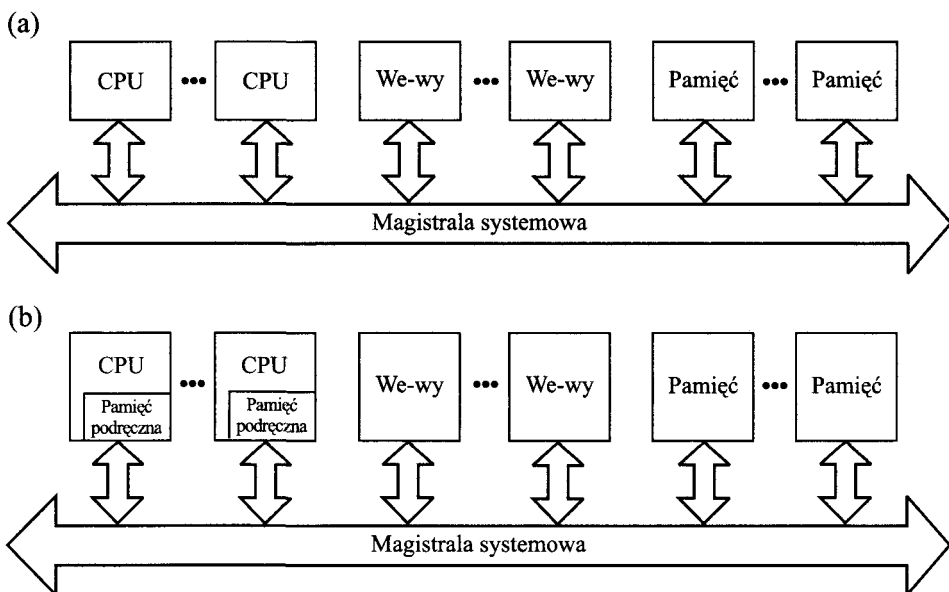
Organizacje systemów wieloprocesorowych można sklasyfikować jako systemy z:

- a magistralą z podziałem czasu lub ze wspólną magistralą,
- a pamięcią wieloportową,
- a centralną jednostką sterującą.

Magistrala z podziałem czasu

Magistrala z podziałem czasu jest najprostszym mechanizmem umożliwiającym zbudowanie systemu wieloprocesorowego (rys. 16.2a). Struktura i interfejsy są w zasadzie takie same, jak w przypadku systemu jednoprocessorowego używającego połączenia magistralowego. Magistrala składa się z linii sterowania, linii adresu i danych. Aby ułatwić transfery DMA* z procesorów wejścia-wyjścia, są przewidziane następujące rozwiązania:

- a *Adresowanie*. Musi istnieć możliwość rozróżniania modułów na magistrali w celu określenia źródła i miejsca docelowego danych.
- Q *Arbitraż*. Dowolny moduł wejścia-wyjścia może okresowo funkcjonować jako jednostka nadrzędna. Przewidziano mechanizm arbitrażu między jednostkami rywalizującymi o sterowanie magistralą, korzystający z pewnego rodzaju schematu priorytetów.



RYSUNEK 16.2. Magistrala z podziałem czasu: (a) organizacja podstawowa; (b) organizacja z pamięcią podręczną

*DMA oznacza dostęp bezpośredni do pamięci (Direct Memory Access) (*przyp.tłum.*).

- ❑ *Podział czasu.* Gdy jeden z modułów steruje magistralą, pozostałe są odłączone i muszą w razie konieczności zawiesić działanie, aż do uzyskania dostępu do magistrali.

Rozwiązania te są bezpośrednio użyteczne w konfiguracji wieloprocessorowej. W tym przypadku istnieje wiele jednostek centralnych i wiele procesorów wejścia-wyjścia, z których wszystkie dążą do uzyskania dostępu do jednego lub wielu modułów pamięci poprzez magistralę.

Organizacja magistralowa ma wiele zalet w porównaniu z innymi rozwiązaniami.

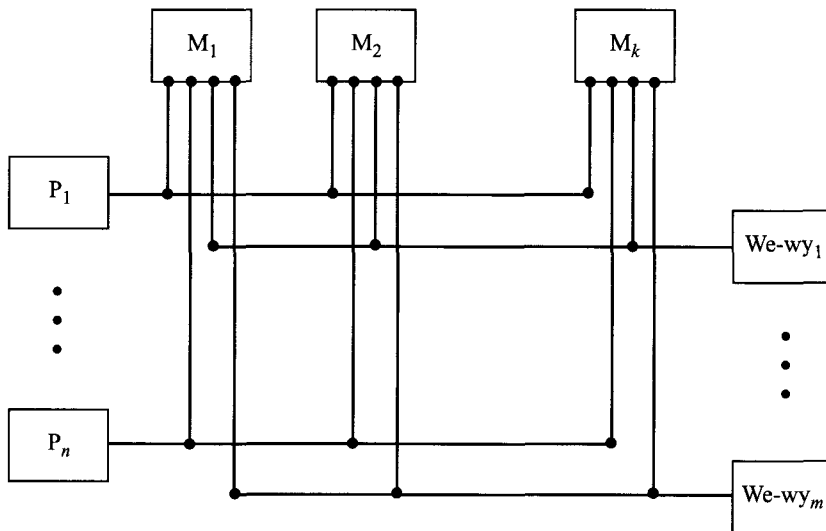
- a *Prostota.* Jest to najprostsze rozwiązanie organizacji wieloprocessorowej. Interfejs fizyczny, adresowanie, arbitraż i układy logiczne podziału czasu w każdym procesorze pozostają takie same, jak w systemie jednoprocessorowym.
- ❑ *Elastyczność.* Łatwo jest na ogół rozszerzyć system przez dołączenie następnych procesorów do magistrali.
- a *Niezawodność.* Magistrala jest w zasadzie środkiem pasywnym i uszkodzenie któregośkolwiek z dołączonych urządzeń nie powinno spowodować uszkodzenia całego systemu.

Główną niedogodnością organizacji magistralowej jest ograniczona wydajność. Wszystkie odniesienia do pamięci przechodzą przez wspólną magistralę, wobec tego szybkość systemu jest ograniczana przez czas cyklu. W celu poprawienia wydajności pożądanym jest wyposażenie każdego procesora w pamięć podręczną (rys. 16.2b). Powinno to drastycznie zredukować liczbę dostępów za pośrednictwem magistrali.

Użycie pamięci podręcznych powoduje pojawienie się nowych problemów projektowych. Ponieważ każda lokalna pamięć podręczna zawiera obraz części pamięci, zmiana słowa w jednej pamięci podręcznej spowodowałaby oczywiście unieważnienie słowa w innej pamięci podręcznej. Aby temu zapobiec, pozostałe procesory muszą być zaalarmowane, że ma miejsce aktualizacja. Zagadnieniu temu jest poświęcony podrozdz. 16.2.

Pamięć wieloportowa

Użycie pamięci wieloportowej umożliwia bezpośredni, niezależny dostęp każdego procesora i każdego modułu wejścia-wyjścia do modułów pamięci głównej (rys. 16.3). Do rozwiązywania konfliktów są wymagane układy logiczne związane z pamięcią. Często stosowaną metodą rozwiązywania konfliktów jest stałe przypisanie priorytetu każdemu portowi pamięci. Zwykle interfejs fizyczny i elektryczny każdego portu jest identyczny z tym, jaki występowałby w jednoportowym module pamięci. Potrzeba więc niewiele (lub wcale) modyfikacji procesora lub modułów wejścia-wyjścia, aby dostosować je do pamięci wieloportowej.



RYSUNEK 16.3. Pamięć wieloportowa

Rozwiązanie z pamięcią wieloportowa jest bardziej złożone niż magistralowe, wymaga bowiem dodania do systemu pamięci dość dużej liczby układów logicznych. Powinno jednak umożliwić uzyskanie większej wydajności, ponieważ każdy procesor ma własną ścieżkę do każdego modułu pamięci. Inną zaletą rozwiązania wieloportowego jest umożliwienie skonfigurowania części pamięci jako pamięci „własnych” jednego lub wielu procesorów i (lub) modułów wejścia-wyjścia. Własność ta pozwala na poprawę zabezpieczenia przed nieupoważnionym dostępem oraz na przechowywanie podprogramów regeneracji w obszarach pamięci, które nie mogą być modyfikowane przez inne procesory.

Zwróćmy uwagę na to, że do sterowania pamięciami podręcznymi powinna być używana metoda jednoczesnego zapisu, ponieważ nie ma innych środków alarmowania innych procesorów o aktualizacji pamięci.

Centralna jednostka sterująca

Centralna jednostka sterująca kieruje przepływem oddzielnych strumieni danych między niezależnymi modułami: procesorem, pamięcią i wejściem-wyjściem. Jednostka ta może buforować zapotrzebowania, dokonywać arbitrażu i realizować taktowanie. Może również przekazywać wiadomości dotyczące stanu i sterowania między procesorami, a także zawiadamiać o aktualizacji pamięci podręcznej.

Ponieważ wszystkie układy logiczne związane z koordynowaniem konfiguracji wieloprocessorowej znajdują się w centralnej jednostce sterującej, interfejsy wejścia-wyjścia i pamięci procesora pozostają w zasadzie niezmienione. Umożliwia to taką samą elastyczność i prostotę interfejsu, jak w przypadku rozwiązania magistralowego. Główną wadą tego rozwiązania jest złożoność jednostki sterującej, która jest potencjalnie wąskim gardłem wydajności.

Struktura oparta na centralnej jednostce sterującej jest stosunkowo rozpowszechniona w dużych systemach wieloprocessorowych, takich jak duże modele z rodziny S/370 IBM.

Wieloprocessorowe systemy operacyjne

W przypadku silnie powiązanego systemu wieloprocessorowego użytkownik widzi jeden system operacyjny sterujący zasobami systemu. Rzeczywiście, taka konfiguracja wygląda jak wieloprogramowy system jednoprocessorowy. W obu przypadkach może być jednocześnie aktywnych wiele zadań i procesów, a rolą systemu operacyjnego jest planowanie ich wykonywania oraz przydzielanie zasobów.

Enslo [ENSL77] identyfikuje następujących siedem funkcji wieloprocessorowego systemu operacyjnego:

- a przydzielanie zasobów i zarządzanie nimi,
- ochrona tablic i danych,
- zapobieganie zakleszczeniom systemu,
- a zakończenie awaryjne,
- równoważenie obciążenia wejścia-wyjścia,
- a równoważenie obciążenia procesora,
- a rekonfiguracja.

Tylko trzy ostatnie funkcje są unikatowe lub różnią się istotnie dla systemów wieloprocessorowych w porównaniu z jednoprocessorowymi systemami wieloprogramowymi. Sprawą wieloprocessorowego systemu operacyjnego jest kontrolowanie efektywnego używania zasobów; w przeciwnym razie inwestowanie w system wieloprocessorowy nie ma sensu. Musi on także rekonfigurować system w przypadku uszkodzenia procesora, aby było możliwe kontynuowanie pracy przy obniżonej wydajności.

Silnie powiązane wieloprocessory mogą być charakteryzowane przez sposób planowania zadań. Istnieją dwa podejścia do funkcji planowania, określające:

- czy procesy są przypisywane procesorom,
- jak procesy są planowane w procesorach.

Jeśli dla każdego procesora jest utrzymywana kolejka krótkookresowa (patrz rys. 7.14), to proces po aktywowaniu jest na stałe przypisany do jednego procesora aż do zakończenia. W tym przypadku jeden procesor może pozostawać bez zajęcia i mieć pustą kolejkę, podczas gdy inny procesor ma zaległości. Aby zapobiec takiej sytuacji, można zastosować wspólną kolejkę. Wszystkie procesy są kierowane do jednej kolejki, po czym trafiają do dowolnego dostępnego procesora. W ten sposób w czasie życia zadania może ono być w różnych okresach wykonywane przez różne procesory.

Niezależnie od tego, czy procesy są na stałe związane z procesorami, pewne środki przypisywania procesów procesorom są potrzebne. Używane są w tym

celu dwa rozwiązania: typu nadrzędny/podległy oraz równoprawne. W przypadku architektury typu *nadrzędny/podległy* system operacyjny zawsze jest uruchomiony na jednym, konkretnym procesorze. Pozostałe procesory mogą tylko wykonywać programy użytkowe. Procesor nadrzędny jest odpowiedzialny za planowanie zadań. Gdy proces jest aktywny i procesor podległy **potrzebuje** obsługi (np. wywołuje wejście-wyjście), musi wysłać zgłoszenie do procesora nadrzędnego i czekać na tę obsługę. Rozwiązanie takie jest całkiem proste i wymaga nieznacznych uzupełnień w stosunku do **jednoprocesorowego** wieloprogramowego systemu operacyjnego. Rozstrzyganie konfliktów jest uproszczone, ponieważ jeden procesor kontroluje wszystkie zasoby pamięci oraz wejście-wyjście. Rozwiązanie to ma dwie wady: (1) uszkodzenie procesora nadrzędnego unieruchamia cały system, (2) procesor nadrzędny może się stać wąskim gardłem wydajności.

W przypadku architektury *równoprawnej* system operacyjny może działać na dowolnym procesorze, a każdy procesor realizuje własne planowanie na podstawie zbioru dostępnych procesów. Rozwiązanie to komplikuje system operacyjny. Musi on zapewnić, żeby dwa procesory nie wybrały tego samego procesu i żeby procesy nie ginęły z kolejki. Muszą być wprowadzone metody synchronizowania i rozstrzygania rywalizujących zapotrzebowań na zasoby.

Wieloprzetwarzanie w systemie 370 IBM

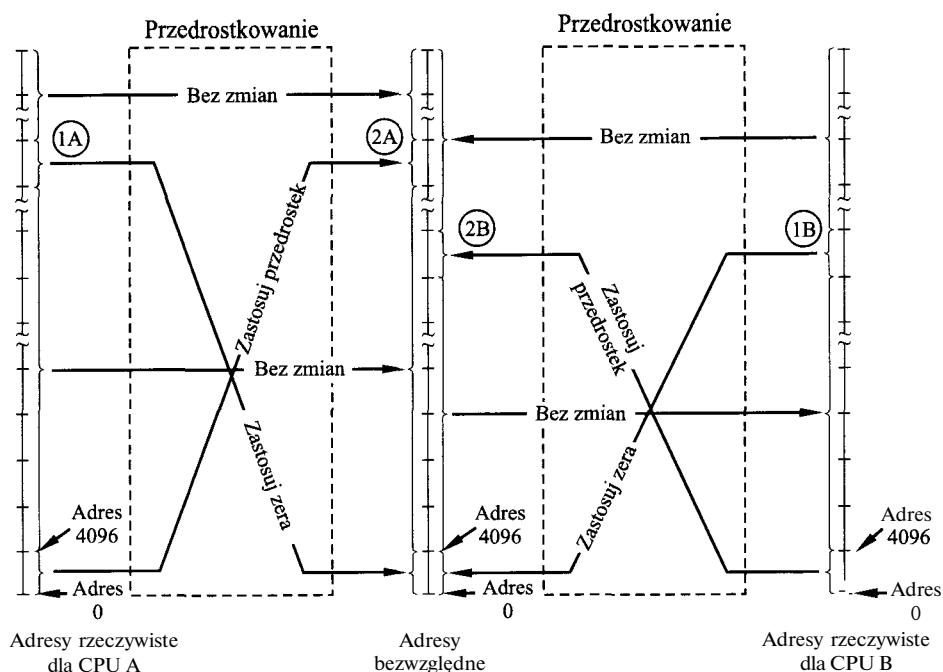
W architekturze systemu 370 przewidziano możliwość wspólnego używania pamięci głównej przez wiele procesorów oraz komunikowanie się wielu procesorów w celu koordynowania działań. Procesory używają wspólnie jednej kopii systemu operacyjnego, a ich obciążenie pracą jest dynamicznie równoważone. Każdy procesor ma własny zestaw kanałów **wejścia-wyjścia**, jednak wszystkie procesory mają dostęp do wszystkich urządzeń **wejścia-wyjścia**. Jeśli jeden procesor wymaga dostępu do urządzenia dołączonego do innego procesora, prosi go o wykonanie dla niego tej operacji.

Następujące podstawowe rozwiązania umożliwiają wieloprzetwarzanie w systemie 370:

- ❑ *Stosowanie przedrostków.* Służy wspólnemu użytkowaniu krytycznych obszarów pamięci głównej.
- ❑ *Sygnalizacja.* Służy do komunikacji między procesorami.
- ❑ *Synchronizacja.* Służy do koordynowania potencjalnie konfliktowych operacji procesora.

Stosowanie przedrostków

Pewne, specjalnie do tego celu przeznaczone lokacje w pamięci głównej są używane do przechowywania informacji sterowania i stanu. W systemie z jednym procesorem do tego celu są zarezerwowane pierwsze 4 KB pamięci. Taka



RYSUNEK 16.4. Zależność między adresami rzeczywistymi a bezwzględnymi

organizacja nie może oczywiście działać w konfiguracji wieloprocesorowej, ponieważ wszystkie procesory dążyłyby do używania tych samych lokacji.

W celu rozwiązania tego problemu rozróżnia się **przechowywanie rzeczywiste** od **przechowywania bezwzględnego** (ang. *absolute storage*). Przechowywanie rzeczywiste wiąże się z odniesieniami do pamięci dokonywanymi przez procesor. W konfiguracji wieloprocesorowej adresy rzeczywiste są układowo tłumaczone na adresy bezwzględne. Wynik tych operacji widać na rys. 16.4.

Każdy procesor jest wyposażony w **rejestr wartości przedrostka** (PVR), zawierający adres początkowy P bloku pamięci o pojemności 4 KB. Wobec tego 12 najmłodszych bitów rejestru PVR stanowią zawsze zera. Gdy procesor generuje adres rzeczywisty R , adres bezwzględny A jest tworzony z wykorzystaniem następujących reguł:

1. Jeśli $R < 4\text{ K}$, to $A = R + P$
2. Jeśli $P < R < P + 4\text{ K}$, to $A = R - P$
3. W innych przypadkach $A = R$

W wyniku tego odniesienia do pierwszych 4 KB pamięci ulegają translacji do innego, wstępnie określonego bloku pamięci o pojemności 4 KB. Każdy procesor otrzymuje do użytku oddzielny blok 4 KB. Algorytm ten odwzorowuje również rzeczywiste odniesienia do tych wyspecjalizowanych obszarów na odniesienia bezwzględne do pierwszego, 4-kilobajtowego bloku pamięci. Pozwala to na używanie pierwszego bloku do koordynowania działań wszystkich proce-

sorów oraz na przypisanie każdemu procesorowi indywidualnego bloku, służącego do sterowania jego własnymi działaniami.

Dodawanie przedrostka jest realizowane po przetworzeniu adresu wirtualnego. Sekwencja zdarzeń jest następująca:

1. Adres wirtualny jest przetwarzany na rzeczywisty za pomocą dynamicznej translacji adresu, wyjaśnionej w podrozdz. 7.3.
2. Adres rzeczywisty jest przetwarzany na adres bezwzględny przez dodanie przedrostka.

Sygnalizacja

Komunikacja między procesorami jest zapewniana za pomocą rozkazu procesora sygnałowego (ang. *Signal Processor Instruction* - SIGP). Ten rozkaz maszynowy ma trzy argumenty: adres procesora, kod rozkazu oraz stan. Adres procesora identyfikuje procesor, do którego jest wysyłany sygnał. Kod rozkazu określa wymagane działanie, a argument stanu określa rejestr, w którym są przechowywane informacje o stanie lub wynik rozkazu SIGP.

W tabeli 16.1 są wymienione niektóre kody rozkazu używane w rozkazie SIGP. Gdy w jednym procesorze jest wykonywany rozkaz SIGP, 2-bitowy kod warunkowy w rejestrze PSW pokazuje wynik operacji:

- a kod rozkazu akceptowany,
- informacja o stanie zapisana,
- a zajęta,
- a niedziałająca.

Pierwsze dwa warunki wskazują, że sygnał został skutecznie przekazany do innego procesora. Niektóre rozkazy wymagają zwrotnej informacji o stanie, inne zaś nie wymagają. Warunek zajętości wskazuje, że ścieżka dostępu do innego procesora jest zajęta lub że inny procesor, chociaż działa, nie może odpowiedzieć na kod rozkazu. Inny procesor może być też niedziałający.

TABELA 16.1. Rozkazy międzyprocesorowe IBM S/370

SENSE

Adresowany procesor wysyła informację o stanie do procesora wydającego wyniki.

START

Adresowany procesor rozpoczyna stan działania, jeśli znajduje się w stanie zatrzymania.

STOP

Adresowany procesor rozpoczyna stan zatrzymania, jeśli znajduje się w stanie działania.

STOP AND STORE STATUS

Adresowany procesor rozpoczyna stan zatrzymania, a jego stan jest zapisywany w lokacjach bezwzględnych od 216 do 512. Stan obejmuje PSW oraz rejestry robocze, sterujące i zmienno-pozycyjne.

EXTERNAL CALL

Przerwanie zewnętrzne jest wysyłane przez jeden procesor do drugiego jako zapotrzebowanie na obsługę.

Synchronizacja

W architekturze systemu S/370 przewidziano szereg mechanizmów ułatwiających współpracę między procesami i procesorami.

Jednym ze środków współpracy jest używanie przez wiele procesów tego samego, wspólnego obszaru pamięci. Obszar ten może być wykorzystywany do wymiany informacji o stanie. Potrzebne są konwencje umożliwiające zapobieganie konfliktom podczas odczytywania i aktualizowania wspólnych obszarów. W tym celu system S/370 posługuje się pewnym rodzajem blokady programowej. Jedno słowo w pamięci wspólnej jest wyznaczone jako blokada. Każdy procesor sprawdza słowo blokujące przed próbą dostępu do wspólnych obszarów danych. Jeśli słowo to jest 0, to procesor zapisuje swój identyfikator w tym słowie, po czym inne procesory muszą czekać, aż pierwszy procesor zakończy operację i sprowadzi słowo blokujące do stanu wyjściowego.

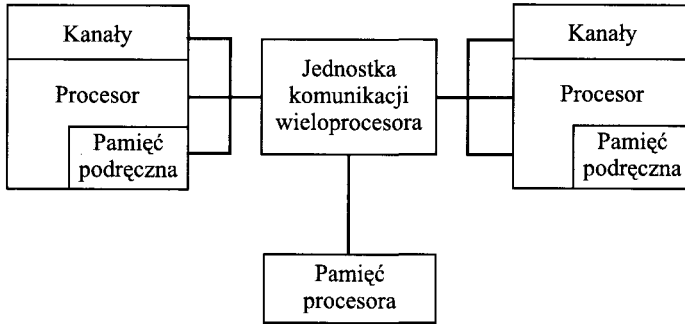
Aby ta konwencja funkcjonowała, musi być możliwe badanie i aktualizowanie słowa blokującego bez możliwości, żeby inny proces uzyskał dostęp do tego obszaru pamięci między operacjami badania i aktualizacji. Do tego celu jest używany rozkaz maszynowy porównania i wymiany (ang. *Compare and Swap* - CS). Rozkaz CS ma 3 argumenty. Argumenty pierwszy i drugi są porównywane. Jeśli są one równe, trzeci argument jest zapisywany w lokacji pierwszego argumentu. Jeśli nie są równe, to do lokacji pierwszego argumentu jest ładowany drugi argument. Aby osiągnąć pożądany rezultat, pierwszy, drugi i trzeci argument powinny być odpowiednio równe: 0, słowu blokującemu oraz identyfikatorowi procesora. Podczas wykonywania rozkazu CS żaden inny procesor nie może sięgać do określonej lokacji (drugiego argumentu).

W architekturze tej przewidziano również mechanizm przerywania oraz rozkazy do ustawiania i synchronizowania zegarów czasu dnia (TOD) we wszystkich procesorach.

Organizacja wieloprocessora IBM 3033

Dotychczasowe rozważania skupiały się na architekturze wieloprzetwarzania w rodzinie S/370. Teraz zapoznamy się pokrótce z organizacją wieloprzetwarzania w konkretnej maszynie, a mianowicie IBM 3033 [CONN79].

IBM 3033 może być skonfigurowany w postaci dwóch silnie powiązanych procesorów. Każdy procesor ma własny zestaw kanałów, które mogą być połączone od uszkodzonego do funkcjonującego procesora. Oba procesory są połączone za pomocą specjalizowanego modułu sprzętowego określanego jako **jednostka komunikacji wieloprocessora** (ang. *Multiprocessor Communication Unit* - MCU), co widać na rys. 16.5. MCU może się komunikować z każdym procesorem oraz bezpośrednio z kanałami i pamięciami podręcznymi każdego procesora. MCU wykonuje następujące funkcje w odniesieniu do obu procesorów:



RYSUNEK 16.5. Organizacja wieloprocessora IBM 3033

- a przedrostkowanie,
- a komunikacja międzyprocesorowa,
- dostęp do pamięci,
- a rozgłaszanie aktualizacji pamięci.

Pierwsze dwie funkcje już opisaliśmy. MCU steruje dostępem do pamięci w celu skoordynowania go oraz rozwiązywania konfliktów między dwoma procesorami dotyczących dostępu do pamięci. Zgłoszenia są buforowane w MCU i w razie braku konfliktu realizowane na zasadzie pierwszy przyszedł - pierwszy obsłużony. Jeśli oba procesory próbują uzyskać dostęp do tego samego bloku pamięci, to do rozwiązywania konfliktu jest używany prosty mechanizm priorytetu.

IBM 3033 używa metody jednoczesnego zapisu w odniesieniu do swoich pamięci podręcznych. Oznacza to, że zapis powoduje aktualizację odpowiedniego słowa zarówno w pamięci podręcznej, jak i w głównej. Ponieważ każdy procesor ma własną pamięć podręczną, konieczna jest aktualizacja obu pamięci podręcznych. Gdy ma miejsce zapis, MCU automatycznie rozgłasza to drugiemu procesorowi i jeśli pamięć podręczna tego procesora zawiera zmieniane dane, to są one automatycznie unieważniane. Przesyłanie danych przez kanały wejścia-wyjścia jest traktowane podobnie.

16.2

SPÓJNOŚĆ PAMIĘCI PODRĘCZNYCH I PROTOKÓŁ MESI

We współczesnych systemach wieloprocessorowych z każdym procesorem jest zwykle związana jedno- lub dwupoziomowa pamięć podręczna. Organizacja taka ma zasadnicze znaczenie dla uzyskania rozsądnej wydajności. Powoduje ona jednak powstanie problemu znanego jako **problem spójności pamięci podręcznych** (ang. *cache coherence*). Istota tego problemu jest następująca: wiele kopii

tych samych danych może jednocześnie występować w różnych pamięciach podręcznych i jeśli procesory mogą swobodnie aktualizować własne kopie, to w wyniku może powstać niespójny obraz pamięci.

W tym podrozdziale dokonamy krótkiego przeglądu różnych rozwiązań problemu spójności pamięci podręcznych, a następnie skupimy się na najszerzej stosowanym rozwiązaniu: na protokole MESI. Pewna wersja tego protokołu jest używana zarówno w procesorze Pentium, jak i w PowerPC.

Rozwiązania problemu spójności pamięci podręcznych są ogólnie dzielone na programowe i sprzętowe. W niektórych implementacjach przyjęto strategię obejmującą elementy zarówno programowe, jak i sprzętowe. Mimo to klasyfikacja na rozwiązania programowe i sprzętowe jest nadal pouczająca i powszechnie używana przy rozpatrywaniu strategii zapewniania spójności.

Rozwiązania programowe

W programowych rozwiązaniach spójności pamięci podręcznych próbuje się uniknąć dodatkowych elementów sprzętowych, polegając na kompilatorze i systemie operacyjnym. Rozwiązania programowe są atrakcyjne, ponieważ ciężar wykrywania potencjalnych problemów jest przenoszony z okresu użytkowania na okres kompilowania, a złożoność projektowania jest przenoszona ze sprzętu na oprogramowanie. Jednak rozwiązania programowe z okresu kompilowania na ogół polegają na decyzjach konserwatywnych, co prowadzi do nieefektywnego wykorzystywania pamięci podręcznych.

Mechanizmy spójności oparte na kompilatorze polegają na analizowaniu programu w celu stwierdzenia, które dane mogą stanowić niebezpieczeństwo przy wprowadzaniu ich do pamięci podręcznych, oraz na odpowiednim znakovaniu tych danych. Następnie system operacyjny lub sprzęt zapobiegają kierowaniu tych danych do pamięci podręcznych.

Najprostszym rozwiązaniem jest zapobieganie temu, żeby jakiegokolwiek wspólne zmienne były kierowane do pamięci podręcznych. Jest to jednak rozwiązanie zbyt konserwatywne, ponieważ wspólne struktury danych mogą być w pewnych okresach używane na zasadzie wyłączości, natomiast w pozostałych okresach mogą być efektywnie odczytywane. Spójność pamięci podręcznych stanowi problem tylko wówczas, gdy przynajmniej jeden proces może aktualizować zmienną i przynajmniej jeden inny proces może sięgać po tę zmienną.

W bardziej efektywnych rozwiązaniach program jest analizowany w celu określenia okresów bezpiecznych dla wspólnych zmiennych. Następnie kompilator umieszcza w generowanym programie rozkaz, który wymusza spójność pamięci podręcznych podczas okresów krytycznych. Opracowano wiele metod przeprowadzania takiej analizy i wymuszania spójności; opisano je w [LILJ93] i w [STEN90].

Rozwiązania sprzętowe

Rozwiązania sprzętowe są na ogół określane jako protokoły spójności pamięci podręcznych. Polegają one na dynamicznym rozpoznawaniu warunków potencjalnej niespójności w czasie użytkowania systemu. Ponieważ problem jest rozwiązywany tylko wtedy, gdy rzeczywiście powstaje, wykorzystanie pamięci podręcznych jest bardziej efektywne, co prowadzi do poprawy wydajności w stosunku do rozwiązań programowych. Ponadto rozwiązania te są widzialne zarówno dla programisty, jak i dla kompilatora, co redukuje ciężar opracowywania oprogramowania.

Rozwiązania sprzętowe różnią się wieloma szczegółami: gdzie jest przechowywana informacja o stanie bloków danych, jak ta informacja jest zorganizowana, gdzie jest wymuszana spójność i jakie są mechanizmy wymuszania. Rozwiązania sprzętowe mogą być podzielone na dwie ogólne kategorie: protokoły katalogowe i protokoły podglądania.

Protokoły katalogowe

Protokoły katalogowe polegają na zbieraniu i przechowywaniu informacji o tym, gdzie rezydują kopie bloków danych. Zwykle istnieje centralny sterownik będący częścią sterownika pamięci głównej, a katalog jest przechowywany w pamięci głównej. Katalog zawiera globalną informację o stanie zawartości różnych lokalnych pamięci podręcznych. Gdy sterownik określonej pamięci podręcznej zgłasza zapotrzebowanie, sterownik centralny sprawdza i wydaje niezbędne rozkazy zapewniające przesyłanie danych między pamięcią główną a pamięciami podręcznymi lub między pamięciami podręcznymi. Jest on także odpowiedzialny za utrzymywanie i ciągłe aktualizowanie informacji o stanie; informacja o każdym działaniu lokalnym, które może wpłynąć na globalny stan bloku danych, musi trafić do sterownika centralnego.

Sterownik zachowuje zwykle informację o tym, które procesory mają kopię których bloków. Zanim procesor będzie mógł dokonać operacji zapisu w lokalnej kopii bloku, musi zgłosić do sterownika centralnego zapotrzebowanie na wyłączny dostęp do tego bloku. Przed udzieleniem zgody na wyłączny dostęp, sterownik wysyła wiadomość do wszystkich procesorów przechowujących w pamięci podręcznej kopię tego bloku, zmuszając każdy procesor do unieważnienia kopii. Po otrzymaniu potwierdzeń unieważnienia od każdego procesora, sterownik udziela prawa wyłącznego dostępu procesorowi zgłaszającemu zapotrzebowanie. Gdy inny procesor próbuje odczytać blok udostępniony na zasadzie wyłączności innemu procesorowi, wysyła zawiadomienie o chybieniu do sterownika. Sterownik wydaje wtedy procesorowi dysponującemu blokiem rozkaz zapisu do pamięci głównej. Teraz blok może być jednocześnie odczytywany przez oba procesory.

Wadą rozwiązań katalogowych jest wąskie gardło, jakie może wystąpić w centralnym sterowaniu, oraz obciążenie komunikacją między różnymi sterownikami pamięci podręcznych a sterownikiem centralnym. Są one jednak efektywne w dużych systemach o wielu magistralach lub z innym złożonym układem połączeń.

Protokoły podglądania

W protokołach podglądania odpowiedzialność za utrzymywanie spójności pamięci podręcznych jest rozkładana na wszystkie sterowniki tych pamięci w wieloprocesorze. Pamięć podręczna musi rozpoznać sytuację, w której przechowywany przez nią blok występuje również w innych pamięciach podręcznych. Gdy aktualizowany jest blok wspólny, muszą być o tym powiadomione pozostałe pamięci podręczne za pośrednictwem pewnego mechanizmu rozgłaszania. Każdy sterownik pamięci podręcznej może przeglądać („podgląda”) sieć w poszukiwaniu takich powiadomień i odpowiednio reagować.

Protokoły podglądania są idealnie dostosowane do wieloprocesorów o organizacji magistralowej, ponieważ wspólna magistrala stanowi idealny ośrodek rozgłaszania i „podglądania”. Ponieważ jednak jednym z celów stosowania lokalnych pamięci podręcznych jest unikanie dostępu do magistrali, trzeba zadbać o to, żeby zwiększony ruch na magistrali wywołany przez rozgłaszanie i „podglądanie” nie skompensował korzyści z używania lokalnych pamięci podręcznych.

Zbadane zostały dwa podstawowe rozwiązania protokołów podglądania: zapis z unieważnieniem i zapis z aktualizacją (lub rozgłaszanie zapisu). W przypadku **zapisu z unieważnieniem** w danej chwili może występować wiele jednostek odczytujących, ale tylko jedna zapisująca. Początkowo, do celów odczytu blok może być wspólny dla kilku pamięci podręcznych. Gdy jedna z nich chce dokonać zapisu w bloku, wydaje najpierw powiadomienie, że unieważnia ten blok w innych pamięciach podręcznych, co umożliwia zapis na zasadzie wyłączości. Dysponując blokiem na zasadzie wyłączości, procesor może dokonywać w nim lokalnych zapisów aż do chwili, w której inny procesor zgłosi zapotrzebowanie na ten blok.

W przypadku protokołu **zapisu z aktualizacją** może występować zarówno wiele jednostek zapisujących, jak i wiele odczytujących. Gdy procesor chce zaktualizować wspólny blok, słowo podlegające aktualizacji jest rozprowadzane do wszystkich innych, dzięki czemu pamięci podręczne zawierające ten blok mogą go zaktualizować.

Żadne z tych dwóch rozwiązań nie jest lepsze od drugiego w każdych warunkach. Wydajność zależy od liczby lokalnych pamięci podręcznych oraz od przebiegu odczytów i zapisów w pamięci. W niektórych systemach jest wdrożony protokół adaptacyjny, w którym stosuje się zarówno zapis z unieważnieniem, jak i zapis z aktualizacją.

Protokół unieważniania zapisu jest najszerzej używany w komercyjnych systemach wieloprocesorowych, takich jak Pentium i PowerPC. W systemach tych znakuje się stan każdego wiersza pamięci podręcznej (używając 2 dodatkowych bitów we wskaźniku pamięci podręcznej) jako: zmodyfikowany, wyłączny, wspólny lub nieważny. Z tego powodu protokół zapisu z unieważnieniem jest nazywany MESI (ang. *modified, exclusive, shared, invalid*). Zetknęliśmy się już z protokołem MESI w rozdziele 4, zajmując się koordynacją między poziomem 1 a poziomem 2 lokalnych pamięci podręcznych. W pozostałej części tego podrozdziału zapoznamy się z jego używaniem w lokalnych pamięciach podręcznych wieloprocesora. W celu uproszczenia prezentacji nie będziemy analizować mechanizmów lokalnej koordynacji między poziomem 1 a poziomem 2, ani też koordynacji w rozproszonym wieloprocesorze. Nie wprowadziłoby to żadnych nowych zasad, natomiast znacznie skomplikowałoby dyskusję.

Protokół MESI

Z rozdziału 4 pamiętamy, że w przypadku protokołu MESI pamięć podręczna danych zawiera w swoim wskaźniku 2 bity stanu; każdy wiersz może znajdować się w jednym z czterech stanów:

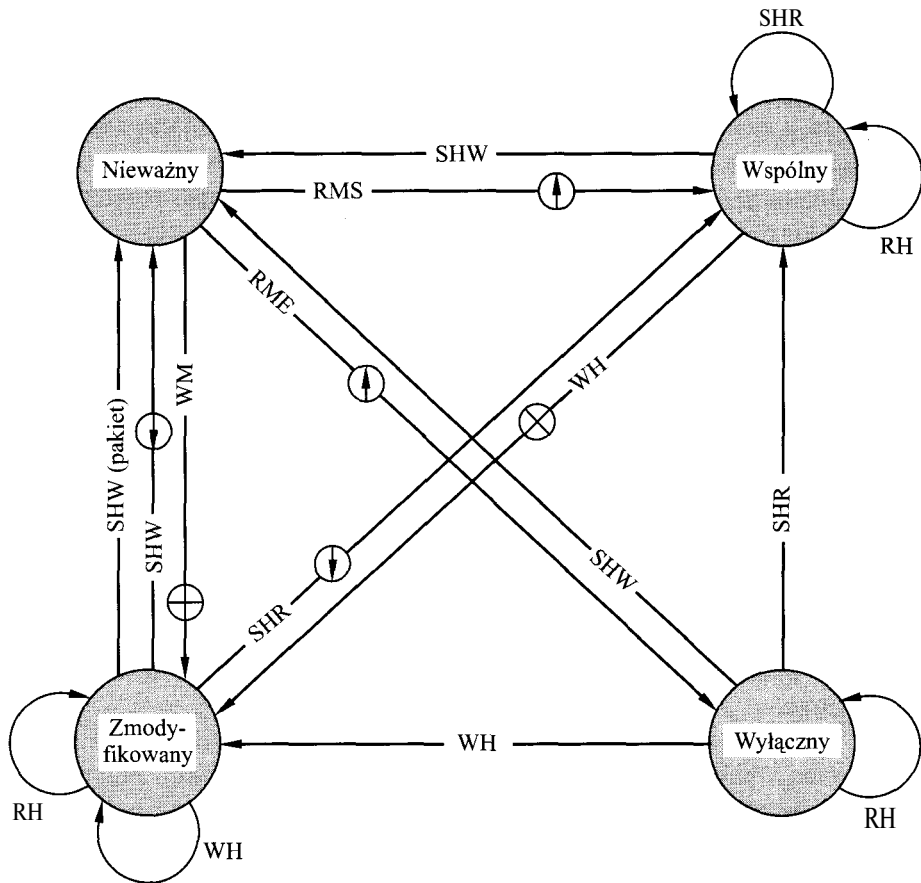
- a *Zmodyfikowany*. Wiersz pamięci podręcznej został zmodyfikowany (różni się od odpowiednika w pamięci głównej) i jest osiągalny tylko w tej pamięci podręcznej.
- a *Wyłączny*. Wiersz pamięci podręcznej jest taki sam, jak w pamięci głównej i nie występuje w innych pamięciach podręcznych.
- Q *Wspólny*. Wiersz w pamięci podręcznej jest taki sam, jak w pamięci głównej i może występować w innej pamięci podręcznej.
- *Nieważny*. Wiersz pamięci podręcznej nie zawiera danych ważnych.

Na rysunku 16.6 jest pokazany wykres stanów dla protokołu MESI. W dalszym ciągu opiszemy kolejno występujące w nim przejścia.

Chybiecie odczytu

Gdy w lokalnej pamięci podręcznej następuje chybiecie odczytu, procesor inicjuje odczyt w pamięci w celu odczytania bloku w pamięci głównej zawierającego brakujący adres. Procesor umieszcza sygnał na magistrali alarmujący wszystkie pozostałe jednostki procesor/pamięć podręczna, żeby śledziły transakcję. Wynikają stąd następujące możliwe sytuacje:

- a Jeśli jeden z pozostałych procesorów ma czystą (niezmodyfikowaną od czasu wczytania z pamięci) kopię bloku w stanie wyłącznym, to sygnalizuje w odpowiedzi, że dysponuje tym blokiem. Odpowiadający procesor zmienia następnie stan swojej kopii z wyłącznego na wspólny, a procesor inicjujący wczytuje blok i zmienia stan bloku w swojej pamięci podręcznej z nieważnego na wspólny.



RH - trafienie odczytu
 RMS - chybiecie odczytu, wspólny
 RME - chybiecie odczytu, wyłączny
 WH - trafienie zapisu
 WM - chybiecie zapisu
 SRH - śledź trafienie odczytu
 SHW - śledź trafienie zapisu lub odczytu
 z zamiarem modyfikacji

⊥ Kopiowanie zmienionego wiersza
 ⊗ Unieważnienie transakcji
 ⊕ Odczyt z zamiarem modyfikacji
 (⊥) Wypełnienie wiersza pamięci podręcznej

RYSUNEK 16.6. Graf przejść MESI

- Q Jeśli jeden lub wiele procesorów ma czystą kopię bloku w stanie wspólnym, to każdy z nich sygnalizuje ten stan. Procesor inicjujący wczytuje blok i zmienia stan bloku w swojej pamięci podręcznej z nieważnego na wspólny.
- Jeśli jeden z pozostałych procesorów ma zmodyfikowaną kopię bloku, to sygnalizuje procesorowi inicjującemu celowość ponownej próby. W tym samym czasie procesor dysponujący zmodyfikowaną kopią przejmuje sterowanie magistralą, zapisuje zmodyfikowany blok w pamięci głównej i zmienia stan tego bloku w swojej pamięci podręcznej ze zmodyfikowanego na

- wspólny. Następnie procesor inicjujący dokonuje ponownej próby i stwierdza, że jeden lub wiele procesorów ma czystą kopię bloku w stanie wspólnym, o czym była mowa w poprzednim punkcie.
- a Jeśli żadna inna pamięć podręczna nie ma kopii bloku (czystej ani zmodyfikowanej), to w odpowiedzi nie są przekazywane żadne sygnały. Procesor inicjujący wczytuje blok i zmienia stan bloku w swojej pamięci podręcznej z nieważnego na wyłączny.

Trafienie odczytu

Gdy następuje trafienie odczytu bloku w lokalnej pamięci podręcznej, procesor po prostu odczytuje wymagane dane; nie następuje zmiana stanu.

Chybiecie zapisu

Gdy następuje chybiecie zapisu w lokalnej pamięci podręcznej, procesor inicjuje odczyt pamięci w celu wczytania z pamięci głównej bloku zawierającego brakujący adres. Aby to osiągnąć, procesor podaje na magistralę sygnał oznaczający odczyt bez zamiaru modyfikacji (RWITM). Blok po załadowaniu jest natychmiast znakowany jako zmodyfikowany. W odniesieniu do pozostałych pamięci podręcznych ładowanie bloku danych poprzedzają dwa możliwe scenariusze.

Po pierwsze, jedna z pozostałych pamięci podręcznych może mieć zmodyfikowaną kopię tego bloku (stan = zmodyfikowany). W takim przypadku zaalarmowany procesor sygnalizuje procesorowi inicjującemu, że inny procesor ma zmodyfikowaną kopię bloku. Procesor inicjujący oddaje sterowanie magistralą i czeka. Drugi procesor przejmuje sterowanie magistralą, zapisuje zmodyfikowany blok w pamięci głównej i zmienia stan bloku w swojej pamięci podręcznej na nieważny (ponieważ procesor inicjujący zamierza zmodyfikować ten blok). Następnie procesor inicjujący ponownie przekazuje na magistralę sygnał RWITM, po czym zapisuje blok w pamięci głównej.

Drugi scenariusz występuje wtedy, gdy żadna inna pamięć podręczna nie ma zmodyfikowanej kopii wymaganego bloku. W tym przypadku w odpowiedzi nie nadchodzi żaden sygnał i procesor inicjujący przystępuje do wczytania i modyfikowania bloku. W tym samym czasie, jeśli jeden lub wiele procesorów ma czystą kopię bloku w stanie wspólnym, to każdy procesor unieważnia swoją kopię bloku. Jeśli jeden z procesorów ma czystą kopię bloku w stanie wyłącznym, to unieważnia ją.

Trafienie zapisu

Gdy następuje trafienie zapisu do bloku znajdującego się w lokalnej pamięci podręcznej, wynik *zależy* od bieżącego stanu tego bloku w pamięci podręcznej:

- Q *Wspólny*. Przed przeprowadzeniem aktualizacji procesor musi uzyskać prawo wyłączności dostępu do bloku. Procesor sygnalizuje swoje intencje za

pośrednictwem magistrali. Każdy procesor, który ma wspólną kopię bloku w swojej pamięci podręcznej, zmienia stan bloku ze wspólnego na nieważny. Następnie procesor inicjujący dokonuje aktualizacji i zmienia stan kopii bloku ze wspólnego na zmodyfikowany.

- *Wyłączny.* Procesor ma już wyłączną kontrolę nad blokiem, więc po prostu przeprowadza aktualizację i zmienia stan kopii bloku z wyłącznego na zmodyfikowany.
- *Zmodyfikowany.* Procesor ma już wyłączną kontrolę nad blokiem i blok ten jest zaznaczony jako zmodyfikowany, więc po prostu dokonuje aktualizacji.

16.3

OBLICZENIA WEKTOROWE

Chociaż wydajność dużych komputerów o ogólnym przeznaczeniu wciąż wzrasta, nadal istnieją zastosowania leżące poza zasięgiem współczesnych dużych komputerów. Występuje potrzeba komputerów rozwiązujących problemy matematyczne w rzeczywistych procesach, takich jakie występują w aerodynamice, sejsmologii, meteorologii oraz w fizyce atomowej, jądrowej i plazmowej [WILS84].

Problemy te charakteryzują się zwykle potrzebą wysokiej dokładności i programem, który powtarzalnie wykonuje zmiennopozycyjne operacje arytmetyczne na dużych tablicach liczb. Większość z tych problemów może być zaliczona do kategorii znanej jako **symulacja pola ciągłego** (ang. *continuous-fieldsimulation*). W istocie, sytuacja fizyczna może być opisana przez powierzchnię lub obszar w trzech wymiarach (np. przepływ powietrza w sąsiedztwie powierzchni rakiety). Powierzchnia ta jest aproksymowana przez siatkę punktów. Zbiór równań różniczkowych określa fizyczne zachowanie powierzchni w każdym punkcie. Równania są reprezentowane jako tablice wartości i współczynników, a rozwiązanie polega na powtarzaniu operacji arytmetycznych na tablicach danych.

Aby umożliwić rozwiązywanie tego rodzaju problemów, opracowano superkomputery. Maszyny te mogą wykonywać setki milionów operacji zmiennopozycyjnych na sekundę, a ich koszt waha się od 10 do 15 milionów dolarów. W przeciwieństwie do dużych komputerów, które zostały zaprojektowane pod kątem wieloprogramowania i intensywnego używania wejścia-wyjścia, superkomputery są optymalizowane pod kątem wyżej opisanych obliczeń numerycznych.

Zakres stosowania superkomputerów jest ograniczony, a ze względu na cenę ograniczony jest również ich rynek. Stosunkowo niewiele tych maszyn funkcjonuje, głównie w centrach badawczych i w niektórych agencjach rządowych wykonujących prace badawcze i inżynierskie. Podobnie jak w innych obszarach techniki komputerowej, także w przypadku superkomputerów występuje ciągłe

zapotrzebowanie na zwiększanie ich wydajności. W niektórych bieżących zastosowaniach w obszarze aerodynamiki i fizyki jądrowej do rozwiązania pojedynczego problemu wymagane jest wykonanie nawet 10^{13} operacji, co zajmuje ponad dwa dni czasu obliczeniowego współczesnego superkomputera [LEVI82]. Wobec tego ewolucja technologii i wydajności superkomputerów trwa nadal.

Istnieje inny rodzaj systemu, który został zaprojektowany w celu zaspokojenia zapotrzebowania na obliczenia wektorowe, nazwany procesorem tablicowym. Choć superkomputery są optymalizowane pod kątem obliczeń wektorowych, pozostają komputerami o ogólnym przeznaczeniu, zdolnymi do przetwarzania skalarne i ogólne przetwarzania danych. Procesory tablicowe nie wykonują przetwarzania skalarne; są skonfigurowane jako urządzenia peryferyjne wykorzystywane przez użytkowników dużych komputerów i minikomputerów w celu wykonywania wektorowych fragmentów programu.

Sposoby rozwiązywania obliczeń wektorowych

Kluczem do projektowania superkomputera lub procesora tablicowego jest uprzątnienie sobie, że głównym zadaniem jest wykonywanie operacji arytmetycznych na tablicach lub wektorach liczb zmiennopozycyjnych. W komputerze o ogólnym przeznaczeniu wymaga to iteracji każdego elementu tablicy. Rozważmy na przykład dwa wektory (tablice jednowymiarowe) liczb, A i B . Chcielibyśmy je dodać i umieścić wynik w wektorze C . W przykładzie z rys. 16.7 wymaga to wykonania sześciu oddzielnych dodawań. Jak moglibyśmy przyspieszyć to obliczenie? Odpowiedzią jest wprowadzenie pewnej postaci paralelizmu.

Stosowano kilka rozwiązań w celu wprowadzenia paralelizmu do obliczeń wektorowych. Zilustrujemy to na przykładzie zaczerpniętym z [STON80]. Rozważmy mnożenie wektorowe $C = A \times B$, gdzie A , B i C są macierzami $N \times N$. Dla każdego elementu macierzy C wykonuje się:

$$c_{i,j} = \sum_{k=1}^N a_{i,k} \cdot b_{k,j}$$

gdzie A , B i C mają elementy odpowiednio a_{ij} , b_{ij} i c_{ij} . Na rysunku 16.8a jest pokazany program w języku FORTRAN realizujący to obliczenie, który może być wykonywany na zwykłym, skalarnym procesorze.

$$\begin{array}{ccc}
 \begin{bmatrix} 1.5 \\ 7.1 \\ 6.9 \\ 100.5 \\ 0 \\ 59.7 \end{bmatrix} & + & \begin{bmatrix} 2.0 \\ 39.7 \\ 1000.003 \\ 11 \\ 21.1 \\ 19.7 \end{bmatrix} & = & \begin{bmatrix} 3.5 \\ 46.8 \\ 1006.903 \\ 111.5 \\ 21.1 \\ 79.4 \end{bmatrix} \\
 A & + & B & = & C
 \end{array}$$

RYSUNEK 16.7. Przykład dodawania wektorowego

```

(a)      DO 100 I = 1, N
          DO 100 J = 1, N
            C(I, J) = 0.0
            DO 100 K = 1, N
              C(I, J) = C(I, J) + A(I, K) * B(K, J)
100      CONTINUE

(b)      DO 100 I = 1, N
          C(I, J) = 0.0 (J = 1, N)
          DO 100 K = 1, N
            C(I, J) = C(I, J) + A(I, K) * B(K, J) (J = 1, N)
100      CONTINUE

(c)      DO 50 J = 1, N
          FORK 100
50      CONTINUE
          J = N
100     DO I = 1, N
          C(I, J) = 0.0
          DO 200 K = 1, N
            C(I, J) = C(I, J) + A(I, K) * B(K, J)
200     CONTINUE
          JOIN N

```

RYSUNEK 16.8. Mnożenie macierzowe ($C = A \times B$): (a) przetwarzanie skalarne; (b) przetwarzanie wektorowe; (c) przetwarzanie równoległe

Jednym z rozwiązań poprawiających wydajność może być **przetwarzanie wektorowe**. Zakłada się wtedy, że jest możliwe operowanie na jednowymiarowych wektorach danych. Na rysunku 16.8b jest pokazany program w FORTRAN-ie z nowym rozkazem, który umożliwia określanie obliczeń wektorowych. Notacja ($J = 1, N$) wskazuje, że operacje na wszystkich elementach J w danym przedziale powinny być przeprowadzone jako jedna operacja. Sposób, w jaki można to osiągnąć, omówimy pokrótce.

Program z rys. 16.8b wskazuje, że wszystkie elementy i -tego wiersza macierzy powinny być obliczane równoległe. Każdy element wiersza jest wynikiem sumowania, a sumowania te (po k) są wykonywane szeregowo, a nie równoległe. Mimo to potrzeba tylko N^2 mnożeń wektorowych przy zastosowaniu tego algorytmu w porównaniu z N^3 mnożeń skalarnych przy zastosowaniu algorytmu skalarnego.

Inne rozwiązanie, **przetwarzanie równoległe**, jest zilustrowane na rys. 16.8c. Zakłada się tutaj, że mamy N niezależnych procesorów, które mogą działać równoległe. W celu efektywnego wykorzystania procesorów musimy jakoś rozdzielić obliczenia na procesory. Używane są dwie funkcje pierwotne. Funkcja FORK n (rozwidlenie n) powoduje zapoczątkowanie niezależnego procesu w lokacji n . W tym samym czasie kontynuowane jest wykonywanie oryginalnego procesu począwszy od rozkazu następującego bezpośrednio po FORK. Każde wykonanie funkcji FORK powoduje powstanie nowego procesu. Rozkaz JOIN

(połączenie) jest w zasadzie odwróceniem FORK. Instrukcja JOIN N powoduje, że N niezależnych procesów ulega połączeniu w jeden, który kontynuuje wykonywanie poczynając od rozkazu następującego po JOIN. System operacyjny musi koordynować to połączenie, dzięki czemu wykonywanie nie jest kontynuowane, dopóki wszystkie N procesów nie dotrze do rozkazu JOIN.

Program pokazany na rys. 16.8c odzwierciedla zachowanie programu przetwarzania wektorowego. W przypadku programu przetwarzania równoległego każda kolumna macierzy C jest obliczana za pomocą oddzielnego procesu. Elementy danego wiersza macierzy C są więc obliczane równoległe.

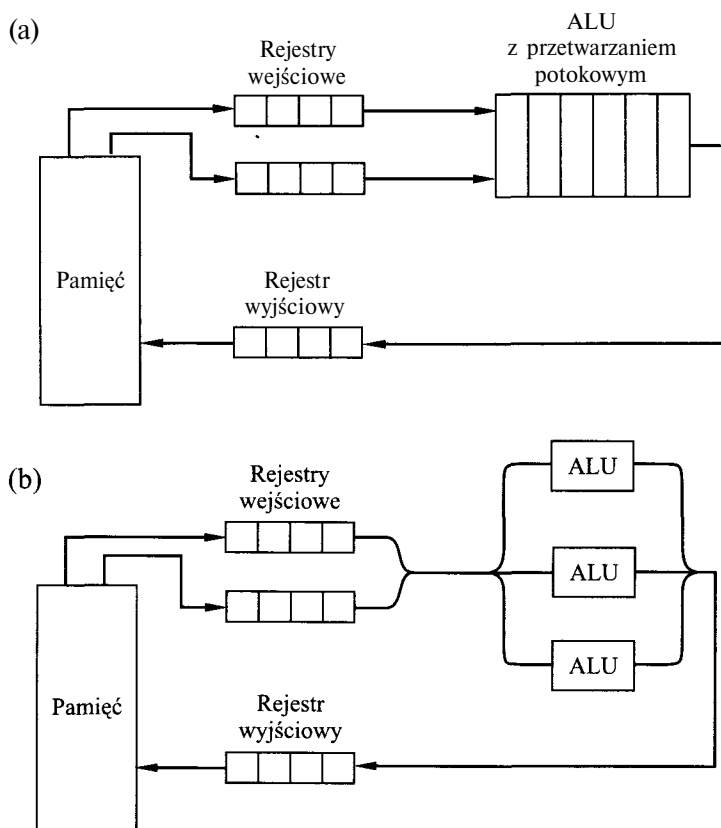
W dotychczasowej dyskusji przedstawiliśmy rozwiązania obliczeń wektorowych w ujęciu logicznym lub architektonicznym. Zajmijmy się teraz rodzajami organizacji procesora, które mogą posłużyć do wdrożenia tych rozwiązań. Próbowano w przeszłości i nadal się próbuje wielu różnych rozwiązań organizacyjnych. Występują trzy główne kategorie:

- a ALU z przetwarzaniem potokowym,
- a równoległe ALU,
- a równoległe procesory.

Na rysunku 16.9 są pokazane dwa pierwsze z tych rozwiązań. Przetwarzanie potokowe przedyskutowaliśmy już w rozdz. 11. Tutaj poszerzymy tę koncepcję o funkcjonowanie ALU. Ponieważ operacje zmiennopozycyjne są raczej złożone, istnieje możliwość dekompozycji tych operacji na etapy, dzięki czemu różne etapy mogą być realizowane współbieżnie na różnych zbiorach danych. Jest to zilustrowane na rys. 16.10a. Dodawanie zmiennopozycyjne zostało podzielone na cztery etapy (patrz rys. 8.21): porównywania, przesuwania, dodawania i normalizacji. Wektor liczb jest kierowany szeregowo do pierwszego etapu. W miarę postępowania przetwarzania, w potoku będą współbieżnie przetwarzane cztery różne zbiory liczb.

Powinno być oczywiste, że taka organizacja jest odpowiednia w przypadku przetwarzania wektorowego. Aby to dostrzec, rozważmy potokowe przetwarzanie rozkazów opisane w rozdz. 11. Procesor wykonuje powtarzalny cykl rozkazów pobierania i wykonywania. Przy braku rozgałęzień procesor wciąż pobiera rozkazy z szeregowo położonych lokacji. W konsekwencji potok jest wypełniony i osiąga się oszczędność czasu. Analogicznie, wykorzystując ALU z przetwarzaniem potokowym, umożliwia się zaoszczędzenie czasu tylko wtedy, gdy do ALU są dostarczane strumieniem dane położone w kolejnych lokacjach. Prosta, izolowana operacja zmiennopozycyjna nie ulega przyspieszeniu w wyniku stosowania przetwarzania potokowego. Osiąga się przyspieszenie, gdy do ALU jest doprowadzany wektor argumentów. Jednostka sterująca taktuje przechodzenie danych przez jednostkę ALU aż do czasu przetworzenia całego wektora.

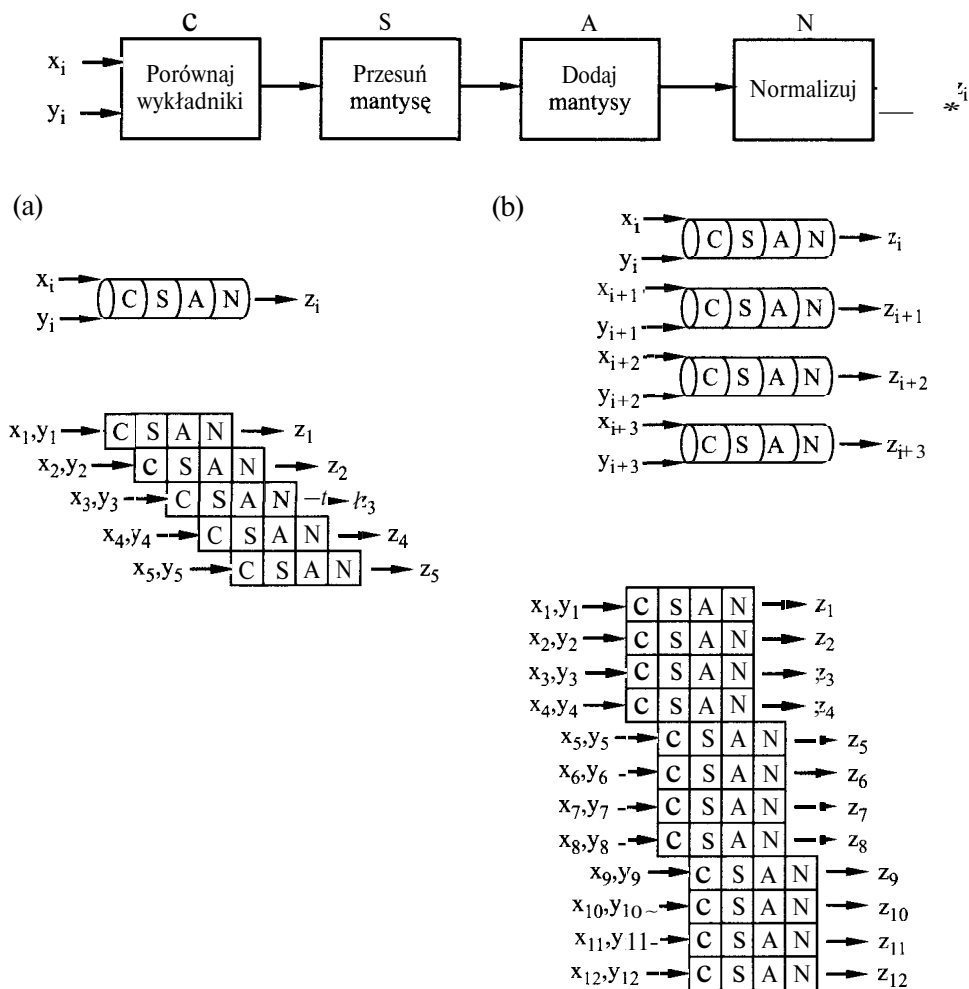
Można ulepszyć funkcjonowanie potoku, jeśli elementy wektora będą dostępne w rejestrach, zamiast w pamięci głównej. W rzeczywistości to właśnie zasugerowano na rys. 16.9a. Elementy każdego wektora argumentów są w pos-



RYSUNEK 16.9. Różne rozwiązania obliczeń wektorowych: (a) ALU z przetwarzaniem potokowym; (b) równoległe ALU

taci bloku ładowane do **rejestru wektorowego**, który jest po prostu dużym bankiem identycznych rejestrów. Wynik jest również umieszczany w rejestrze wektorowym. Wobec tego większość operacji obejmuje tylko użycie rejestrów i tylko operacje ładowania oraz zapisu na początku i na końcu operacji wektorowej wymagają dostępu do pamięci.

Mechanizm zilustrowany na rys. 16.10 może być określony jako **przetwarzanie potokowe wewnątrz operacji**. Oznacza to, że mamy pojedynczą operację arytmetyczną (np. $C = A + B$), która ma być zastosowana do argumentów wektora, natomiast przetwarzanie potokowe pozwala na równoległe przetwarzanie wielu elementów wektora. Mechanizm ten może być wzmocniony przez **przetwarzanie potokowe poprzez operacje**. W tym ostatnim przypadku występuje sekwencja arytmetycznych operacji wektorowych, a do przyspieszenia przetwarzania jest stosowane potokowe przetwarzanie rozkazów. Jedno z rozwiązań takiego mechanizmu, określane jako **łączenie w łańcuchy** (ang. *chaining*), występuje w superkomputerach Cray. Podstawowa reguła łączenia w łańcuchy jest następująca. Operacja wektorowa może być rozpoczęta, gdy tylko jest



RYСУNEK 16.10. Przetwarzanie potokowe: (a) ALU z przetwarzaniem potokowym; (b) cztery równoległe ALU

osiągalny pierwszy element wektora (wektorów) argumentów i gdy jednostka funkcjonalna (np. dodawania, odejmowania, mnożenia, dzielenia) jest wolna. W istocie łączenie w łańcuchy powoduje, że wyniki z jednej jednostki funkcjonalnej są natychmiast kierowane do następnej itd. Jeśli używane są rejestry wektorowe, to wyniki pośrednie nie muszą być zapisywane w pamięci i mogą być używane, zanim jeszcze zakończy się operacja wektorowa, która była ich źródłem.

Obliczając na przykład $C = (sXA) + B$, gdzie A , B i C są wektorami, a s jest skalar, Cray może jednocześnie wykonywać trzy rozkazy. Elementy pobrane do ładowania są natychmiast wprowadzane do potokowo przetwarzającego układu mnożącego, iloczyny są wysyłane do potokowo przetwarzającego sumatora, natomiast sumy są umieszczane w rejestrze wektorowym natychmiast po obliczeniu ich przez sumator:

1. Ładowanie wektora	$A \rightarrow \text{Rejestr wektorowy (VR1)}$
2. Ładowanie wektora	$B \rightarrow \text{VR2}$
3. Mnożenie wektora	$s \times \text{VR1} \rightarrow \text{VR3}$
4. Dodawanie wektora	$\text{VR3} + \text{VR2} \rightarrow \text{VR4}$
5. Zapis wektora	$\text{VR4} \rightarrow C$

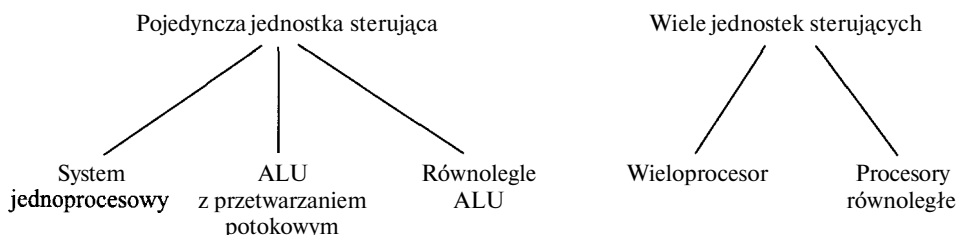
Rozkazy 2 i 3 mogą być łączone w łańcuch (przetwarzane potokowo), ponieważ dotyczą one innych lokacji w pamięci i rejestrach. Rozkaz 4 potrzebuje wyników rozkazów 2 i 3, jednak również może być wraz z nimi łączony w łańcuch. Gdy tylko są osiągalne pierwsze elementy rejestrów wektorowych 2 i 3, może się rozpocząć operacja należąca do rozkazu 4.

Innym sposobem przetwarzania wektorowego jest użycie wielu ALU w jednym procesorze pod kontrolą jednej jednostki sterującej. W tym przypadku jednostka sterująca rozprowadza dane do wszystkich ALU, dzięki czemu mogą one pracować równolegle. Jest również możliwe użycie przetwarzania potokowego w każdej z równoległych ALU. Jest to zilustrowane na rys. 16.10b. W tym przykładzie równolegle funkcjonują cztery ALU.

Podobnie jak organizacja potokowa, także zastosowanie równoległych ALU jest odpowiednie do przetwarzania wektorowego. Jednostka sterująca rozprowadza elementy wektora do wszystkich ALU w sposób cykliczny, aż do przetworzenia wszystkich elementów. Ten rodzaj organizacji jest bardziej złożony niż procesor z jedną ALU.

Przetwarzanie wektorowe może być wreszcie realizowane za pomocą wielu równolegle przetwarzających procesorów. W tym przypadku konieczne jest rozbicie zadania na wiele procesów, które mogą być wykonywane równolegle. Organizacja taka jest efektywna tylko wtedy, gdy jest dostępne oprogramowanie i sprzęt umożliwiające skuteczną koordynację równoległych procesorów. Wciąż jeszcze jest to czynnym obszarem badań, chociaż pojawiły się już pewne wyroby [GEHR88].

Możemy teraz poszerzyć naszą taksonomię przedstawioną w podrozdz. 16.1 w celu uwzględnienia tych nowych struktur, co widać na rys. 16.11. Organizacje komputerów mogą być rozróżnione na podstawie obecności jednej lub wielu jednostek sterujących. Wiele jednostek sterujących implikuje zastosowanie wielu procesorów. Zgodnie z naszymi poprzednimi rozważaniami, jeśli wiele



RYСУNEK 16.11. Taksonomia organizacji komputerów

procesorów może współpracować przy wykonywaniu danego zadania, są one określane jako **procesory równoległe**.

Należy wystrzegać się pewnej niefortunnej terminologii spotykanej w literaturze. Termin *procesor wektorowy* jest często zrównywany z ALU o organizacji potokowej, chociaż równoległa organizacja ALU jest również przeznaczona do przetwarzania wektorowego i, jak mówiliśmy, organizacja procesorów równoległych również może służyć przetwarzaniu wektorowemu. *Przetwarzanie tablicowe* jest czasem wiązane z równoległymi ALU, chociaż każda z tych trzech organizacji również jest optymalizowana pod kątem przetwarzania tablic. Jest jeszcze gorzej; termin *procesor tablicowy* zwykle odnosi się do procesora pomocniczego, który jest dołączony do procesora o ogólnym przeznaczeniu i jest używany do prowadzenia obliczeń wektorowych. W procesorach tablicowych mogą być stosowane ALU o organizacji potokowej lub równoległej.

Obecnie na rynku dominuje organizacja z ALU przetwarzającymi potokowo [KOZD80, RIGA84]. Systemy z przetwarzaniem potokowym są mniej złożone niż dwa pozostałe rozwiązania. Projektowanie jednostek sterujących i systemów operacyjnych zostało opanowane na tyle dobrze, żeby uzyskać efektywne przydzielanie zasobów i wysoką wydajność. Pozostałą część tego podrozdziału poświęcimy bardziej szczegółowej analizie tego właśnie rozwiązania na podstawie konkretnego przykładu.

Urządzenie wektorowe IBM 3090

Dobrym przykładem organizacji z ALU przetwarzającą potokowo, przeznaczoną do przetwarzania wektorowego, jest urządzenie wektorowe opracowane przez IBM dla architektury IBM 370 i wdrożone w największych systemach serii 3090 [PADE88, TUCK87]. Urządzenie to jest opcjonalnym dodatkiem do systemu bazowego, jednak jest z nim ściśle zintegrowane. Zastępuje ono rozwiązania przetwarzania wektorowego występujące w superkomputerach, takich jak rodzina Cray.

Urządzenie IBM używa pewnej liczby rejestrów wektorowych. Każdy z nich jest w istocie bankiem rejestrów skalarnych. W celu obliczenia sumy wektorowej $C = A + B$, wektory A i B są ładowane do dwóch rejestrów wektorowych. Dane z tych rejestrów są przeprowadzane przez ALU tak szybko, jak tylko jest to możliwe, wyniki zaś są zapisywane w trzecim rejestrze wektorowym. Nakładanie się obliczeń i ładowanie danych wejściowych do rejestrów w postaci bloku powodują znaczne przyspieszenie w porównaniu ze zwykłą operacją ALU.

Organizacja

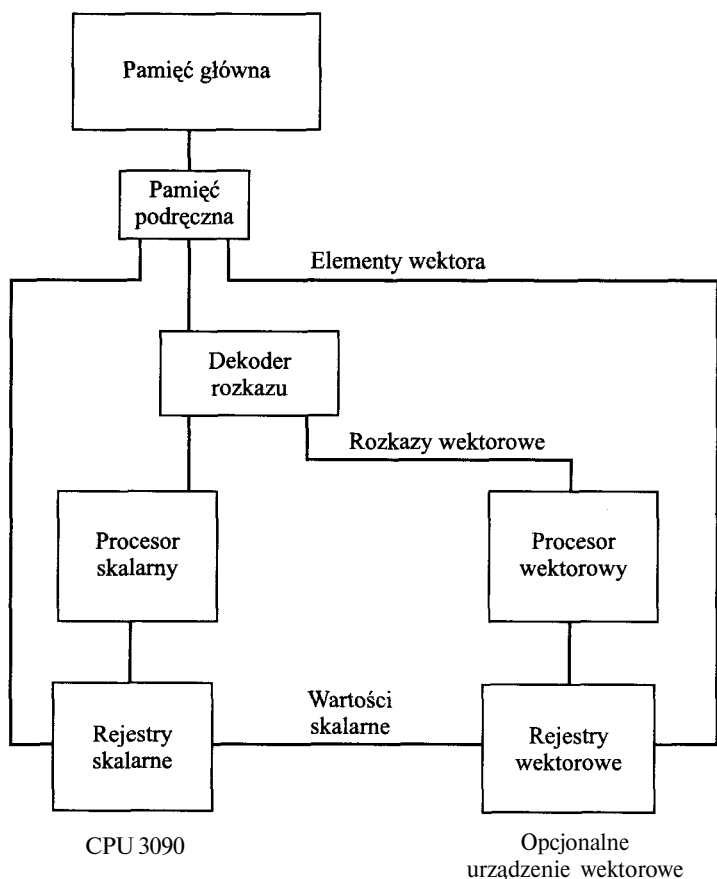
Architektura wektorowa IBM oraz podobne do niej wektorowe ALU z przetwarzaniem potokowym pozwalają na uzyskiwanie większej wydajności niż pętle skalarnych rozkazów arytmetycznych dzięki temu, że:

- a Ustalona i z góry określona struktura danych wektorowych umożliwia zastąpienie rozkazów uporządkowanych w pętli przez szybsze, wewnętrzne (układowe lub mikroprogramowane) operacje maszynowe.
- a Operacje dostępu do danych i arytmetyczne na wielu kolejnych elementach wektorowych mogą być prowadzone współbieżnie przez nakładanie się tych operacji w rozwiązaniu potokowym lub przez wieloelementowe operacje równoległe.
- a Używanie rejestrów wektorowych do przechowywania wyników pośrednich pozwala uniknąć dodatkowych odniesień do pamięci.

Na rysunku 16.12 jest pokazana ogólna organizacja urządzenia wektorowego. Chociaż jest ono widziane jako fizycznie oddzielony dodatek do procesora, jego architektura stanowi poszerzenie architektury systemu 370 i jest z nią kompatybilna. Urządzenie wektorowe jest integrowane z architekturą systemu 370 w następujący sposób:

- a Do wszystkich operacji skalarnych są używane istniejące rozkazy systemu 370.
- Operacje arytmetyczne na indywidualnych elementach wektorów dają dokładnie takie same wyniki, jak odpowiednie rozkazy skalarnie systemu 370. Na przykład jedna z decyzji projektowych dotyczyła określenia wyniku zmiennopozycyjnej operacji DIVIDE (dzielenia). Czy wynik powinien być tak samo dokładny, jak wynik zmiennopozycyjnego dzielenia skalarnego, czy też dopuszczalna jest pewna aproksymacja umożliwiająca zwiększenie szybkości, lecz czasem prowadząca do błędu na jednej lub wielu pozycjach bitowych niskiego rzędu? Podjęto decyzję utrzymania całkowitej kompatybilności z architekturą systemu 370 kosztem nieznacznego pogorszenia wydajności.
- a Rozkazy wektorowe mogą być przerywane, a ich wykonywanie może być wznawiane od punktu przerwania po podjęciu odpowiednich działań w sposób kompatybilny ze schematem obsługi programu przerwań w systemie 370.
- a Wyjątki arytmetyczne są takie same lub stanowią poszerzenie w stosunku do wyjątków związanych ze skalarnymi rozkazami arytmetycznymi systemu 370, mogą też być używane podobne podprogramy porządkujące. W tym celu zastosowano wskaźnik przerwania wektorowego wskazujący położenie w rejestrze wektorowym, którego dotyczy wyjątek (np. przepełnienie). Po przywróceniu wykonywania rozkazu wektorowego następuje więc dostęp do właściwego miejsca w rejestrze wektorowym.
- a Dane wektorowe pozostają w pamięci wirtualnej, przy czym błędy stron są traktowane w sposób standardowy.

Ten poziom integracji jest powodem wielu korzyści. Istniejące systemy operacyjne mogą współpracować z urządzeniem wektorowym po niewielu tylko rozszerzeniach. Istniejące programy aplikacyjne, kompilatory języka i inne



RYSUNEK 16.12. IBM 3090 z urządzeniem wektorowym

oprogramowanie mogą być używane bez zmian. W celu wykorzystania możliwości przetwarzania wektorowego oprogramowanie może też być modyfikowane według życzenia.

Rejestry

Kluczowym zagadnieniem przy projektowaniu urządzenia wektorowego jest to, czy argumenty są umieszczane w rejestrach, czy w pamięci. Organizacja IBM jest określana jako **z rejestru do rejestru**, ponieważ zarówno wejściowe, jak i wyjściowe argumenty wektorowe mogą być umieszczane w **rejestrach** wektorowych. Rozwiązanie takie zostało również użyte w superkomputerze Cray. Rozwiązaniem alternatywnym, stosowanym w maszynach Control Data, jest uzyskiwanie argumentów bezpośrednio z pamięci. Główną niedogodnością używania rejestrów wektorowych jest to, że muszą one być uwzględniane przez programistę lub przez kompilator. Załóżmy na przykład, że długość rejestrów

wektorowych wynosi K , a długość przetwarzanych wektorów jest równa $N > K$. W takim przypadku musi być wykonana pętla wektorowa, podczas której w określonej chwili prowadzona jest operacja na K elementach, a pętla jest powtarzana N/K razy. Główną zaletą stosowania rejestrów wektorowych jest to, że operacje są oderwane od wolniejszej pamięci i zachodzą przede wszystkim w rejestrach.

Przyspieszenie, jakie można osiągnąć dzięki rejestrom, jest zilustrowane na rys. 16.13 [PADE88]. Program FORTRAN mnoży wektor A przez wektor B w celu uzyskania C , przy czym każdy wektor ma część rzeczywistą (AR, BR, CR) i urojoną (AI, BI, CI). Urządzenie 3090 może dokonywać jednego dostępu do pamięci głównej na jeden cykl procesora lub zegara (może to być zapis, albo odczyt). Urządzenie to ma rejestry, które w ciągu cyklu umożliwiają dwa dostępy dla odczytu i jeden dla zapisu. W jednostce arytmetycznej urządzenia powstaje jeden wynik w ciągu jednego cyklu. Załóżmy użycie rozkazów, które umożliwiają określanie dwóch argumentów źródłowych i wyniku¹. Na części rysunku widać, że w przypadku rozkazów z pamięci do pamięci każda iteracja wymaga łącznie 18 cykli. W przypadku architektury typu z rejestru do rejestru (część b) czas ten został zredukowany do 12 cykli. Oczywiście w przypadku operacji z rejestru do rejestru wielkości wektorowe muszą być przed obliczaniem załadowane do rejestrów wektorowych i zapisane w pamięci po zakończeniu obliczeń. W przypadku dużych wektorów ta strata jest stosunkowo niewielka. Na rysunku 16.13c widać, że możliwość określania w jednym rozkazie zarówno argumentów w rejestrach, jak i w pamięci, powoduje zmniejszenie czasu iteracji do 10 cykli. Ten ostatni rodzaj rozkazów został uwzględniony w architekturze wektorowej².

Na rysunku 16.14 są pokazane rejestry stanowiące część urządzenia wektorowego IBM 3090. Występuje tu 16 32-bitowych rejestrów wektorowych. Rejestry wektorowe mogą być łączone w celu utworzenia 8 rejestrów 64-bitowych. Każdy element rejestru może zawierać liczbę całkowitą lub zmiennopozycyjną. Rejestry wektorowe mogą więc służyć do przechowywania 32- i 64-bitowych wartości całkowitych oraz 32- i 64-bitowych wartości zmiennopozycyjnych.

W tej architekturze każdy rejestr zawiera od 8 do 512 elementów skalarnych. Wyborowi rzeczywistej długości towarzyszy pewien dylemat projektowy. Czas wykonania operacji wektorowej składa się w zasadzie z czasu uruchomienia potoku i zapełnienia rejestru plus jeden cykl na każdy element wektora. Wobec tego używanie dużej liczby elementów rejestrów powoduje zmniejszenie względnego czasu uruchamiania obliczeń. Ta efektywność jest jednak równowa-

¹ W architekturze 370 jedyne 3-argumentowe rozkazy (rozkazy rejestru i pamięci, RS) określają dwa argumenty w rejestrach i jeden w pamięci. W części tego przykładu zakładamy istnienie rozkazów 3-argumentowych, których wszystkie argumenty znajdują się w pamięci głównej. Czynione to jest dla celów porównania. W rzeczywistości taki właśnie format rozkazów mógłby być wybrany dla architektury wektorowej.

² Przedyskutowane w dalszym ciągu rozkazy mieszane umożliwiają dalszą redukcję.

Program w języku FORTRAN:

```
DO 100 J=1,50
  CR (J) = AR (J) *BR (J) - AI (J)*BI (J)
100  CI (J) = AR (J) *BI (J) - AI (J)*BR (J)
```

a)

Operacja	Cykle
AR (J) *BR (J) → T1 (J)	3
AI (J) *BI (J) → T2 (J)	3
T1 (J) - T2 (J) → CR(J)	3
AR (J) *BI (J) → T3 (J)	3
AI (J) *BR (J) → T4 (J)	3
T3 (J) + T4 (J) → C1 (J)	3
Razem	18

b)

Operacja	Cykle
AR (J) → V1 (J)	1
BR (J) → V2 (J)	1
V1 (J) *V2 (J) → V3 (J)	1
AI (J) → V4 (J)	1
BI (J) → V5 (J)	1
V4 (J) *V5 (J) → V6 (J)	1
V3 (J) - V6 (J) → V7 (J)	1
V7 (J) → CR (J)	1
V1 (J) *V5 (J) → V8 (J)	1
V4 (J) *V2 (J) → V9 (J)	1
V8 (J) + V9 (J) → V0 (J)	1
V0 (J) → CI (J)	1
Razem	12

c)

Operacja	Cykle
AR (J) → V1 (J)	1
V1 (J) *BR (J) → V2 (J)	1
AI (J) → V3 (J)	1
V3 (J) *BI (J) → V4 (J)	1
V2 (J) - V4 (J) → V5 (J)	1
V5 (J) → CR (J)	1
V1 (J) *BI (J) → V6 (J)	1
V3 (J) *BR (J) → V7 (J)	1
V6 (J) + V7 (J) → V8 (J)	1
V8 (J) CI (J)	1
Razem	10

d)

Operacja	Cykle
AR (J) → V1 (J)	1
V1 (J) *BR (J) → V2 (J)	1
AI (J) → V3 (J)	1
V2 (J) - V3 (J) → V2 (J)	1
V2 (J) → CR (J)	1
V1 (J) *BI (J) → V4 (J)	1
V4 (J) + V3 (J) → V8 (J)	1
V4 (J) CI (J)	1
Razem	8

Vi - rejestry wektorowe

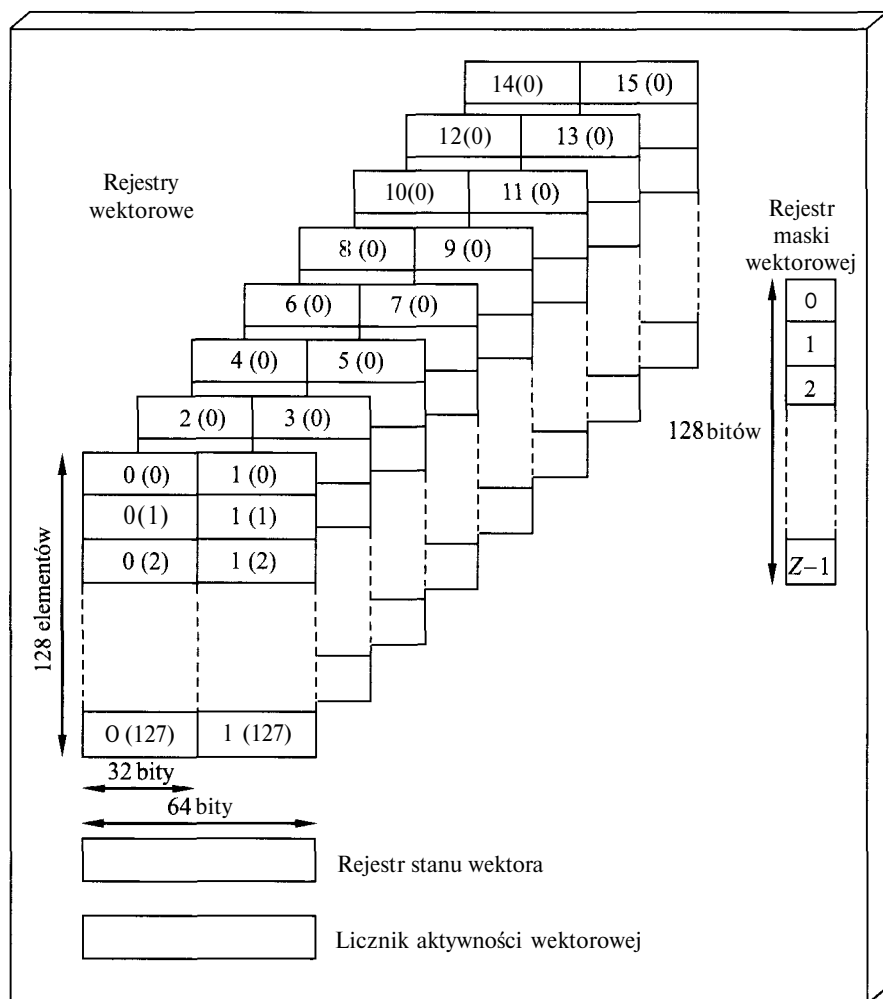
AR, BR, AI, BI - argumenty w pamięci

Ti - tymczasowe lokacje w pamięci

RYSUNEK 16.13. Alternatywne programy obliczeń wektorowych; (a) pamięć-pamięć; (b) rejestr-rejestr; (c) pamięć-rejestr; (d) rozkazy złożone

zona przez dodatkowy czas potrzebny do zachowywania i odnawiania rejestrów wektorowych przy zmianie procesów oraz przez praktyczne ograniczenia kosztów i przestrzeni. Rozważania te doprowadziły do użycia 128 elementów w rejestrze w aktualnym wdrożeniu 3090.

W urzędzeniu tym okazały się potrzebne 3 dodatkowe rejestry. Rejestr maskowania wektorowego zawiera bity maskujące, które mogą być używane do wybierania tych elementów rejestrów wektorowych, które mają być przetwarzane w przypadku szczególnych operacji. Rejestr stanu wektorowego zawiera poła sterowania, takie jak licznik wektorowy, który określa, ile elementów zawartych w rejestrach wektorowych ma być przetwarzanych. Licznik aktywności wektorowej rejestruje czas spędzany na wykonywaniu rozkazów wektorowych.



RYSUNEK 16.14. Rejestry urządzenia wektorowego IBM 3090

Rozkazy złożone

Jak stwierdziliśmy powyżej, można poprawić wydajność, nakładając na siebie wykonywanie rozkazów za pomocą łączenia w łańcuchy. Projektanci urządzenia wektorowego IBM zdecydowali się nie stosować tego rozwiązania z kilku powodów. Architektura systemu 370 musiałaby być poszerzona w celu umożliwienia przetwarzania złożonych przerwań (przy uwzględnieniu ich wpływu na zarządzanie pamięcią wirtualną), a odpowiednie zmiany musiałby też nastąpić w oprogramowaniu. Bardziej nawet istotną sprawą był koszt włączenia dodatkowych układów sterujących i ścieżek dostępu do rejestrów dla potrzeb łączenia w łańcuchy.

Zamiast tego przewidziano trzy operacje, łączące w postaci jednego rozkazu (jeden kod operacji) najczęściej spotykane sekwencje obliczania wektorowego, a mianowicie mnożenie z następującym po nim dodawaniem, odejmowaniem lub sumowaniem*. Na przykład rozkaz typu z pamięci do rejestru MULTIPLY-AND-ADD (pomnóż i dodaj) powoduje pobranie wektora z pamięci, pomnożenie go przez wektor z rejestru i dodanie iloczynu do trzeciego wektora w rejestrze. Dzięki zastosowaniu złożonych rozkazów MULTIPLY-AND-ADD oraz MULTIPLY-AND-SUBTRACT w przykładzie z rys. 16.13 można zredukować łączny czas iteracji z 10 do 8 cykli.

W przeciwieństwie do łączenia w łańcuchy, złożone rozkazy nie wymagają dodatkowych rejestrów do czasowego przechowywania wyników pośrednich i potrzebują o jeden mniej dostęp do rejestru. Rozważmy na przykład następujący łańcuch:

$$\begin{aligned} A &\rightarrow VR1 \\ VR1 + VR2 &\rightarrow VR1 \end{aligned}$$

W tym przypadku są wymagane dwa zapisy w rejestrze wektorowym VR1. W architekturze IBM występuje rozkaz typu z pamięci do rejestru ADD (dodaj). Przy użyciu tego rozkazu tylko suma jest umieszczana w VR1. Rozkaz złożony czyni również niepotrzebnym odzwierciedlanie w opisie stanu maszyny współbieżnego wykonywania pewnej liczby rozkazów, co upraszcza zachowywanie i odtwarzanie stanu przez system operacyjny oraz przetwarzanie przerw.

Lista rozkazów

W tabeli 16.2 są podsumowane operacje arytmetyczne i logiczne określone dla architektury wektorowej. Ponadto istnieją rozkazy ładowania z pamięci do rejestru i rozkazy zapisu z rejestru do pamięci. Zauważmy, że wiele z tych rozkazów używa formatu 3-argumentowego. Istnieje także pewna liczba wariantów wielu rozkazów, zależnie od położenia argumentów. Argumentem źródłowym może być zawartość rejestru wektorowego (V), pamięci (S) lub rejestru skalarnego (Q). Miejscem docelowym jest zawsze rejestr wektorowy, z wyjątkiem porównania, którego wynik trafia do rejestru maskowania wektorowego. Przy uwzględnieniu tych wszystkich wariantów łączna liczba kodów operacji (oddzielnych rozkazów) wynosi 171. Jest to raczej duża liczba, jednak nie jest to aż tak kosztowne przy wdrażaniu, jak można by sobie wyobrażać. Gdy maszyna zawiera jednostki arytmetyczne, ścieżki danych dostarczające argumentów z pamięci, rejestry skalarne i rejestry wektorowe związane z potokami wektorowymi, główne koszty zostały już poniesione. Przedstawiona architektura może, przy niewielkim koszcie dodatkowym, dostarczyć bogatego zbioru wariantów wykorzystania tych rejestrów i potoków.

* Sumowanie jest tu rozumiane jako sumowanie elementów wektora wykonywane na rozkaz ACCUMULATE (*przyp. tłum.*).

TABELA 16.2. Urządzenie wektorowe IBM 3090: rozkazy arytmetyczne i logiczne

Operacja	Rodzaje danych			Lokalizacje argumentów			
	Zmiennopozycyjne		Binarne lub logiczne				
	Długie	Krótkie					
Dodaj	FL	FS	BI	$V + V \rightarrow V$	$V + S \rightarrow V$	$Q + V \rightarrow V$	$Q + S \rightarrow V$
Odejmij	FL	FS	BI	$V - V \rightarrow V$	$V - S \rightarrow V$	$Q - V \rightarrow V$	$Q - S \rightarrow V$
Pomnóż	FL	FS	BI	$V * V \rightarrow V$	$V * S \rightarrow V$	$Q * V \rightarrow V$	$Q * S \rightarrow V$
Podziel	FL	FS	–	$V/V \rightarrow V$	$V/S \rightarrow V$	$Q/V \rightarrow V$	$Q/S \rightarrow V$
Porównaj	FL	FS	BI	$V \bullet V \rightarrow V$	$V \bullet S \rightarrow V$	$Q \bullet V \rightarrow V$	$Q \bullet S \rightarrow V$
Pomnóż i dodaj	FL	FS	–		$V + V * S \rightarrow V$	$V + Q * V \rightarrow V$	$V + Q * S \rightarrow V$
Pomnóż i odejmij	FL	FS	–		$V - V * S \rightarrow V$	$V - Q * V \rightarrow V$	$V - Q * S \rightarrow V$
Pomnóż i akumuluj	FL	FS	–	$P + \bullet V \rightarrow V$	$P + \bullet S \rightarrow V$		
Dopełnij	FL	FS	BI	$\neg V \rightarrow V$			
Dodatnia bezwzględna	FL	FS	BI	$ V \rightarrow V$			
Ujemna bezwzględna	FL	FS	BI	$\neg V \rightarrow V$			
Maksymalna	FL	FS	–			$Q \bullet V \rightarrow V$	
Maksymalna bezwzględna	FL	FS	–			$Q \bullet V \rightarrow V$	
Minimalna	FL	FS	–			$Q \bullet V \rightarrow V$	
Przesuń logicznie w lewo	–	–	LO	$\bullet V \rightarrow V$			
Przesuń logicznie w prawo	–	–	LO	$\bullet V \rightarrow V$			
Wykonaj AND	–	–	LO	$V \& V \rightarrow V$	$V \& S \rightarrow V$	$Q \& V \rightarrow V$	$Q \& S \rightarrow V$
Wykonaj OR	–	–	LO	$V V \rightarrow V$	$V S \rightarrow V$	$Q V \rightarrow V$	$Q S \rightarrow V$
Wykonaj Exclusive-OR	–	–	LO	$V \neq V \rightarrow V$	$V \neq S \rightarrow V$	$Q \neq V \rightarrow V$	$Q \neq S \rightarrow V$

Objaśnienia:

Rodzaje danych

FL - długa zmiennopozycyjna,

FS - krótka zmiennopozycyjna,

BI — binarna całkowita,

LO - logiczna.

Lokalizacje argumentów

V - rejestr wektorowy,

S - pamięć,

Q - rejestr skalarny (roboczy lub zmiennopozycyjny),

P — sumy częściowe w rejestrze wektorowym,

• — operacja specjalna.

Większość rozkazów zamieszczonych w tablicy 16.2 jest zrozumiała sama przez się. Dwa rozkazy sumowania wymagają dodatkowych wyjaśnień. Operacja akumulowania powoduje zsumowanie elementów pojedynczego wektora (ACCUMULATE) lub elementów iloczynu dwóch wektorów (MULTIPLY-AND-ACCUMULATE). Rozkazy te stanowią interesujący problem projektowy. Chcielibyśmy wykonywać te operacje możliwie szybko, w pełni wykorzystując możliwości potoku ALU. Trudność polega na tym, że suma dwóch liczb wprowadzona do potoku jest osiągalna dopiero po upływie kilku cykli. Wobec tego trzeci element wektora nie może być dodany do sumy dwóch pierwszych elementów, dopóki te elementy nie przejdą przez cały potok. W celu przezwyciężenia tego problemu elementy wektora są dodawane w taki sposób, żeby powstały cztery sumy cząstkowe. W szczególności kolejno dodawane elementy 0, 4, 8, 12, ..., 124 dają sumę cząstkową 0; elementy 1, 5, 9, 13, ..., 125 sumę cząstkową 1; elementy 2, 6, 10, 14, ..., 126 sumę cząstkową 2 oraz elementy 3, 7, 11, 15, ..., 127 sumę cząstkową 3. Każda z tych sum cząstkowych może przechodzić przez potok z maksymalną szybkością, ponieważ opóźnienie w potoku wynosi około 4 cykli. Do przechowywania sum cząstkowych jest używany oddzielny rejestr wektorowy. Gdy wszystkie elementy oryginalnego wektora zostały przetworzone, sumowane są 4 sumy cząstkowe, dając wynik końcowy. Wydajność tej drugiej fazy nie jest krytyczna, ponieważ obejmuje ona tylko 4 elementy wektorowe.

16.4

PROCESORY RÓWNOLEGŁE

Typowy, dostępny dzisiaj handlowo system wieloprocesorowy obejmuje tylko kilka oddzielnych procesorów, zwykle 2+12. Od lat istniało zainteresowanie rozwojem systemów wieloprocesorowych o wielkiej liczbie procesorów - liczonej w setkach, tysiącach a nawet w dziesiątkach tysięcy. Systemy takie są zwykle określane jako systemy procesorów równoległych (ang. *parallel processor systems*) lub, w przypadku tysięcy procesorów, jako masowe systemy procesorów równoległych (ang. *massively parallel processor systems*).

Aż do niedawna niewiele było produktów handlowych odzwierciedlających prowadzone w tym obszarze badania. Obecnie wyroby takie zaczęły się pojawiać. Do głównych czynników, które doprowadziły do praktycznej realizacji procesorów równoległych, należą:

1. *Zastosowanie handlowo dostępnych procesorów RISC.* Tanie procesory RISC o wysokiej wydajności są obecnie handlowo dostępne. W wielu systemach służą one jako podstawowe elementy składowe, uwalniając projektanta systemu procesorów równoległych od potrzeby projektowania podstawowego procesora.

2. *Postęp w schematach komunikacji międzyprocesorami.* Przy setkach i tysiącach współpracujących procesorów wydajne i bezpieczne metody koordynacji i wymiany danych mają żywotne znaczenie.
3. *Zapotrzebowanie na wydajność.* Nawet przy zdumiewającym zwiększaniu wydajności towarzyszącym każdej kolejnej generacji procesorów, zapotrzebowanie na wydajność rośnie jeszcze szybciej. Systemy procesorów równoległych stanowią środek zwiększania wydajności przy użyciu istniejącej technologii procesorów.

Ten ostatni punkt jest szczególnie ważny. Mimo szybkości, z jaką powstają nowe generacje procesorów, zapotrzebowanie pozostaje zawsze o krok w przodzie. Dotyczy to nie tylko sfery superkomputerów, ale także systemów biurowych. Za każdym razem, gdy firmy komputerowe dostarczają produkty o coraz większej mocy, powstają nowe zastosowania wykorzystujące w pełni tę moc i domagające się większej mocy. Komputer osobisty Intel 80486 przetwarzający teksty w systemie DOS ma oczywiście nadmiar mocy, jednak jest zaledwie wystarczający w środowisku Windows. Systemy Pentium z łatwością wykorzystują możliwości Windows, jednak zaczynają ustępować przy próbie obsługiwanie funkcji multimedialnych (np. wizji na żywo lub wideofonów) lub tak zaawansowanych technik, jak rozpoznawanie głosu w czasie rzeczywistym.

Zagadnienie przetwarzania równoległego jest złożone i obejmuje problemy organizacji, struktury połączeń, komunikacji między procesorami, projektowania systemów operacyjnych i tworzenia oprogramowania aplikacyjnego. Celem tego podrozdziału jest dokonanie krótkiego przeglądu najważniejszych koncepcji organizacyjnych.

Rodzaje procesorów równoległych

Systemem procesorów równoległych jest taki system, w którym wiele połączonych ze sobą procesorów współpracuje przy wykonywaniu programów użytkowych. Termin ten odnosi się do systemów, które zapewniają duży stopień współbieżności (paralelizmu). Systemy te oferują struktury umożliwiające rozwijanie oprogramowania systemowego i użytkowego wykorzystującego paralelizm. Wyłączone z tej klasy są systemy o niskim stopniu paralelizmu, jednak te właśnie systemy są na ogół systemami jednoprocessorowymi. Do przykładów niskiego stopnia paralelizmu należą:

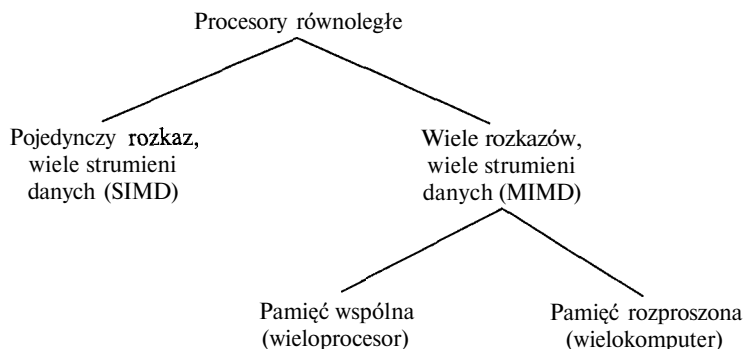
- ❑ *Potokowe przetwarzanie rozkazów.* Wykonywanie każdego rozkazu jest dzielone na sekwencję etapów, dzięki czemu możliwe jest równoległe wykonywanie wielu rozkazów, z których każdy znajduje się na innym etapie wykonywania.
- ❑ *Zwielokrotnione jednostki funkcjonalne procesora.* Powielenie ALU pozwala na rozwiązanie superskalarne, w którym wiele rozkazów jest wykonywanych równoległe, przy czym wszystkie znajdują się na tym samym etapie wykonywania.

- ❑ *Oddzielne procesory wyspecjalizowane.* Najczęściej spotykanym przykładem jest stosowanie procesorów wejścia-wyjścia, które uwalniają procesor od odpowiedzialności za szczegółowe sterowanie wejściem-wyjściem.

Taksonomia porządkująca systemy procesorów równoległych wprowadzona po raz pierwszy przez Flynna [FLYN72] nadal jest najczęściej stosowana. Flynn zaproponował następujące kategorie systemów komputerowych:

- Q Z pojedynczym strumieniem rozkazów i danych (ang. *single instruction single data stream* - SISD). Pojedynczy procesor interpretuje pojedynczy strumień rozkazów i operuje na danych przechowywanych w pojedynczej pamięci. Możliwe jest osiągnięcie pewnego stopnia paralelizmu przy zastosowaniu jednej z wyżej wymienionych metod.
- ❑ Z pojedynczym strumieniem rozkazów i zwielokrotnionym strumieniem danych (ang. *single instruction multiple data stream* - SIMD). Pojedynczy rozkaz maszynowy steruje jednoczesnym działaniem pewnej liczby elementów przetwarzających. Każdy element przetwarzający ma towarzyszącą pamięć danych, dzięki czemu każdy rozkaz jest wykonywany na innym zbiorze danych przez różne procesory. Do tej kategorii należą procesory wektorowe i tablicowe.
- a Ze zwielokrotnionym strumieniem rozkazów i pojedynczym strumieniem danych (ang. *multiple instruction single data stream* - MISD). Sekwencja danych jest transmitowana do zbioru procesorów, z których każdy wykonuje inną sekwencję rozkazów. Struktura taka nigdy nie była wdrożona.
- a Ze zwielokrotnionym strumieniem rozkazów i danych (ang. *multiple instruction multiple data stream* - MIMD). Zbiór procesorów jednocześnie wykonujących różne sekwencje rozkazów na różnych zbiorach danych.

W organizacji MIMD procesory są procesorami o ogólnym przeznaczeniu, ponieważ muszą móc przetwarzać wszystkie rozkazy konieczne do przeprowadzenia odpowiedniej transformacji danych. Systemy MIMD mogą być dalej podzielone zależnie od środków komunikowania się procesorów (rys. 16.15). Jeśli procesory używają wspólnej pamięci, to każdy z nich ma dostęp do programów

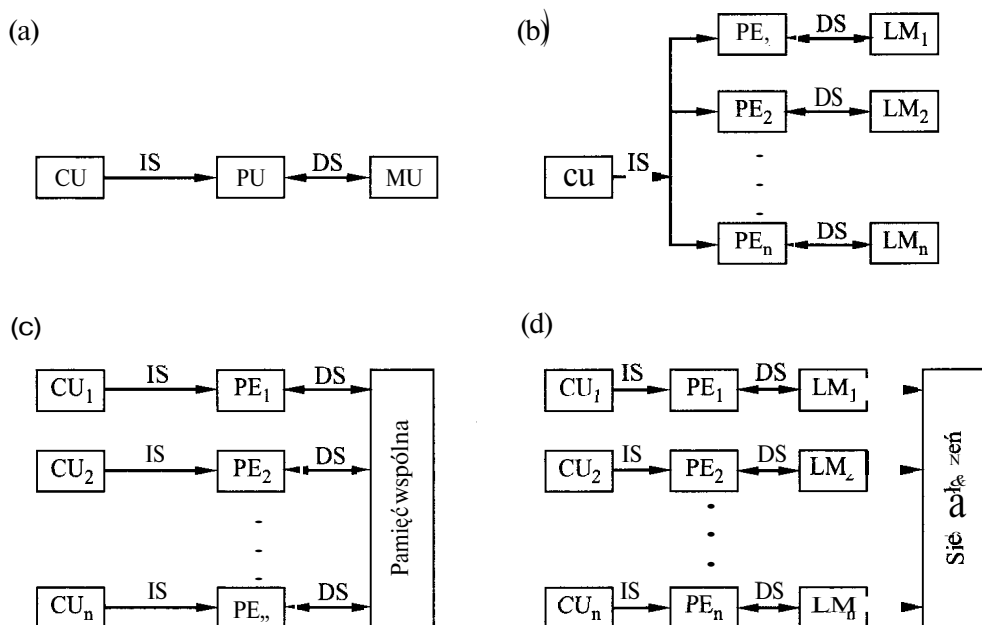


RYСУNEK 16.15. Praktyczne rozwiązania procesorów równoległych

i danych przechowywanych w tej pamięci, a procesory porozumiewają się między sobą za pośrednictwem tej pamięci; system taki nazywa się **wieloprocesorem**. Wieloprocesory zostały omówione w podrozdz. 16.1 i 16.2.

Jeżeli każdy procesor ma wydzieloną pamięć, to każdy element przetwarzający jest wydzielonym komputerem. Komunikacja między komputerami jest prowadzona albo poprzez stałe ścieżki, albo przez pewien mechanizm przełączania wiadomości. System taki nazywa się **wielokomputerem**.

Praktyczne systemy przetwarzania równoległego obejmują SIMD, wieloprocesory i wielokomputery. Na rysunku 16.16 jest pokazana ogólna klasyfikacja procesów równoległych. Najpierw, jako podstawę, na rys. 16.16a zilustrowano ogólną organizację SISD. Istnieje tu pewien rodzaj jednostki sterującej doprowadzającej strumień rozkazów (IS) do jednostki przetwarzającej. Jednostka ta pracuje na pojedynczym strumieniu danych (DS) pochodzącym z pamięci. W przypadku SIMD jest nadal jedna jednostka sterująca, kierująca teraz pojedynczy strumień danych do wielu elementów przetwarzających. Każdy element przetwarzający ma własną, wydzieloną pamięć (pokazaną na rys. 16.16b) lub może występować pamięć wspólna. W przypadku MIMD istnieje wiele jednostek sterujących, z których każda kieruje oddzielny strumień rozkazów do własnego elementu przetwarzającego. System MIMD może być wieloprocesorem o wspólnej pamięci (rys. 16.16c) lub wielokomputerem o rozproszonej pamięci (rys. 16.16d).



RYSunEK 16.16. Alternatywne organizacje komputerów: (a) SISD; (b) SIMD (z pamięcią rozproszoną); (c) MIMD (z pamięcią wspólną); (d) MIMD (z pamięcią rozproszoną)

Wieloprocesory

W przypadku systemu MIMD o pamięci wspólnej wszystkie procesory mają dostęp do tego samego obszaru pamięci wspólnej, zwykle za pośrednictwem szybkiej magistrali. Komunikacja między procesami jest w zasadzie łatwa. Każdy procesor może pozostawić komunikat lub wyniki w określonej lokacji, a następnie przekazać innemu procesorowi adres, pod którym może on znaleźć dane. Jest to typowe dla wieloprzetwarzania, w odróżnieniu od wieloprocesorowości. Jak widzieliśmy w rozdz. 7, w przypadku wieloprzetwarzania system komputerowy pracuje nad jednym lub wieloma programami aplikacyjnymi, które są dzielone na pewną liczbę współdziałających procesów sekwencyjnych. Schemat taki może być wdrożony w systemie jednoprocessorowym, jednak może być również łatwo zrealizowany w wieloprocesorze. W określonej chwili każdy z wielu procesorów wykonuje oddzielny proces. Komunikacja między procesami jest realizowana przez procesory za pośrednictwem pamięci głównej.

W praktyce wymagania wydajnościowe komplikują problem komunikowania się procesorów. Gdy występuje wiele szybkich procesorów rywalizujących o dostęp do tej samej pamięci przez tę samą magistralę, współzawodnictwo to może poważnie obniżyć ogólną wydajność. Jak widzieliśmy, rozwiązaniem jest dodanie do każdego procesora lokalnej pamięci podręcznej. Rozwiązanie to stwarza nowy problem spójności pamięci podręcznych, a do jego rozwiązania potrzebny jest protokół, taki jak MESI. Koordynacja między procesorami wymagana do egzekwowania protokołu MESI powoduje z kolei powstawanie wąskiego gardła w postaci wspólnej magistrali. Rezultat jest taki, że typowe systemy wieloprocesorowe są ograniczone do kilkudziesięciu procesorów. Wieloprocesor o wspólnej pamięci zawierający tysiące procesorów nie wydaje się być praktyczny.

Wielokomputery

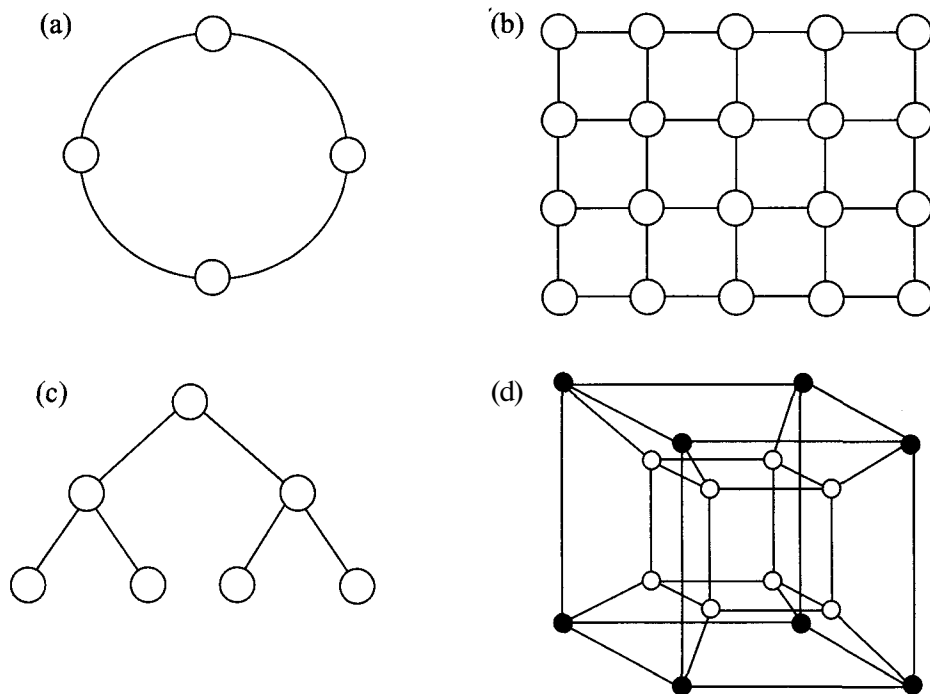
W przypadku systemu MIMD o pamięci rozproszonej każdy procesor ma własną przestrzeń pamięciową, która nie jest widzialna dla innych procesorów. Wobec tego wyniki i informacje koordynacyjne muszą być przekazywane między węzłami sieci połączeń, zwykle w postaci sformatowanych wiadomości.

Jednym z głównych powodów rozwoju organizacji wielokomputerowych jest pokonanie ograniczeń skali, jakie wiążą wieloprocesory. Celem jest opracowanie organizacji *skalowalnej*, w której może występować szeroki zakres liczby procesorów.

Ponieważ procesory w wielokomputerze muszą się komunikować wymieniając komunikaty, kluczowym elementem jest zaprojektowanie sieci wzajemnych połączeń, która musi funkcjonować tak efektywnie, jak tylko jest to możliwe. Istnieje na ogół wymiennosc między najdłuższą ścieżką między węzłami a liczbą połączeń fizycznych wymaganych w każdym węźle. Badano różne to-

pologie połączeń umożliwiające skalowalność i zapewniające wysoką wydajność. Na rysunku 16.17 są pokazane następujące topologie:

- **Pierścień.** Jeśli komunikacja wzdłuż pierścienia jest dwukierunkowa, to maksymalna odległość między dwoma dowolnymi węzłami n -węzłowego pierścienia wynosi $n/2$. Zwykle są używane pakiety wiadomości o ustalonym wymiarze, zawierające adres miejsca przeznaczenia. Taka topologia jest odpowiednia dla stosunkowo niewielkiej liczby procesorów przy minimalnej komunikacji danych.
- a **Siatka.** Najprostszą formą siatki jest dwuwymiarowa matryca, w której każdy węzeł jest połączony z czterema sąsiadami. Zasięg komunikacji w prostej sieci wynosi $2(n - 1)$. Połączenia skrótowe na krawędziach siatki redukują ten zasięg do $2(n/2)$. Organizacja siatkowa jest dobrze dostosowana do realizowania algorytmów zorientowanych na macierze.
- a **Drzewo.** Sieć o topologii drzewiastej była badana pod kątem wspierania algorytmów typu „dziel i rządź”, takich jak przeszukiwanie i sortowanie. Na rysunku pokazano proste drzewo binarne.
- a **Hipersześcián.** Topologia hipersześcianna używa $N = 2^n$ procesorów ułożonych w postaci n -wymiarowego sześcianu, w którym każdy węzeł ma $n = \log_2 N$ dwukierunkowych połączeń z sąsiednimi węzłami. Zasięg komunikacyjny takiego hipersześciánu wynosi n .



RYСУNEK 16.17. Topologie połączeń MIMD: (a) pierścień; (b) siatka; (c) drzewo; (d) hipersześcián

W przypadku topologii układowej, topologia hipersześcienna jest być może najbardziej atrakcyjna. Jak widać w tabeli 16.3, hipersześcienn jest odpowiedni do skalowania: zasięg systemu wzrasta powoli wraz ze wzrostem liczby węzłów w porównaniu z siatką.

TABELA 16.3. Łączność w warunkach najgorszego przypadku

Liczba węzłów	Siatka	Hipersześcienn
16	6 skoków	4 skoki
256	30 skoków	8 skoków
1 024	62 skoki	10 skoków
2 048	126 skoków	11 skoków
16 384	254 skoki	16 skoków

Ostatnio wzrasta zainteresowanie odejściem od topologii ustalonych w kierunku topologii wybieranych przez użytkownika, sterowanych za pomocą sieci tras. Zamiast łączyć wszystkie procesory razem, łączy się je za pomocą sieci szybkich tras, w której używa się przełączników do nawiązywania i przerywania połączeń wirtualnych, podobnie jak w sieci z komutacją pakietów. Jeśli przełączniki zaprojektowano tak, aby powodowały minimalne opóźnienia wiadomości, to zwłoka w komunikacji będzie wzrastała powoli przy dodawaniu do systemu nowych węzłów. Inną atrakcyjną własnością jest to, że każdy procesor potrzebuje jedynie prostego, dwukierunkowego połączenia z siecią przełączającą.

16.5

POLECANA LITERATURA

W [CATA94] dokonano przeglądu podstaw wieloprocesorów i szczegółowo przeanalizowano wieloprocesory oparte na SPARC. Wieloprocesory omówiono również w [MOLD93], [MORS94] i [STON93].

Doskonały przegląd problemów związanych ze spójnością pamięci podręcznych w wieloprocesorach zawiera [LILJ93]. W [TOMA93] zamieszczono przedruki wielu kluczowych artykułów na ten temat.

Wartościowe omówienie obliczeń wektorowych można znaleźć w pracach [STON93] i [HWAN93].

CATA94 Catanzaro B.: *Multiprocessor System Architectures*. Mountain View, Sunsoft Press, 1994.

HWAN93 Hwang K.: *Advanced Computer Architecture*. New York, McGraw-Hill, 1993.

LILJ93 Lilja D.: „Cache Coherence in Large-Scale Shared-Memory Multiprocessors: Issues and Comparisons”. *ACM Computing Surveys*, September 1993.

MOLD93 Moldovan D.: *Parallel Processing: from Applications to Systems*. San Mateo, Morgan Kaufmann, 1993.

MORS94 Morse H.: *Practical Parallel Computing*. Boston, AP Professional, 1994.

STON93 Stone H.: *High-Performance Computer Architecture*. Reading, Addison-Wesley, 1993.

TOMA93 Tomasevic M., Milutinovic V.: *The Cache Coherence Problem in Shared-Memory Multiprocessors: Hardware Solutions*. Los Alamitos, IEEE Computer Society Press, 1993.

16.6

PROBLEMY DO ROZWIĄZANIA

- 16.1. W wieloprocessorowej konfiguracji IBM S/370 każdemu procesorowi jest przypisany oddzielny, 4-kilobajtowy blok pamięci bezwzględnej przeznaczony do przechowywania pierwszego 4-kilobajowego bloku pamięci rzeczywistej. Zauważ, że mechanizm przedrostkowania nie zapobiega temu, żeby procesor X dokonał zapisu w obszarze pamięci wydzielonym dla procesora Y. Jaki inny mechanizm może być użyty do zapewnienia tej ochrony?
- 16.2. Czy możesz przewidzieć jakiś problem, który może zaistnieć w razie zastosowania pamięci podręcznej z jednokrotnym zapisem w wieloprocessorach o organizacji magistralowej? Jeśli tak, zasugeruj rozwiązanie.
- 16.3. Napisz zwektoryzowaną wersję następującego programu:

```
DO 20 I = 1, N
  B(I, 1) = 0
  DO 10 J = 1, M
    A(I) = A(I) + B(I,J) * C(I,J)
  10 CONTINUE
  D(I) = E(I) + A(I)
20 CONTINUE
```

- 16.4. Rozważ następującą metodę sortowania N pozycji:

```
Procedurę QUICKSORT (LOW; HIGH, A);
Array A[1:N], integer LOW; HIGH;
begin integer PIVOTPOINT;
  comment: QUICKSORT sortuje część A od A[LOW] do A[HIGH];
  call PARTITION (LOW, HIGH, A, PIVOTPOINT);
  comment: PARTITION jest procedurą, która przemieszcza elementy
    w A od A[LOW] do A[HIGH] w ten sposób, że dla wszystkich I
    w zakresie  $LOW \leq I < PIVOTPOINT$ ,  $A[I] \leq A[PIVOTPOINT]$ ,
    oraz dla wszystkich J w zakresie  $PIVOTPOINT < J < HIGH$ ,
     $A[PIVOTPOINT] \leq A[J]$ . PIVOTPOINT jest wskaźnikiem wybieranym
    przez PARTITION i zwracany przez PIVOTPOINT;
  if PIVOTPOINT > LOW + 1 then QUICKSORT (LOW, PIVOTPOINT - 1, A);
  if PIVOTPOINT < HIGH - 1 then QUICKSORT (PIVOTPOINT + 1, HIGH, A);
end of QUICKSORT;
```

- (a) Wyjaśnij pokrótce działanie procedury QUICKSORT, żeby wykazać, że rozumiesz, iż rozkaz ten sortuje A, jeśli został wywołany początkowo za pomocą dyrektywy QUICKSORT (1, N, A).
- (b) Zrewiduj QUICKSORT przez wstawienie zgodnie z potrzebą instrukcji FORK i odpowiednich instrukcji JOIN. Wskaż dokładnie, jaka informacja musi być przeniesiona do przodu z danych globalnych w przypadku każdego nowego rozwidlenia, a jaka informacja musi być przetwarzana w sposób lokalny w przypadku tego rozwidlenia.
- (c) Załóż, że użycie instrukcji FORK i JOIN zajmuje mniej więcej tyle samo czasu, ile zużywa procedura QUICKSORT do sortowania tablicy o długości 7 w komputerze szeregowym. Oszacuj względny czas przebiegu sortowania 1023 pozycji na komputerze szeregowym oraz za pomocą wieloprocesorów o 2, 3, 4, 8 i 16 procesorach. Załóż przy tym, że jakimś cudem PARTITION zawsze znajduje PIVOTPOINT dokładnie w środku zakresu. Czas PIVOTPOINT jest proporcjonalny do HIGH - LOW.
- (d) Opisz, jak można zmodyfikować procedurę QUICKSORT, żeby była ona w stanie rozpoznać za pomocą wywołań systemowych, ile procesorów znajduje się w systemie i ile z nich jest wolnych, a następnie użyć tej informacji do zoptymalizowania swojej wydajności.

16.5. Wewnętrzna pętla obliczenia macierzowego używa instrukcji:

```
for J: = K step until N do
  A[I, J]: = A[I, J] × Q – A[K, J];
```

Załącz, że ta instrukcja jest rozłożona na procesory systemu wieloprocesorowego w ten sposób, że każdy procesor wykonuje instrukcję dla innej wartości J. Załącz również, że system wieloprocesorowy ma scentralizowaną, wspólną pamięć. Gdy dwa różne procesory sięgają jednocześnie do tego samego modułu pamięci, moduł odpowiada natychmiast jednemu procesorowi, a drugi procesor musi czekać przez jeden cykl pamięci, podczas którego powtarza swoje zapotrzebowanie.

- (a) Załącz, że tablica jest przechowywana w postaci wierszy w modułach pamięci w ten sposób, że pod rosnącymi adresami znajdują się elementy o parach wskaźnikowych (1, 1), (1, 2), ..., (1, N), (2, 1), ..., (2, N), ..., (N, N) w miarę, jak rosnące adresy obiegają moduły pamięci.

Narysuj diagram obrazujący cykl po cyklu wykonywanie pierwszych pięciu pakietów obliczeniowych przy założeniu, że wszystkie z nich są inicjowane równocześnie i że liczba pamięci wynosi N.

- (b) Teraz załóż, że tablica jest przechowywana w ten sposób, że każdy wiersz A znajduje się w oddzielnym module pamięci. Powtórz część (a).
- (c) Skomentuj problem rywalizacji danych w tego rodzaju systemie pamięci **wspólnej**, korzystając ze swoich obserwacji.

Cyfrowe układy logiczne

Działanie komputera cyfrowego jest oparte na przechowywaniu i przetwarzaniu danych binarnych. W całej tej książce zakładaliśmy istnienie elementów pamięciowych występujących w jednym z dwóch stanów stabilnych oraz układów operujących na danych binarnych pod kontrolą sygnałów sterujących w celu realizacji różnych funkcji komputera. W tym dodatku powiemy, w jaki sposób te elementy i układy pamięciowe mogą posłużyć do realizacji cyfrowych układów logicznych, zarówno kombinacyjnych, jak i sekwencyjnych. Rozpocniemy od krótkiego przeglądu algebry Boole'a, stanowiącej matematyczną podstawę cyfrowych układów logicznych. Następnie wprowadzimy pojęcie bramki. Na zakończenie opiszemy układy kombinacyjne i sekwencyjne zbudowane z bramek.

A.1

ALGEBRA BOOLE'A

Przy projektowaniu i analizowaniu układów cyfrowych w komputerach i w innych systemach cyfrowych używana jest gałąź matematyki zwana **algebrą Boole'a**. Nazwano ją tak dla uczczenia angielskiego matematyka George'a Boole'a, który zaproponował podstawowe zasady tej algebry w traktacie zatytułowanym *An Investigation of the Laws of Thought on Which to Found the Mathematical Theories of Logic and Probabilities* (*Badanie praw myśli, które mogą być podstawą matematycznych teorii logiki i prawdopodobieństwa*). W roku 1938 Claude Shannon, asystent na wydziale elektrycznym MIT, zasugerował zastosowanie algebry Boole'a do rozwiązywania problemów projektowania układów przekaznikowych [SHAN38]. Metody Shannona zostały następnie użyte do analizowania i projektowania elektronicznych układów cyfrowych. Algebra Boole'a jest wygodnym narzędziem w dwóch obszarach:

- *analizy* - jest ekonomicznym sposobem opisywania działania układów cyfrowych;
- a *projektowania* - algebra Boole'a może być stosowana do uproszczonej realizacji pożądaných funkcji.

Podobnie jak inne rodzaje algebry, algebra Boole'a używa zmiennych i operacji. W tym przypadku są to logiczne zmienne i operacje. Wobec tego zmienna może przyjmować wartość 1 (PRAWDA) lub 0 (FAŁSZ). Podstawowymi operacjami logicznymi są AND (I), OR (LUB) i NOT (NIE), reprezentowane symbolicznie przez kropkę, znak plus oraz kreskę nad zmienną:

$$A \text{ AND } B = A \cdot B$$

$$A \text{ OR } B = A + B$$

$$\text{NOT } A = \bar{A}$$

Operacja AND daje w wyniku prawdę (wartość binarną 1) wtedy i tylko wtedy, gdy jej obydwa argumenty są prawdziwe. Operacja OR daje w wyniku prawdę, gdy którykolwiek z jej argumentów lub obydwa są prawdziwe. Operacja jednoargumentowa NOT odwraca wartość swojego argumentu. Rozważmy na przykład równanie

$$D = A + (\bar{B} \cdot C)$$

D jest równe 1, jeżeli A jest 1 lub jeżeli jednocześnie $B = 0$ i $C = 1$. W przeciwnym razie D jest równe 0.

Oto kilka uwag dotyczących notacji. Przy braku nawiasów operacja AND ma pierwszeństwo przed operacją OR. Gdy nie grozi to dwuznacznością, operację AND zapisuje się przez kolejne wypisywanie argumentów z pominięciem operatora w postaci kropki. Wobec tego

$$A + B \cdot C = A + (B \cdot C) = A + BC$$

Wszystkie te wyrażenia oznaczają: oblicz AND z B i C; następnie oblicz OR z wyniku i z A.

W tabeli A.1 są zdefiniowane podstawowe operacje logiczne w postaci tablicy **prawdy**, w której po prostu wymienia się wartości operacji dla każdej możliwej kombinacji wartości argumentów. W tabeli są również wymienione trzy inne użyteczne operatory: XOR (LUB wykluczające), NAND (NIE-I) oraz NOR (NIE-LUB). Operacja LUB wykluczające dwóch argumentów logicznych daje w wyniku 1 wtedy i tylko wtedy, gdy dokładnie jeden z argumentów ma wartość 1. Funkcja NAND jest dopełnieniem (NOT) funkcji AND, a NOR jest dopełnieniem OR:

$$A \text{ NAND } B = \text{NOT}(A \text{ AND } B) = \overline{AB}$$

$$A \text{ NOR } B = \text{NOT}(A \text{ OR } B) = \overline{A + B}$$

Jak zobaczymy, te trzy nowe operacje mogą być przydatne przy realizacji pewnych cyfrowych układów logicznych.

TABELA A.1. Operatory boolowskie

P	Q	NOT P	P AND Q	P OR Q	P XOR Q	P NAND Q	P NOR Q
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0

W tabeli A.2 są podane podstawowe tożsamości algebry Boole'a. Równania ułożono w dwóch kolumnach w celu pokazania komplementarnej (lub dualnej) natury operacji AND i OR. Występują tu dwie klasy tożsamości: reguły podstawowe (zwane postulatami), które są podawane bez dowodu, oraz tożsamości pozostałe, które mogą być wyprowadzone na podstawie postulatów podstawowych. Postulaty definiują sposób interpretowania wyrażeń Boole'a. Warto zwrócić uwagę na jedno z dwóch praw rozdzielczości, ponieważ różni się ono od tego, które znajdujemy w zwykłej algebrze:

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

Dwa ostatnie wyrażenia w tabeli noszą nazwę prawa De Morgana. Można je również zapisać w następujący sposób:

$$A \text{ NOR } B = \overline{A} \text{ AND } \overline{B}$$

$$A \text{ NAND } B = \overline{A} \text{ OR } \overline{B}$$

Jako ćwiczenie zalecamy zweryfikowanie wyrażeń w tabeli A.2 przez podstawianie rzeczywistych wartości (1 i 0) zamiast zmiennych A, B i C.

TABELA A.2. Podstawowe tożsamości algebry Boole'a

Postulaty podstawowe	
$A + B = B + A$	Prawa przemienności
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	Prawa rozdzielczości
$A + A = A$	Prawa tożsamości
$A \cdot A = A$	Prawa odwrotności

Jak zobaczamy, te trzy nowe operacje mogą być przydatne przy realizacji pew-
nych cyfrowych układów logicznych.

$$A \text{ NOR } B = \text{NOT}(A \text{ OR } B) = \overline{A + B}$$

$$A \text{ NAND } B = \text{NOT}(A \text{ AND } B) = \overline{A \cdot B}$$

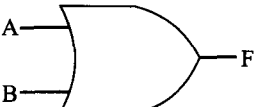
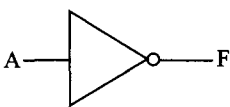
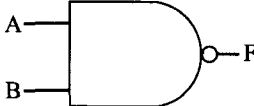
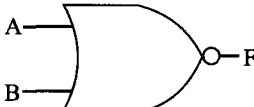
de Shannon, asystent na wydziale elektrycznym MIT, zasugerował użycie algebry Boole'a do rozwiązywania problemów projektowania układów przekazy-
nikowych [SHAN38]. Metody Shannona zostały następnie użyte do analizowa-
nia i projektowania elektronicznych układów cyfrowych. Algebra Boole'a jest
wygodnym narzędziem w dwóch obszarach:

NAND i NOR. Na rysunku A.1 jest pokazanych tych pięć bramek. Każda z nich została zdefiniowana za pomocą trzech sposobów: symbolu graficznego, notacji algebraicznej i tablicy prawdy. Użyta tutaj i w całym dodatku symbolika jest zgodna z normą IEEE Std 91 [IEEE84]. Zauważmy, że operacja inwersji (NOT) jest oznaczana za pomocą kółka.

Każda z tych bramek ma jedno lub dwa wejścia i jedno wyjście. Gdy zmieniają się wartości na wejściu, prawidłowy sygnał wyjściowy pojawia się niemal natychmiast, po opóźnieniu wynikającym tylko z propagacji sygnałów przez bramkę (zwanym **opóźnieniem bramkowym**). Znaczenie tego omówimy w podrozdz. A.3.

Poza bramkami pokazanymi na rys. A.1, mogą być używane bramki o 3, 4 i o większej liczbie wejść. Dzięki temu wyrażenie $X + Y + Z$ można zrealizować za pomocą jednej bramki OR o 3 wejściach.

Zwykle nie wszystkie bramki są używane do implementacji. Projektowanie i wytwarzanie są prostsze, jeśli używa się tylko jednego lub dwóch rodzajów bramek. Ważne jest więc wyznaczenie funkcjonalnie pełnych zbiorów bramek.

Nazwa	Symbol graficzny	Funkcja algebraiczna	Tablica prawdy															
AND		$F = A \cdot B$ lub $F = AB$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = A + B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$F = \overline{A}$ lub $F = A'$	<table><tr><th>A</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	F	0	1	1	0									
A	F																	
0	1																	
1	0																	
NAND		$F = \overline{(AB)}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	1	0	1	1	1	0	1	1	1	1
A	B	F																
0	0	1																
0	1	1																
1	0	1																
1	1	1																
NOR		$F = \overline{(A + B)}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	0
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																

RYСУNEK A.1. Podstawowe bramki logiczne

Oznacza to, że dowolna funkcja Boole'a może być zrealizowana za pomocą wyłącznie bramek pochodzących z takiego zbioru. Następujące zbiory są funkcjonalnie pełne:

- AND, OR, NOT
- a AND, NOT
- a OR, NOT
- a NAND
- a NOR

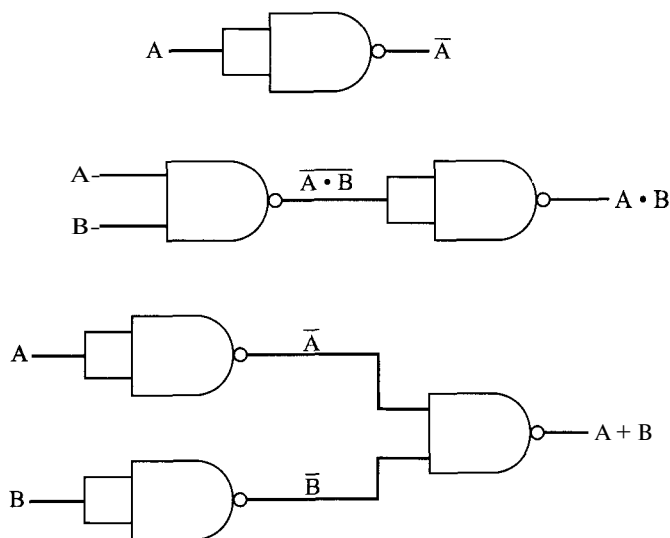
Powinno być oczywiste, że bramki AND, OR i NOT tworzą zbiór funkcjonalnie pełny, ponieważ reprezentują one trzy operacje algebry Boole'a. Aby bramki AND i NOT miały stanowić zbiór funkcjonalnie pełny, musi istnieć sposób syntetyzowania operacji OR za pomocą operacji AND i NOT. Można to uczynić, posługując się prawem De Morgana:

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

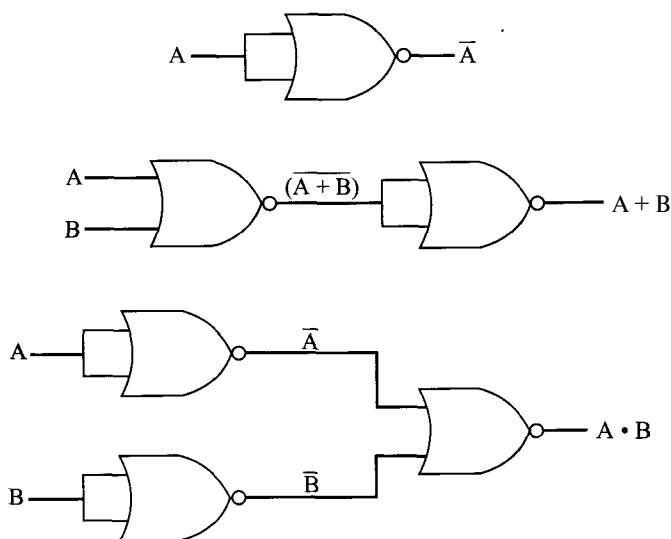
$$A \text{ OR } B = \text{NOT}((\text{NOT } A) \text{ AND } (\text{NOT } B))$$

Podobnie operacje OR i NOT stanowią zbiór funkcjonalnie pełny, ponieważ mogą one być użyte do syntetyzowania operacji AND.

Na rysunku A.2 widać, w jaki sposób funkcje AND, OR i NOT mogą być zrealizowane wyłącznie za pomocą bramek NAND, a na rys. A.3 jest pokazane to samo za pomocą bramek NOR. Z tego powodu układy cyfrowe mogą być (i często są) realizowane wyłącznie za pomocą bramek NAND lub wyłącznie NOR.



RYSUNEK A.2. Zastosowanie bramek NAND



RYSUNEK A.3. Zastosowanie bramek NOR

Zajmując się bramkami, sięgnęliśmy do najbardziej podstawowego poziomu nauki i techniki komputerowej. Analizowanie kombinacji tranzystorów używanych do budowy bramek wykracza poza ten zakres i wchodzi w obszar elektroniki. Zainteresowanym czytelnikom można polecić pracę [CLAR80] będącą zajmującym a jednocześnie treściwym opisem tych zagadnień przeznaczonym dla specjalistów komputerowych. Dla naszych celów wystarczy jednak opis sposobu używania bramek jako podstawowych elementów służących do budowy podstawowych układów logicznych komputera cyfrowego.

A.3

UKŁADY KOMBINACYJNE

Układ kombinacyjny jest zbiorem wzajemnie połączonych bramek, którego stan wyjść w dowolnej chwili jest wyłącznie funkcją stanu wejść w tej samej chwili. Podobnie jak w przypadku pojedynczej bramki, po ustaleniu stanu na wejściu prawie natychmiast pojawia się sygnał na wyjściu, przy czym występują tylko opóźnienia bramkowe.

Ogólnie rzecz biorąc, układ kombinacyjny zawiera n wejść i m wyjść binarnych. Podobnie jak bramka, układ kombinacyjny może być zdefiniowany przez podanie:

- a *Tablicy prawdy*. Dla każdej z 2^n możliwych kombinacji sygnałów wejściowych podana jest wartość binarna każdego z m sygnałów wyjściowych.
- a *Symbolu graficznego*. Przedstawiony jest schemat połączeń bramek.
- a *Równania Boole'a*. Każdy sygnał wyjściowy jest wyrażony jako funkcja Boole'a sygnałów wejściowych.

Realizacja funkcji Boole'a

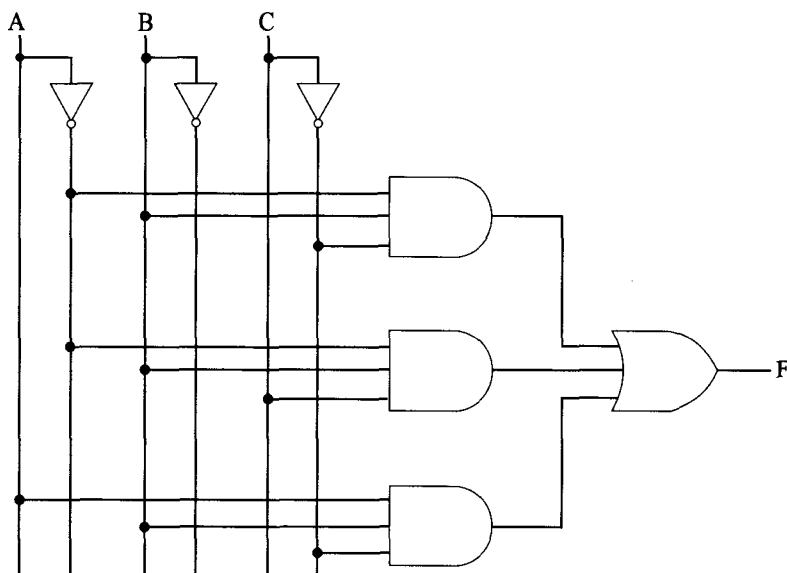
Dowolna funkcja Boole'a może być zrealizowana w postaci elektronicznej jako sieć bramek. Dla danej funkcji istnieje pewna liczba rozwiązań alternatywnych. Rozważmy funkcję Boole'a reprezentowaną przez tablicę prawdy w tabeli A.3. Możemy wyrazić tę funkcję po prostu przez wyszczególnienie kombinacji wartości A, B i C, które powodują, że F jest równe 1:

$$F = \overline{A}\overline{B}\overline{C} + \overline{A}BC + A\overline{B}\overline{C} \quad (\text{A-1})$$

TABELA A.3. Funkcja Boole'a trzech zmiennych

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Istnieją trzy kombinacje wartości wejściowych, które powodują, że F jest równe 1, i jeśli wystąpi którakolwiek z tych kombinacji, to wynik jest równy 1. Ta forma wyrażenia z oczywistych powodów nazywa się **sumą iloczynów**. Na rysunku A.4 jest pokazana bezpośrednia realizacja za pomocą bramek AND, OR i NOT.



RYSunek A.4. Realizacja tabeli A.3 za pomocą sumy iloczynów

Inny sposób realizacji może być wyprowadzony na podstawie tablicy prawdy. Z postaci (A-1) wynika, że na wyjściu jest 1, jeżeli którakolwiek z kombinacji wejściowych, która daje w wyniku 1, jest prawdziwa. Możemy również powiedzieć, że wyjście jest równe 1, jeśli żadna z kombinacji wejściowych dających 0 nie jest prawdziwa. Wobec tego:

$$F = (\overline{ABC}) \cdot (\overline{AB\overline{C}}) \cdot (\overline{A\overline{B}C}) \cdot (\overline{A\overline{B}\overline{C}}) \cdot (\overline{ABC})$$

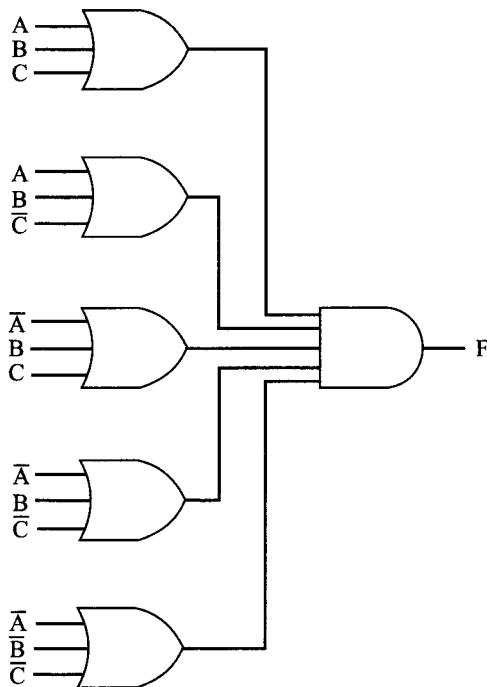
Można to przekształcić, stosując uogólnienie prawa De Morgana:

$$(\overline{X \cdot Y \cdot Z}) = \overline{X} + \overline{Y} + \overline{Z}$$

Wobec tego

$$\begin{aligned} F &= (\overline{A+B+C}) \cdot (\overline{A+B+\overline{C}}) \cdot (\overline{A+\overline{B}+C}) \cdot (\overline{A+\overline{B}+\overline{C}}) \cdot (\overline{A+B+C}) = \\ &= (A+B+C) \cdot (A+B+\overline{C}) \cdot (\overline{A+B+C}) \cdot (\overline{A+B+\overline{C}}) \cdot (\overline{A+B+C}) \end{aligned} \quad (\text{A-2})$$

Jest to postać iloczynu sum, zilustrowana na rys. A.5. Dla jasności nie pokazano bramek NOT. Zamiast tego założono, że osiągalne są: każdy sygnał wejściowy i jego dopełnienie. Upraszcza to schemat logiczny i uwidacznia wejścia bramek.



RYSUNEK A.5. Realizacja tabeli A.3 za pomocą iloczynu sum

Funkcja Boole'a może więc być zrealizowana albo w postaci sumy iloczynów albo iloczynu sum. W tym momencie mogłoby się wydawać, że wybór powinien zależeć od tego, czy tablica prawdy zawiera więcej 1, czy 0 po stronie funkcji wyjściowej. Zastosowanie sumy iloczynów oznacza jedno wyrażenie dla każdej 1, zaś zastosowanie iloczynu sum – jedno wyrażenie dla każdego 0. Należy jednak wziąć pod uwagę, że:

- Na ogół jest możliwe wyprowadzenie prostszego wyrażenia Boole'a na podstawie tablicy prawdy niż na podstawie sumy iloczynów czy iloczynu sum.
- a Może być bardziej pożądane zrealizowanie funkcji za pomocą bramek jednego rodzaju (NAND lub NOR).

W odniesieniu do pierwszego punktu ważne jest to, że przy prostszym wyrażeniu Boole'a do realizacji funkcji będzie potrzebnych mniej bramek. W celu uproszczenia mogą być stosowane następujące trzy metody:

- a upraszczanie algebraiczne,
- mapy Karnaugh'a,
- a tablice Quine'a-McKluskeya.

Upraszczenie algebraiczne

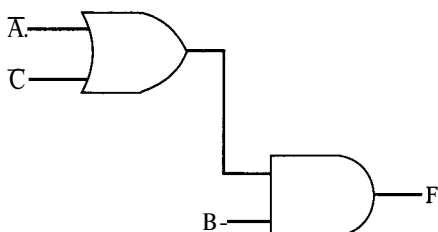
Upraszczenie algebraiczne polega na stosowaniu tożsamości przedstawionych w tabeli A.2 do redukcji liczby elementów w wyrażeniach Boole'a. Rozważmy na przykład ponownie równanie (A-1). Równoważnym wyrażeniem jest

$$F = \overline{A}B + B\overline{C} \quad (\text{A-3})$$

lub nawet prościej

$$F = B(\overline{A} + \overline{C})$$

Wyrażenie to może być zrealizowane w sposób pokazany na rys. A.6. Uproszczenie równania (A-1) zostało wykonane w zasadzie w wyniku obserwacji. W przypadku bardziej złożonych wyrażeń potrzebne jest podejście bardziej systematyczne.

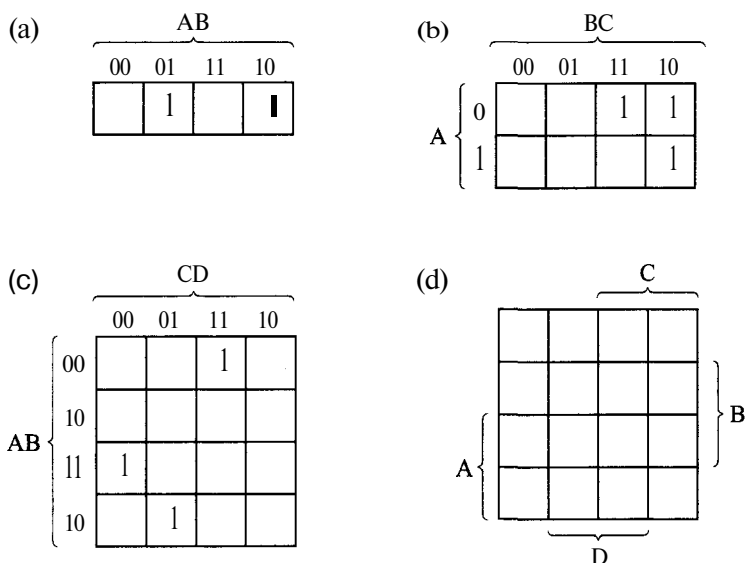


RYSUNEK A.6. Uproszczona realizacja tabeli A.3

Mapy Karnaugh

Mapa Karnaugh jest wygodną metodą upraszczania funkcji Boole'a o niewielkiej liczbie zmiennych ($4+6$). Mapa jest tablicą złożoną z 2^n kwadratów reprezentujących możliwe kombinacje wartości n zmiennych binarnych. Na rysunku A.7a widać mapę złożoną z czterech kwadratów, odnoszącą się do funkcji dwóch zmiennych. Dla celów wyjaśnionych w dalszym ciągu wygodne jest wymienianie kombinacji wartości zmiennych w porządku 00, 01, 11, 10. Ponieważ kwadraty odpowiadające kombinacjom mają być używane do zapisywania informacji, kombinacje te są zwyczajowo zapisywane nad kwadratami. W przypadku czterech zmiennych mapa składa się z 16 kwadratów ułożonych w sposób pokazany na rys. A.7c.

Mapa Karnaugh może być używana do reprezentowania funkcji Boole'a w następujący sposób. Każdy kwadrat odpowiada unikatowemu iloczynowi występującemu w sumie iloczynów, przy czym wartość 1 odpowiada zmiennej, a 0 odpowiada funkcji NOT tej zmiennej. Wobec tego iloczyn $\overline{A}\overline{B}$ odpowiada czwartemu kwadratowi na rys. A.7a. Dla każdego takiego iloczynu w odpowiednim kwadracie jest wstawiana 1. W przypadku dwóch zmiennych mapa ta odpowiada więc funkcji $\overline{A}\overline{B} + \overline{A}B$. Mając tablicę prawdy funkcji Boole'a, łatwo jest zbudować mapę: dla każdej kombinacji wartości zmiennych, która daje w wyniku 1 w tablicy prawdy, wstawia się 1 w odpowiednim kwadracie mapy.



RYSUNEK A.7. Zastosowanie map Karnaugh do reprezentowania funkcji Boole'a: (a) $F = \overline{A}\overline{B} + \overline{A}B$; (b) $F = \overline{A}\overline{B}C + \overline{A}BC + A\overline{B}C$; (c) $F = \overline{A}\overline{B}CD + \overline{A}B\overline{C}D + A\overline{B}C\overline{D}$; (d) uproszczone etykiety mapy

Na rysunku A.7b jest pokazany wynik odnoszący się do tablicy prawdy z tabeli A.3. Aby przekształcić wyrażenie Boole'a na mapę, konieczne jest najpierw przetworzenie tego wyrażenia na tzw. postać kanoniczną: każdy składnik wyrażenia musi zawierać każdą zmienną. Jeśli mamy na przykład równanie (A-3), to musimy najpierw rozszerzyć je do pełnej postaci równania (A-1), a następnie przekształcić na mapę.

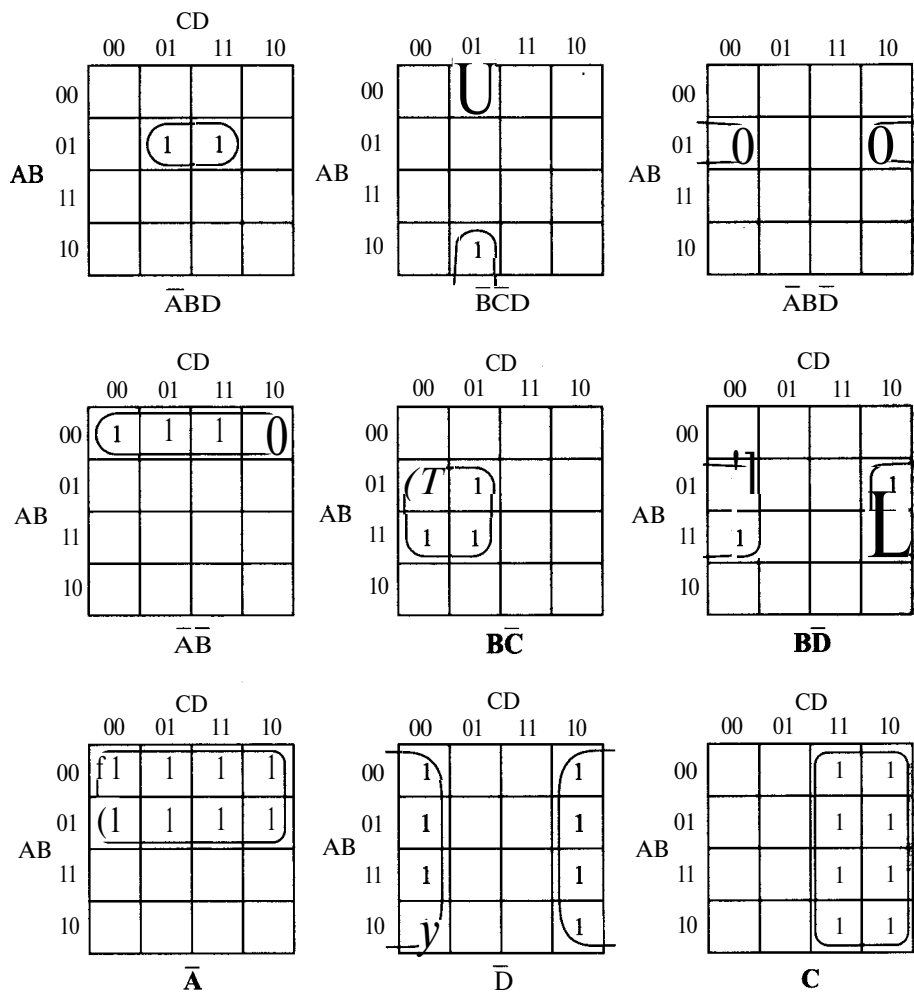
Oznaczenia użyte na rys. A.7d akcentują zależność między zmiennymi a wierszami i kolumnami mapy. Dwa wiersze objęte symbolem A to te wiersze, w których wartość A ma wartość 1; wiersze nie objęte przez A są tymi wierszami, w których zmienna A jest 0. Podobnie jest w przypadku B, C i D.

Gdy mapa funkcji jest już utworzona, możemy napisać proste wyrażenie algebraiczne, zwracając uwagę na układ jedynek na mapie. Zasada jest następująca. Dowolne dwa sąsiadujące ze sobą kwadraty różnią się tylko jedną ze zmiennych. Jeśli oba sąsiadujące kwadraty zawierają 1, to odpowiedni iloczyn różni się tylko jedną zmienną. W takim przypadku obydwa składniki mogą być połączone przez wyeliminowanie tej zmiennej. Na przykład na rys. A.8a dwa sąsiednie kwadraty odpowiadają dwóm składnikom $\overline{A}B\overline{C}D$ i $\overline{A}BCD$. Wobec tego funkcję możemy wyrazić następująco:

$$\overline{A}B\overline{C}D + \overline{A}BCD = \overline{A}BD$$

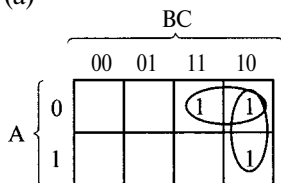
Proces ten może być rozszerzony na kilka sposobów. Po pierwsze, może być rozszerzona koncepcja przyległości przez uwzględnienie sąsiedztwa wokół krawędzi mapy. Wobec tego górny kwadrat kolumny jest sąsiedni w stosunku do dolnego kwadratu, a lewy kwadrat w wierszu sąsiaduje z prawym. Warunki te są zilustrowane na rys. A.8b i c. Po drugie, możemy zgrupować nie tylko 2 kwadraty, lecz 2ⁿ sąsiednich kwadratów, tzn. 4, 8 itd. Następne trzy przykłady na rys. A.8 pokazują grupowanie 4 kwadratów. Zauważmy, że w tym przypadku dwie spośród zmiennych mogą być wyeliminowane. W ostatnich trzech przykładach pokazano grupowanie 8 kwadratów, co umożliwia wyeliminowanie trzech zmiennych.

Dążąc do uproszczenia, przeanalizujemy najpierw największe możliwe grupowanie (8 w przypadku mapy o 4 zmiennych). Jeśli jakikolwiek z kwadratów zawierających 1 pozostaje *niezakreślony*, to analizujemy następne mniejsze grupy. Gdy zakreśla się grupy, dozwolone jest użycie tej samej jedynki więcej niż raz. Jest to zilustrowane na rys. A.9a opartym na tabeli A.3. Jeśli którakolwiek spośród izolowanych jedynek pozostaje poza ugrupowaniem, to każda z nich jest zakreślana jako grupa. Na zakończenie, zanim przejdziemy od mapy do uproszczonego wyrażenia Boole'a, możemy wyeliminować dowolną z grup jedynek, która w całości nakłada się z innymi grupami (patrz rys. A.9b). W tym przypadku pozioma grupa nakłada się z innymi i przy tworzeniu wyrażenia Boole'a może być zignorowana.

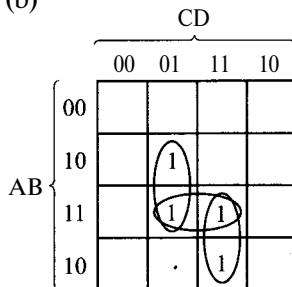


RYSUNEK A.8. Zastosowanie map Karnaugh

(a)



(b)

RYSUNEK A.9. Nakładające się grupy: (a) $F = \bar{A}\bar{B} + \bar{B}\bar{C}$; (b) $F = \bar{B}\bar{C}D + ACD$

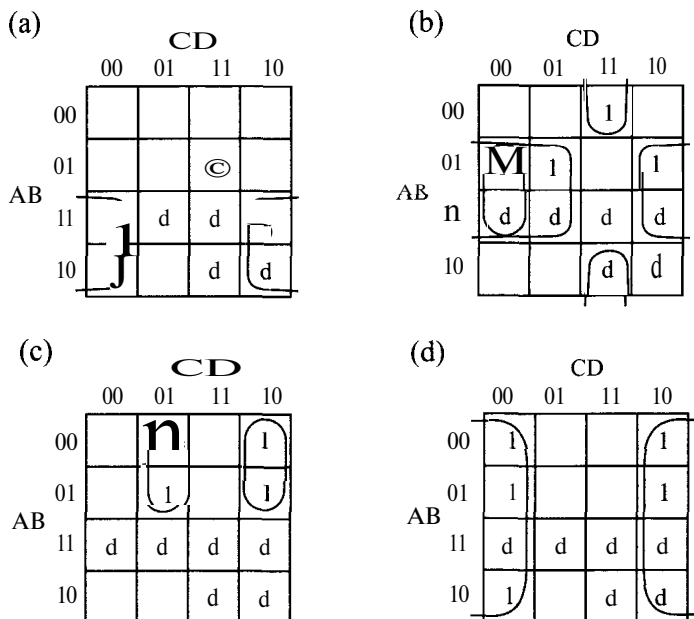
Warto zwrócić uwagę na pewną dodatkową własność map Karnaugh. W niektórych przypadkach pewne kombinacje wartości zmiennych nigdy nie występują i wobec tego nie występują odpowiadające im sygnały wyjściowe. Te warunki określa się jako „bez znaczenia” („don't care”). W przypadku każdego z takich warunków, do odpowiedniego kwadratu mapy wprowadza się literę „d”. Podczas grupowania i upraszczania każde „d” może być traktowane jako 1 lub 0, zależnie od tego, co prowadzi do prostszego wyrażenia.

Przykład przedstawiony w [HAYE88] ilustruje elementy, które przedyskutowaliśmy. Chcielibyśmy *znaleźć* wyrażenie Boole'a dla układu, który dodaje 1 do upakowanej cyfry dziesiętnej. Przypomnijmy sobie na podstawie podrozdz. 9.2, że w przypadku upakowanej notacji dziesiętnej każda cyfra dziesiętna jest reprezentowana za pomocą kodu 4-bitowego, tworzonego w zwykły sposób. Wobec tego 0 = 0000, 1 = 0001, ..., 8 = 1000 i 9 = 1001. Pozostałe wartości 4-bitowe, od 1010 do 1111, nie są używane. Kod ten jest również nazywany kodem BCD od ang. *Binary Coded Decimal*.

W tabeli A.4 znajduje się tablica prawdy odnosząca się do 4-bitowego wyniku, który jest o 1 większy niż 4-bitowe dane wejściowe BCD. Dodawanie to jest wykonywane modulo 10, więc $9 + 1 = 0$. Zauważmy również, że sześć spośród kodów wejściowych daje wyniki „bez znaczenia”, ponieważ nie są to ważne dane wejściowe BCD. Na rysunku A.10 jest pokazana wynikowa mapa Karnaugh dla każdej ze zmiennych wyjściowych. Kwadraty d zostały użyte w celu osiągnięcia możliwie najlepszych ugrupowań.

TABELA A.4. Tablica prawdy jednocyfrowego rejestru następnikowego w notacji upakowanej dziesiętnej

Liczba	Wejście				Liczba	Wyjście			
	A	B	C	D		W	X	Y	Z
0	0	0	0	0	1	0	0	0	1
1	0	0	0	1	2	0	0	1	0
2	0	0	1	0	3	0	0	1	1
3	0	0	1	1	4	0	1	0	0
4	0	1	0	0	5	0	1	0	1
5	0	1	0	1	6	0	1	1	0
6	0	1	1	0	7	0	1	1	1
7	0	1	1	1	8	1	0	0	0
8	1	0	0	0	9	1	0	0	1
9	1	0	0	1	0	0	0	0	0
Wartości „bez znaczenia”	1	0	1	0		d	d	d	d
	1	0	1	1		d	d	d	d
	1	1	0	0		d	d	d	d
	1	1	0	1		d	d	d	d
	1	1	1	0		d	d	d	d
	1	1	1	1		d	d	d	d



RYSUNEK A. 10. Mapy Karnaugh dla układu inkrementującego:

(a) $W = A\bar{D} + \bar{A}BCD$; (b) $X = B\bar{D} + B\bar{C} + \bar{B}CD$; (c) $Y = \bar{A}\bar{C}\bar{D} + \bar{A}C\bar{D}$; (d) $Z = \bar{D}$

Metoda Quine'a-McKluskeya

W przypadku więcej niż 4 zmiennych mapa Karnaugh staje się coraz bardziej kłopotliwa. Dla 5 zmiennych potrzebne są dwie mapy 16×16 , przy czym przy analizowaniu przyległości jedna z map jest rozpatrywana jako leżąca nad drugą w 3 wymiarach. Sześć zmiennych wymaga użycia 4 tablic 16×16 w 4 wymiarach! Podejściem alternatywnym jest metoda tablicowa określana jako metoda Quine'a-McKluskeya. Metoda ta nadaje się do programowania, dzięki czemu może być zautomatyzowanym narzędziem do minimalizowania wyrażeń Boole'a.

Najlepiej jest objaśnić tę metodę za pomocą przykładu. Rozważmy następujące wyrażenie:

$$ABCD + ABCD + AB\bar{C}\bar{D} + ABCD + ABCD + \bar{A}BCD + \bar{A}BCD + \bar{A}BCD$$

Założmy, że wyrażenie to zostało wyprowadzone na podstawie tablicy prawdy. Chcielibyśmy utworzyć wyrażenie minimalne odpowiednie do realizacji za pomocą bramek.

Pierwszym krokiem jest zbudowanie tablicy, w której każdy wiersz odpowiada jednemu z członów wyrażenia. Członcy są grupowane zależnie od liczby zanegowanych zmiennych. Rozpoczynamy od członku nie zawierającego zmiennych zanegowanych, jeśli taki istnieje, następnie bierzemy pod uwagę wszystkie

iloczynu z jedną zmienną zanegowaną i tak dalej. W tabeli A.5 jest pokazana lista dotycząca naszego przykładu, przy czym wiersze tabeli pokazują grupowanie. Dla jasności, każdy człon jest reprezentowany przez 1 odpowiadającą każdej zmiennej niezanegowanej oraz przez 0 dla każdej zanegowanej. Grupujemy więc człony zależnie od liczby jedynek, jaką zawierają. Wskaźnik kolumny jest po prostu równoważnikiem dziesiętnym i będzie użyteczny w dalszym ciągu.

TABELA A.5. Pierwszy etap metody Quine'a-McKluskeya dla wyrażenia
 $F = ABCD + \overline{A}BCD + ABC\overline{D} + \overline{A}BC\overline{D} + \overline{A}BCD + \overline{A}BC\overline{D} + \overline{A}BC\overline{D} + \overline{A}BC\overline{D}$

Iloczyn	Indeks	A	B	C	D	
$\overline{A}BCD$	1	0	0	0	1	V
$\overline{A}BC\overline{D}$	5	0	1	0	1	V
$\overline{A}BC\overline{D}$	6	0	1	1	0	V
$\overline{A}BCD$	12	1	1	0	0	V
$\overline{A}BCD$	7	0	1	1	1	V
$\overline{A}BCD$	11	1	0	1	1	V
$\overline{A}BCD$	13	1	1	0	1	V
$\overline{A}BCD$	15	1	1	1	1	V

Następnym krokiem jest znalezienie wszystkich par iloczynów, które różnią się tylko jedną zmienną; to znaczy wszystkich par iloczynów, które są takie same z wyjątkiem tego, że jedna ze zmiennych jest 0 w jednym z iloczynów, a 1 w pozostałych. Ze względu na sposób, w jaki pogrupowaliśmy iloczyny, możemy to uczynić, rozpoczynając od pierwszej grupy i porównując każdy iloczyn z pierwszej grupy z każdym iloczynem z drugiej grupy. Następnie porównujemy każdy iloczyn z drugiej grupy ze wszystkimi iloczynami trzeciej grupy i tak dalej. Gdy tylko stwierdza się zgodność, umieszcza się znak (V) za każdym członem, łączy się parę (eliminując zmienną, która różni się w obydwu członach), a wynik dodaje się do nowej listy. Na przykład człony $\overline{A}BCD$ i $\overline{A}BC\overline{D}$ są łączone, co daje $\overline{A}BC$. Kontynuuje się ten proces, aż do przebadania całej oryginalnej tablicy. Wynikiem jest nowa tablica o następujących zapisach:

$\overline{A}CD$	
$\overline{A}BC$	
BCD	V
$\overline{A}BC$	
ABD	V
ABD	V
ACD	
BCD	V

Nowa tablica jest podzielona na grupy w ten sam sposób, jak pierwsza. Następnie jest ona przetwarzana w ten sam sposób jak pierwsza. Oznacza to, że zaznaczane są iloczyny różniące się tylko jedną zmienną i w trzeciej tablicy tworzony jest nowy iloczyn. W naszym przykładzie trzecia tablica zawiera tylko jeden iloczyn:

BD

Opisany proces jest kontynuowany poprzez kolejne tablice, aż zostanie utworzona tablica nie wykazująca żadnych zgodności. W naszym przypadku proces ten objął trzy tablice.

W wyniku tego procesu wyeliminowaliśmy wiele z możliwych członów wyrażenia. Te człony, które nie zostały wyeliminowane, są używane do zbudowania tablicy, takiej jak pokazana w tabeli A.6. Każdy wiersz tej tablicy odpowiada jednemu członowi, który nie został wyeliminowany (nie został zaznaczony) w którejkolwiek z dotychczasowych tablic. Każda kolumna odpowiada jednemu członowi w wyrażeniu oryginalnym. Na przecięciu wiersza i kolumny stawia się znak x wskazujący „zgodność”. Oznacza on, że zmienne występujące w elemencie wiersza mają te same wartości, co zmienne występujące w elemencie kolumny. Następnie zakreślamy kółkiem każdy znak x, który występuje pojedynczo w kolumnie. Każdy x w wierszu, w którym występuje x zakreślony kółkiem, zaznaczamy za pomocą kwadratu. Gdy każda kolumna ma teraz x zaznaczone kwadratem lub kółkiem, to znaczy, że zadanie zostało zakończone. Elementy odpowiadające wierszom, których znaki x zostały zaznaczone, składają się na wyrażenie minimalne. W naszym przykładzie ostatecznym wyrażeniem jest więc

$$ABC + ACD + ABC + \overline{A}\overline{C}\overline{D}$$

W przypadkach, gdy niektóre kolumny nie zawierają kółka ani kwadratu, wymagane jest dodatkowe przetwarzanie. W istocie kontynuujemy dodawanie wierszy, aż we wszystkich kolumnach znajdzie się znak x.

Podsumujmy teraz metodę Quine’a-McKluskeya, próbując intuicyjnie uzasadnić jej funkcjonowanie. Pierwsza faza operacji wydaje się prosta. Proces po-

TABELA A.6. Ostatni etap metody Quine’a-McKluskeya dla wyrażenia $F = ABCD + \overline{A}BCD + \overline{A}BC\overline{D} + \overline{A}B\overline{C}\overline{D} + \overline{A}BCD + \overline{A}BC\overline{D} + \overline{A}BC\overline{D}$

	ABCD	$\overline{A}BCD$	$\overline{A}B\overline{C}\overline{D}$	$\overline{A}B\overline{C}D$	$\overline{A}BC\overline{D}$	$\overline{A}BCD$	$\overline{A}\overline{C}\overline{D}$	$\overline{A}\overline{C}D$
BD	x	x			x		x	
$\overline{A}CD$							S	O
$\overline{A}BC$					H	0		
$\overline{A}BC$		⊗	Ⓜ					
ACD	⊗			0				

lega na eliminowaniu niepotrzebnych zmiennych w poszczególnych członach. Wobec tego wyrażenie $ABC + AB\bar{C}$ jest równoważne AB , ponieważ

$$ABC + AB\bar{C} = AB(C + \bar{C}) = AB1 = AB$$

Po wyeliminowaniu zmiennych pozostało nam wyrażenie, które w oczywisty sposób jest równoważne wyrażeniu oryginalnemu. Mogą w nim jednak występować człony nadmiarowe, podobnie jak nadmiarowe grupy w mapach Karnauha. Ostatnia tablica pozwala na uwzględnienie każdego iloczynu wyrażenia oryginalnego w ten sposób, że liczba iloczynów wyrażenia finalnego jest minimalna.

Realizacja NAND i NOR

Innym problemem realizacji funkcji Boole'a jest rodzaj użytych bramek. Często pożądana jest realizacja funkcji Boole'a wyłącznie za pomocą bramek NAND lub wyłącznie NOR. Chociaż może to nie być realizacja o minimalnej liczbie bramek, ma ona zaletę regularności, co może uprościć proces wytwarzania. Rozważmy ponownie równanie (A-3):

$$F = B(\bar{A} + \bar{C})$$

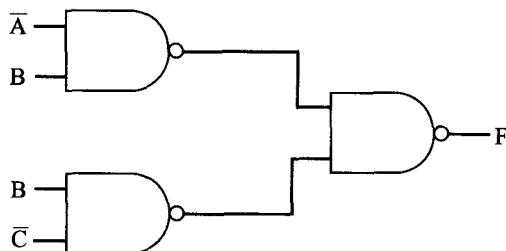
Ponieważ dopełnienie dopełnienia jest po prostu wartością początkową, można napisać:

$$F = B(A + C) = \overline{(\overline{AB}) + (\overline{BC})}$$

Po zastosowaniu prawa De Morgana otrzymujemy:

$$F = \overline{(\overline{AB})} \cdot \overline{(\overline{BC})}$$

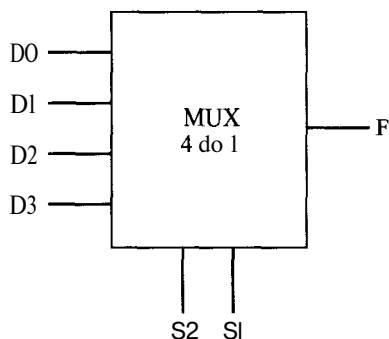
Do realizacji tego wyrażenia są więc potrzebne trzy bramki NAND, co widać na rys. A.11.



RYSUNEK A.11. Realizacja tabeli A.3 za pomocą bramek NAND

Multiplexery

Multiplexer łączy wiele wejść z jednym wyjściem. W dowolnej chwili jedno z tych wejść jest wybrane jako połączone z wyjściem. Ogólny schemat blokowy jest pokazany na rys. A.12. Przedstawia on multiplexer 4 do 1 o czterech liniach wejściowych, oznaczonych D0, D1, D2 i D3. Na wyjście F multiplexera jest dostarczany sygnał z jednej wybranej linii wejściowej. Aby wybrać jedno z 4 możliwych wejść, potrzebny jest 2-bitowy kod wyboru realizowany za pomocą 2 linii wyboru oznaczonych S1 i S2.

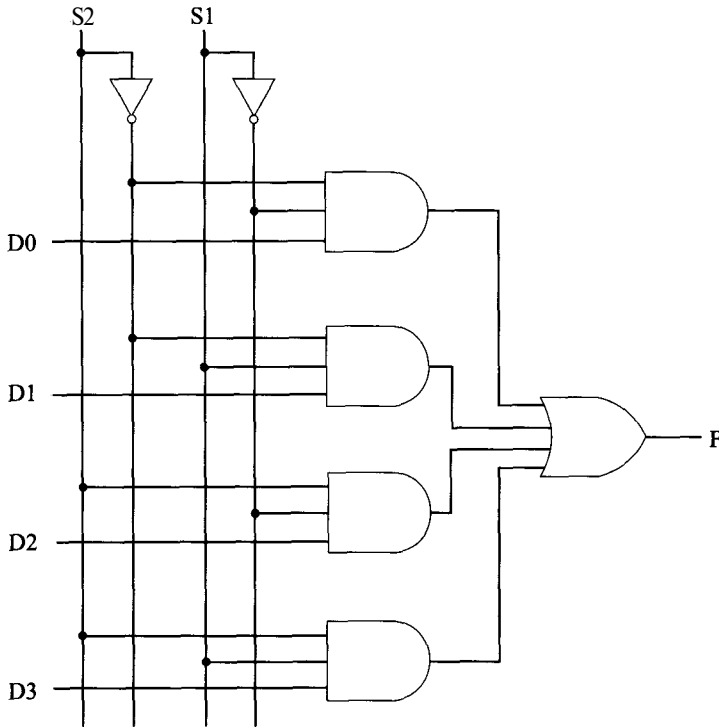


RYСУNEK A.12. Reprezentacja multiplexera 4 do 1

Przykład multiplexera 4 do 1 został zdefiniowany za pomocą tablicy prawdy w tabeli A.7. Jest to uproszczona postać tablicy prawdy. Zamiast pokazywać wszystkie możliwe kombinacje zmiennych wejściowych, zawiera ona sygnały wyjściowe jako dane z linii wejściowej D0, D1, D2 lub D3. Na rysunku A.13 widać realizację przy użyciu bramek AND, OR i NOT. Linie S1 i S2 są połączone z bramkami AND w taki sposób, że dla dowolnej kombinacji S1 i S2 trzy z bramek AND dadzą sygnał wyjściowy 0. Na wyjściu czwartej bramki AND pojawi się sygnał wybranej linii, który może być albo 0, albo 1. Wobec tego na trzech wejściach bramki OR są zawsze 0, a stan wyjścia bramki OR będzie zawsze równy wartości wybranej bramki wejściowej. Przy zastosowaniu tej regularnej organizacji łatwo jest zbudować multiplexery o rozmiarach 8 do 1, 16 do 1 i tak dalej.

TABELA A.7. Tablica prawdy multiplexera 4 do 1

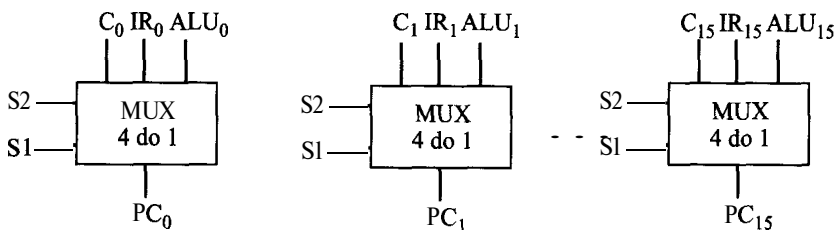
S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3



RYSUNEK A. 13. Realizacja multipleksera

Multipleksery są używane w układach cyfrowych do sterowania przepływem sygnałów i danych. Przykładem jest ładowanie licznika programu (PC). Wartość, która ma być załadowana do licznika programu, może pochodzić z jednego z wielu różnych źródeł:

- a licznika binarnego, jeśli stan PC ma być inkrementowany w celu określenia kolejnego rozkazu;
- rejestru rozkazu, jeśli został właśnie wykonany rozkaz rozgałęzienia używający adresu bezpośredniego;
- a wyjścia ALU, jeśli rozkaz rozgałęzienia określa adres, używając trybu indeksowego.



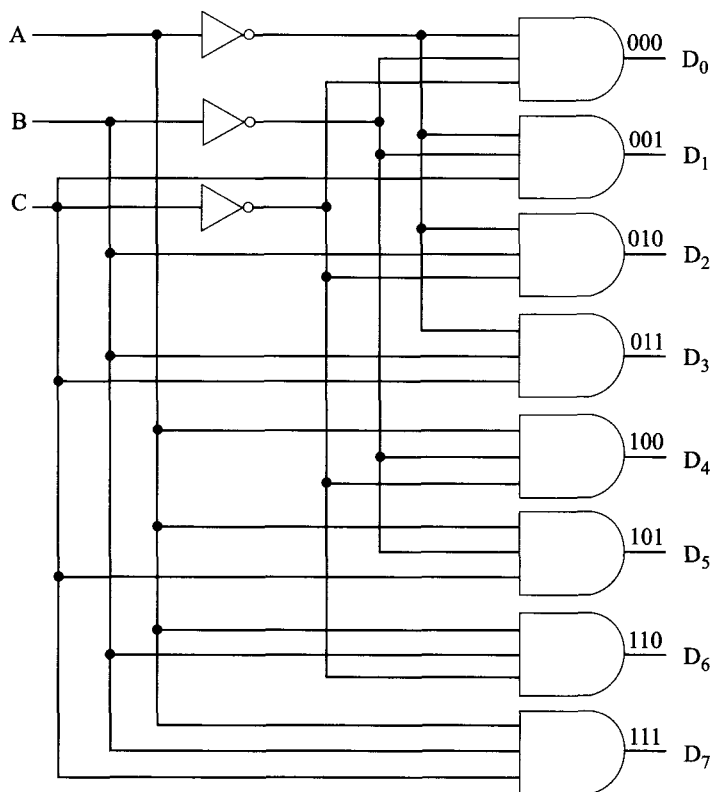
RYSUNEK A. 14. Wejście multiplekserowe do licznika rozkazów

Źródła te mogą być połączone z liniami wejściowymi multiplexera, przy czym PC jest połączony z linią wyjściową. Linie wyboru określają, która wartość jest ładowana do PC. Ponieważ PC ma wiele bitów, używa się wielu multiplexerów, po jednym na bit. Na rysunku A. 14 jest pokazane rozwiązanie dotyczące adresu 16-bitowego.

Dekodery

Dekoder jest układem kombinacyjnym o pewnej liczbie linii wyjściowych, z których w określonej chwili potwierdzana jest tylko jedna, zależnie od kombinacji sygnałów na liniach wejściowych. Ogólnie rzecz biorąc, dekodek ma n wejść i 2^n wyjść. Na rysunku A.15 widać dekodek o 3 wejściach i 8 wyjściach.

Dekodery znajdują wiele zastosowań w komputerach. Jednym z przykładów jest dekodowanie adresu. Załóżmy, że chcemy zbudować pamięć 1-kilobajtową przy użyciu 4 mikroukładów RAM o pojemności 256x8 bitów każdy. Chcemy mieć jedną, zunifikowaną przestrzeń adresową, która może być podzielona następująco:

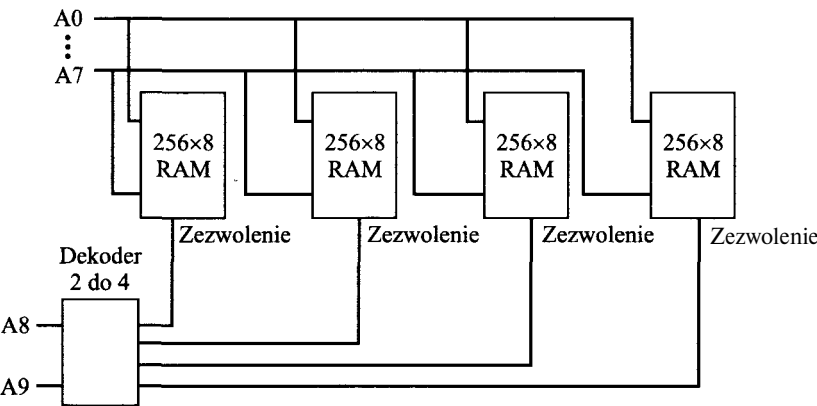


RYSUNEK A. 15. Dekoder z trzema wejściami i $2^3 = 8$ wyjściami

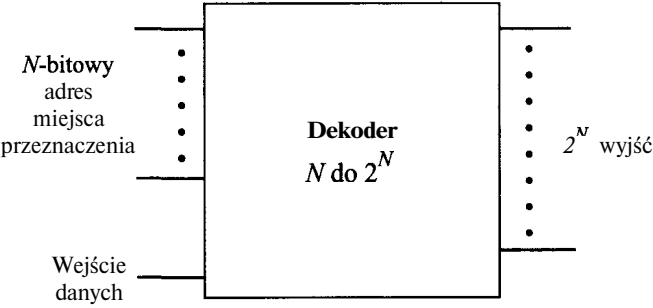
Adres	Mikroukład
0000-00FF	0
0100-01FF	1
0200-02FF	2
0300-03FF	3

Każdy mikroukład wymaga 8 linii adresowych, które mogą być wyznaczone przez 8 najmłodszych bitów adresu. Dwa najstarsze bity w 10-bitowym adresie są używane do wybierania jednego z 4 mikroukładów RAM. Do tego celu zastosowano dekodery 2 do 4, którego sygnały wyjściowe oznaczają zezwolenie dla 1 z 4 mikroukładów, co widać na rys. A. 16.

Po dodaniu jednej linii wejściowej dekodery mogą służyć jako demultipleksery. Demultipleksery wykonują funkcję odwrotną do multipleksersa; łączy on jedno wejście z jednym z kilku wyjść. Widać to na rys. A.17. Jak przedtem, n sygnałów wejściowych po zdekodowaniu daje jeden z 2^n sygnałów wyjściowych. Wszystkie 2^n linii wyjściowych połączono z linią wejściową danych w bramce AND. Wobec tego n wejść działają jako adres służący do wybrania określonej linii wyjściowej, a wartość wejściowej linii danych (0 lub 1) jest kierowana do tej właśnie linii wyjściowej.



RYSUNEK A. 16. Dekodowanie adresu



RYSUNEK A. 17. Realizacja demultipleksersa za pomocą dekodera

Konfiguracja pokazana na rys. A.17 może być postrzegana w inny sposób. Zmieńmy nazwę nowej linii z *wejścia danych* na *zezwoleń*. Umożliwia to takowanie dekodera. Zdekodowane sygnały wyjściowe pojawiają się tylko wtedy, gdy doprowadzone są zakodowane sygnały wejściowe i jednocześnie na linii zezwolenia jest wartość 1.

Programowalne tablice logiczne (PLA)

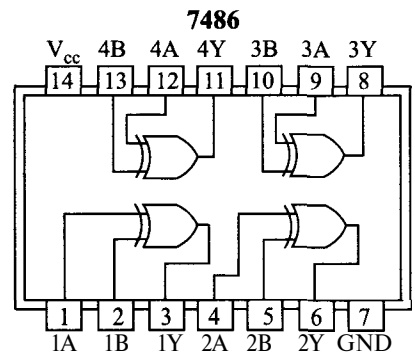
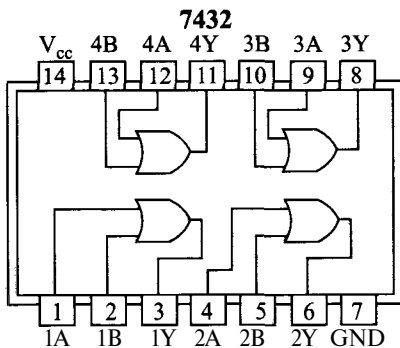
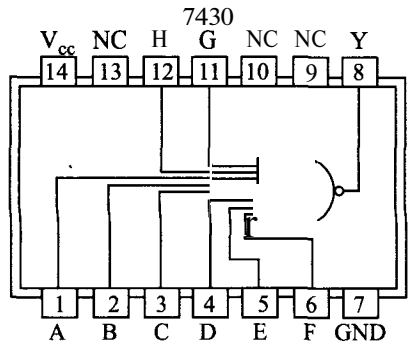
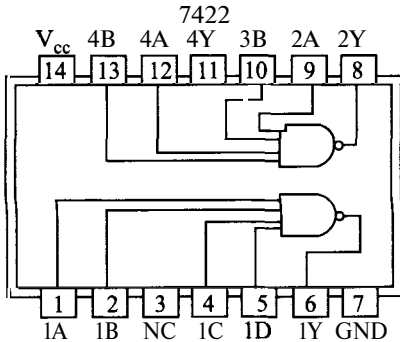
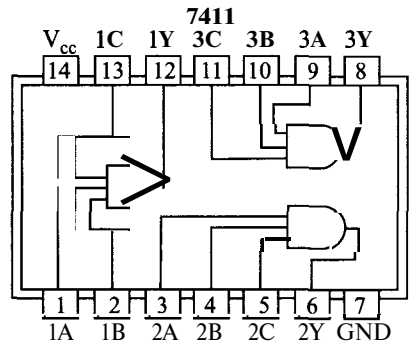
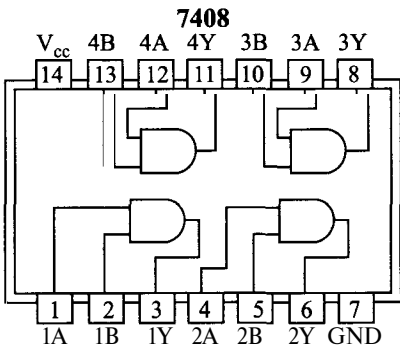
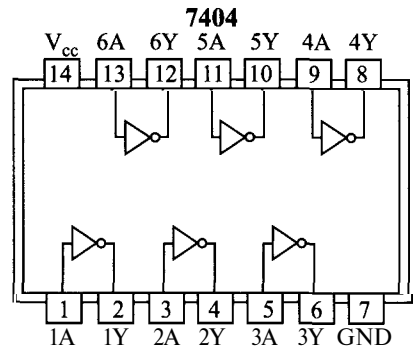
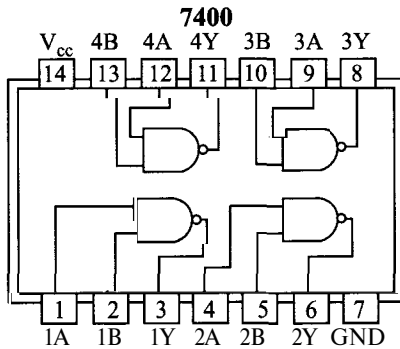
Dotychczas traktowaliśmy pojedyncze bramki jako elementy, za pomocą których mogą być realizowane dowolne funkcje. Projektant dążył do zminimalizowania liczby bramek przez przekształcanie odpowiednich wyrażeń **Boole’a**.

W miarę wzrostu stopnia scalenia układów, pojawia się możliwość innego podejścia. Wczesne układy scalone o małej skali integracji (SSI) zawierały 1÷10 bramek w mikroukładzie. Każda bramka jest traktowana niezależnie, w sposób dotychczas opisany. Na rysunku A.18 są pokazane przykłady mikroukładów SSI. Aby zrealizować funkcję logiczną, pewną liczbę tych mikroukładów umieszcza się na płycie drukowanej, stanowiącej odpowiedni układ połączeń.

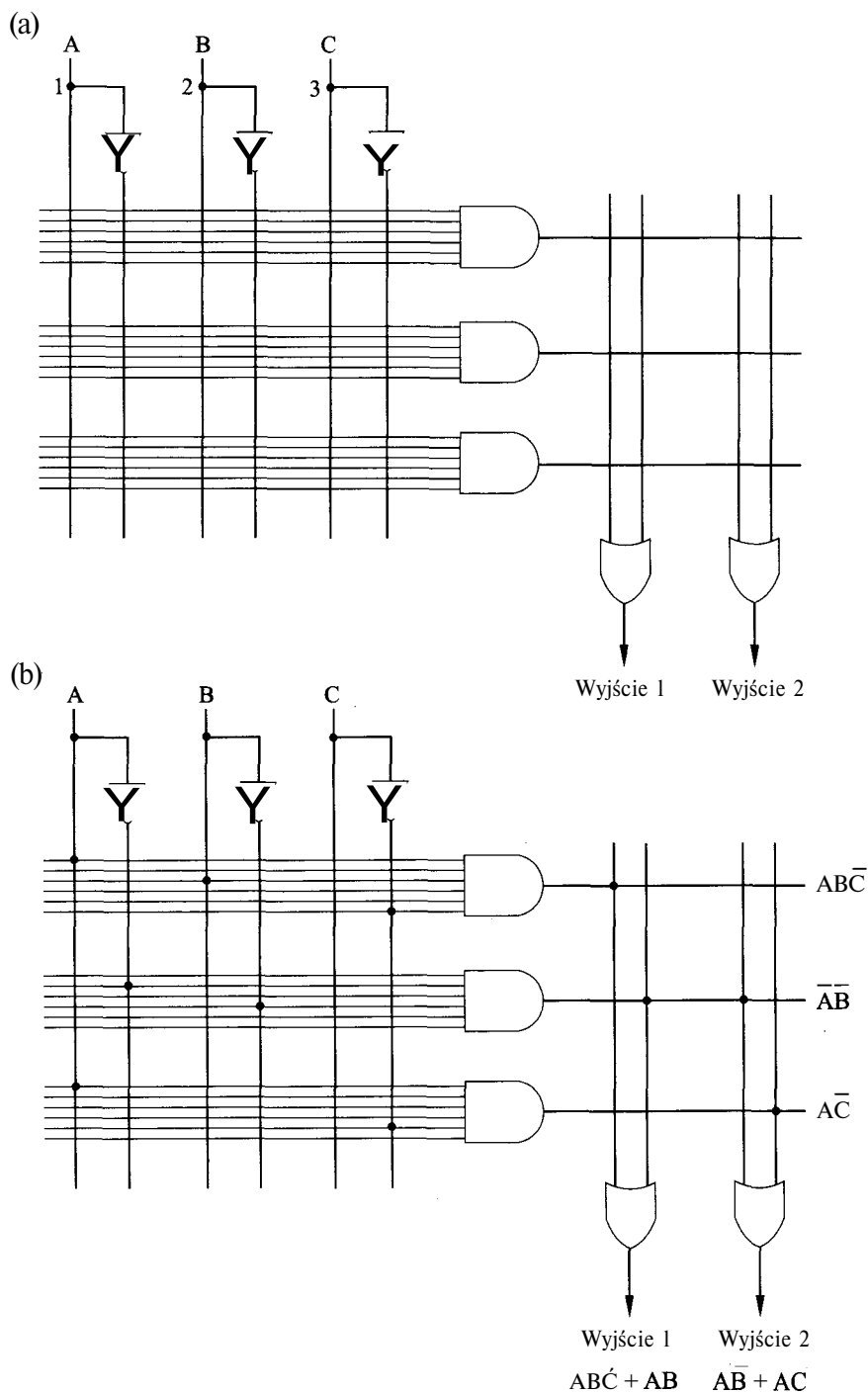
Wzrost stopnia scalenia umożliwił umieszczenie większej liczby bramek w mikroukładzie i zrealizowanie połączeń między nimi wewnątrz mikroukładu. Uzyskano korzyści w postaci zmniejszenia kosztu, zmniejszenia rozmiaru i zwiększenia szybkości (ponieważ opóźnienia wewnątrz mikroukładu są mniejsze niż zewnętrzne). Powstaje jednak pewien problem projektowy. Dla każdej określonej funkcji logicznej lub zbioru funkcji musi być zaprojektowany układ bramek i ich wzajemnych połączeń w mikroukładzie. Koszt i czas wymagane do takiego projektowania na życzenie są znaczne. Wobec tego staje się atrakcyjne opracowanie mikroukładu o ogólnym przeznaczeniu, który można łatwo adaptować do określonych celów. Z tego powodu opracowano *programowalne tablice logiczne* (PLA).

Koncepcja PLA jest oparta na tym, że dowolna funkcja **Boole’a** (tablica prawdy) może być wyrażona w postaci sumy iloczynów. Tablica PLA jest regularnym układem bramek NOT, AND i OR w mikroukładzie. Sygnał z każdego wejścia mikroukładu jest przesyłany przez bramkę NOT, dzięki czemu każdy sygnał wejściowy oraz jego dopełnienie mogą być doprowadzone do każdej bramki AND. Sygnał z wyjścia każdej bramki AND jest osiągalny na wejściu każdej bramki OR, a wyjście każdej bramki OR jest wyjściem mikroukładu. Przez wykonanie odpowiednich połączeń może być zrealizowane dowolne wyrażenie w postaci sumy iloczynów.

Na rysunku A.19a widać układ PLA o 3 wejściach, 8 bramkach i 2 wyjściach. Największe tablice PLA zawierają kilkaset bramek, 15÷25 wejść i 5÷15 wyjść. Połączenia między wejściami a bramkami AND oraz między bramkami AND i OR nie zostały pokazane.



RYSUNEK A. 18. Wybrane mikroukłady SSI. Rozkład wyprowadzeń na podstawie *The TTL data book for design engineers*, ©1976 Texas Instruments Incorporated



RYSUNEK A.19. Przykład programowalnej tablicy logicznej: (a) PLA o 3 wejściach i 2 wyjściach; (b) schemat połączeń PLA o 3 wejściach i 2 wyjściach

Układy PLA są wytwarzane na dwa sposoby w celu umożliwienia łatwego programowania (wykonywania połączeń). Pierwszy sposób polega na tym, że w każdym punkcie przecięcia jest wykonywane połączenie za pomocą pewnego rodzaju bezpiecznika. Niepożądane połączenia mogą być następnie usuwane przez przepalenie bezpiecznika. Ten rodzaj PLA jest określany jako **programowalny przez użytkownika**. Alternatywnie, odpowiednie połączenia mogą być wykonane podczas wytwarzania mikroukładu za pomocą odpowiedniej maski określającej pożądany schemat połączeń. W każdym przypadku PLA jest elastycznym, tanim rozwiązaniem realizacji cyfrowych funkcji logicznych.

Na rysunku A. 19b jest pokazany projekt realizacji dwóch wyrażeń Boole'a.

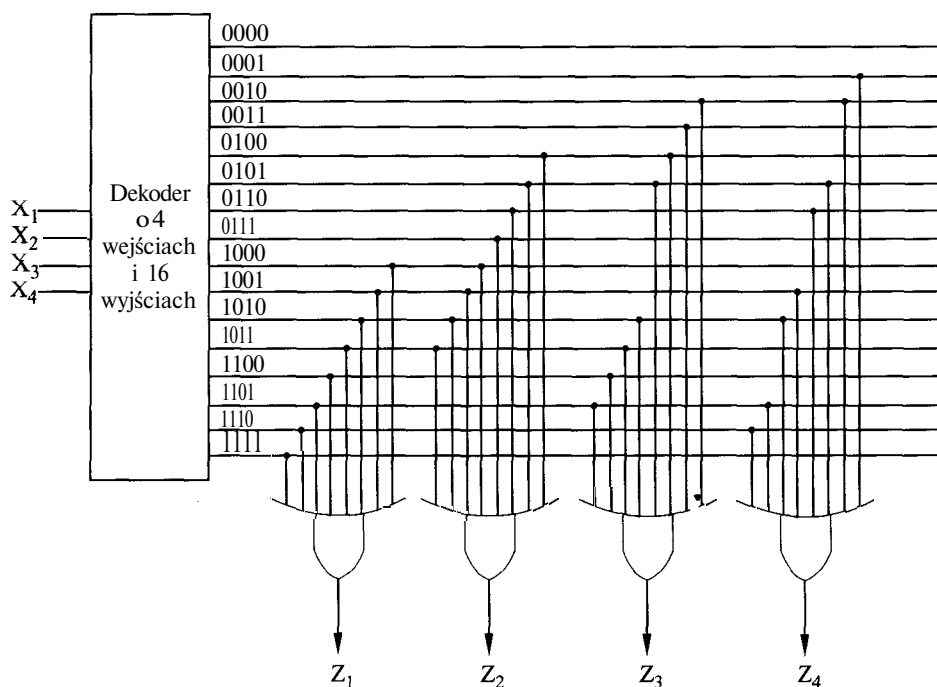
Pamięć stała (ROM)

Układy kombinacyjne są często określane jako układy „bez pamięci”, ponieważ ich wyjście zależy wyłącznie od bieżącego stanu ich wejścia i żadne informacje historyczne o poprzednich stanach wejść nie są zachowywane. Istnieje jednak pewien rodzaj pamięci, który jest realizowany za pomocą układów kombinacyjnych, a mianowicie **pamięć stała (ROM)**.

Przypomnijmy sobie, że ROM jest jednostką pamięci, która umożliwia wykonanie wyłącznie operacji odczytu. Informacja binarna zawarta w ROM jest w związku z tym trwała. Jest ona zapisywana w pamięci podczas procesu wytwarzania. Wobec tego określona kombinacja sygnałów wejściowych ROM (na liniach adresowych) prowadzi do zawsze tej samej kombinacji sygnałów wyjściowych (na liniach danych). Ponieważ stany na wyjściach są funkcją wyłącznie aktualnych stanów wejść, ROM jest w rzeczywistości układem kombinacyjnym.

TABELA A.8. Tablica prawdy pamięci stałej (ROM)

Wejście				Wyjście			
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0



RYSUNEK A.20. 64-bitowa pamięć ROM

Pamięć ROM może być wdrożona za pomocą dekodera i zbioru bramek OR. Jako przykład rozważmy tabelę A.8. Może ona być postrzegana jako tablica prawdy o 4 wejściach i 4 wyjściach. Dla każdej z 16 możliwych wartości wejściowych jest podany odpowiedni zbiór wartości wyjściowych. Tabele tę można również widzieć jako definiującą zawartość 64-bitowej pamięci ROM składającej się z 16 słów 4-bitowych. Cztery wejścia określają adres, a 4 wyjścia - zawartość lokacji w pamięci określonej przez ten adres. Na rysunku A.20 jest pokazany sposób realizacji tej pamięci za pomocą dekodera 4 do 16 i 4 bramek OR. Podobnie jak w przypadku PLA, stosowana jest regularna organizacja, a schemat połączeń określa pożądaną zawartość.

Sumatory

Zobaczyliśmy dotychczas, w jaki sposób połączone bramki mogą być użyte do realizacji takich funkcji, jak kierowanie przepływem sygnałów, dekodowanie i pamięć ROM. Obszarem o zasadniczym znaczeniu, którego jeszcze nie rozpatrywaliśmy, są operacje arytmetyczne. W ramach tego krótkiego przeglądu zadowolimy się analizą funkcji dodawania.

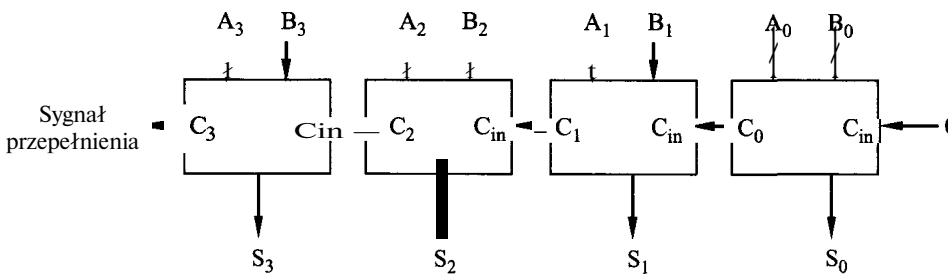
Suma binarna różni się tym od sumy w algebrze Boole'a, że wynik obejmuje składnik przeniesienia. Wobec tego

$$\begin{array}{r} 0 \\ + 0 \\ \hline 0 \end{array} \quad \begin{array}{r} 0 \\ + 1 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ + 0 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ + 1 \\ \hline 10 \end{array}$$

Mimo to dodawanie może być realizowane za pomocą wyrażeń Boole’a. W tabeli A.9a jest pokazana tablica prawdy dla sumowania dwóch bitów wejściowych w celu uzyskania 1-bitowej sumy i bitu przeniesienia. Tablica ta z łatwością może być zrealizowana w postaci cyfrowego układu logicznego. Nie jesteśmy jednak zainteresowani dodawaniem tylko pary bitów. Chcielibyśmy dodawać dwie liczby *n-bitowe*. Można to realizować przez zestawienie szeregu sumatorów w ten sposób, że przeniesienie z jednego sumatora jest doprowadzone na wejście następnego. Sumator 4-bitowy widać na rys. A.21.

TABELA A.9. Tablica prawdy dodawania binarnego

(a) Dodawanie pojedynczych bitów				(b) Dodawanie wraz z przeniesieniem				
A	B	Suma	Przeniesienie	C _{in}	A	B	Suma	C _{out}
0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	1	1	0
1	0	1	0	0	1	0	1	0
1	1	0	1	0	1	1	0	1
				1	0	0	1	0
				1	0	1	0	1
				1	1	0	0	1
				1	1	1	1	1

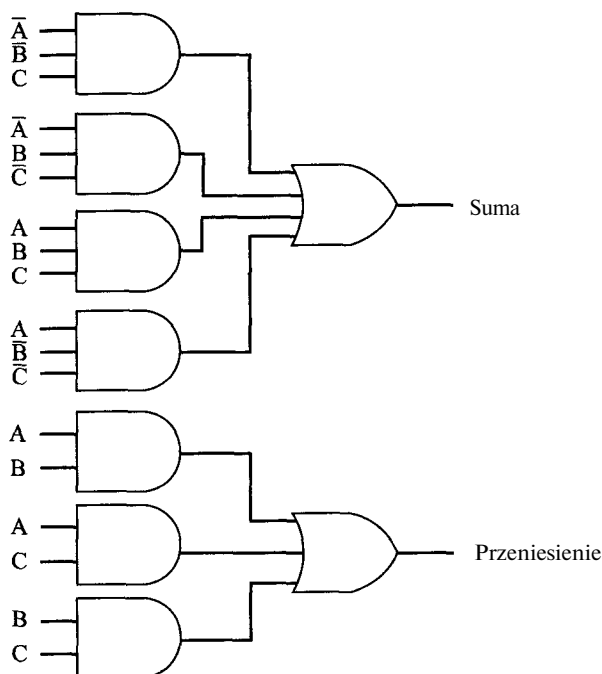


RYSUNEK A.21. Sumator 4-bitowy

Aby sumator wielobitowy mógł działać poprawnie, każdy z sumatorów 1-bitowych musi mieć 3 wejścia, włącznie z przeniesieniem z następnego sumatora niższego rzędu. Zrewidowana tablica prawdy znajduje się w tabeli A.9b. Stany dwóch wyjść mogą być wyrażone następująco:

$$\begin{aligned} \text{Suma} &= \overline{A}\overline{B}C + \overline{A}B\overline{C} + AB\overline{C} + ABC \\ \text{Przeniesienie} &= AB + AC + BC \end{aligned}$$

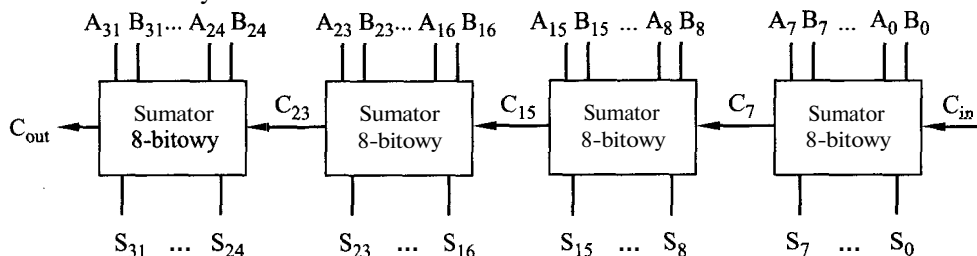
Na rysunku A.22 jest pokazana realizacja za pomocą bramek AND, OR i NOT.



RYSUNEK A.22. Realizacja sumatora

Dysponujemy więc układami logicznymi niezbędnymi do tego, by wdrożyć wielobitowy sumator, taki jak na rys. A.23. Zauważmy, że ponieważ wartość wyjściowa każdego sumatora zależy od przeniesienia z poprzedniego sumatora, istnieje opóźnienie narastające od najmniej znaczącego do najbardziej znaczącego bitu. W każdym sumatorze 1-bitowym następują pewne opóźnienia bramkowe, które się kumulują. W dużych sumatorach zakumulowane opóźnienie może wzrosnąć do wartości nie do zaakceptowania.

Gdyby wartości przeniesienia mogły być określone bez przechodzenia przez wszystkie poprzednie stopnie, każdy sumator 1-bitowy mógłby działać niezależnie i opóźnienia by się nie kumulowały. Można to osiągnąć, stosując rozwiązanie nazywane **układem przeniesienia na bardziej znaczące pozycje** (ang. *carry lookahead*). W celu wyjaśnienia tego rozwiązania ponownie rozważmy sumator 4-bitowy.



RYSUNEK A.23. Budowanie sumatora 32-bitowego za pomocą sumatorów 8-bitowych

Chcielibyśmy otrzymać wyrażenie, które określa wartość na wejściu przeniesienia dowolnego stopnia sumatora bez odnoszenia się do poprzednich wartości przeniesienia. Mamy więc

$$C_0 = A_0 B_0 \quad (A-4)$$

$$C_1 = A_1 B_1 (A_1 + B_1) C_0 \quad (A-5)$$

Podstawiając równanie (A-4) do (A-5), otrzymujemy

$$C_1 = A_1 B_1 + A_1 A_0 B_0 + B_1 A_0 B_0$$

Powtarzając tę samą procedurę, otrzymujemy

$$C_2 = A_2 B_2 + A_2 A_1 B_1 + A_2 A_1 A_0 B_0 + A_2 B_1 A_0 B_0 + B_2 A_1 B_1 + B_2 A_1 A_0 B_0 + B_2 B_1 A_0 B_0$$

Proces ten może być powtarzany dla dowolnie długich sumatorów. Każdy składnik przeniesienia może być wyrażony w postaci sumy iloczynów jako funkcja wyłącznie oryginalnych danych wejściowych, bez zależności od przeniesień. Wobec tego, niezależnie od długości sumatora, występują tylko dwa poziomy opóźnienia bramkowego.

W przypadku długich liczb rozwiązanie to staje się nadmiernie skomplikowane. Obliczanie wyrażenia dla najbardziej znaczącego bitu sumatora n -bitowego wymaga bramki OR o $n - 1$ wejściach oraz n bramek AND z liczbą wejść $2 \div (n + 1)$. Dlatego też układ przeniesienia na bardziej znaczące pozycje jest zwykle realizowany jednocześnie dla 4÷8 bitów. Na rysunku A.23 jest pokazana budowa sumatora 32-bitowego złożonego z 4 sumatorów 8-bitowych. W tym przypadku przeniesienie musi przechodzić przez 4 sumatory 8-bitowe, jednak proces ten będzie znacznie szybszy niż przechodzenie przez 32 sumatory 1-bitowe.

M

UKŁADY SEKWENCYJNE

Układy kombinacyjne służą do wdrażania podstawowych funkcji komputera cyfrowego. Jednak, poza szczególnym przypadkiem pamięci ROM, nie umożliwiają one zrealizowania pamięci lub przechowywania informacji o stanie, które również mają zasadnicze znaczenie dla działania komputera. Do tego ostatniego celu służy bardziej złożona postać cyfrowych układów logicznych: układy sekwencyjne. Bieżący stan wyjścia układu sekwencyjnego zależy nie tylko od bieżącego stanu wejścia, ale również od historii stanu wejścia. Inny i na ogół bardziej użyteczny punkt widzenia jest taki, że bieżący stan wyjścia układu sekwencyjnego zależy od bieżącego wejścia oraz od bieżącego stanu samego układu.

W tym podrozdziale przeanalizujemy pewne proste, lecz użyteczne przykłady układów sekwencyjnych. Jak zobaczymy, w układzie sekwencyjnym wykorzystuje się układy kombinacyjne.

Przerzutniki

Najprostszą formą układu sekwencyjnego jest **przerzutnik**. Istnieją różne przerzutniki, jednak wszystkie mają następujące dwie własności:

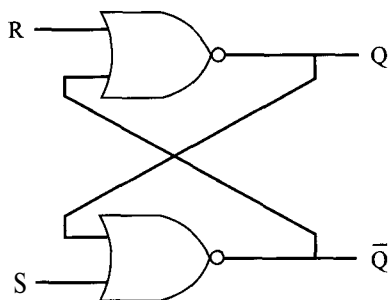
- a Przerzutnik jest układem **dwustabilnym**. Istnieje w jednym z dwóch stanów i bez sygnałów wejściowych zachowuje swój stan. Wobec tego przerzutnik może działać jako pamięć 1-bitowa.
- a Przerzutnik ma dwa wyjścia, które zawsze dopełniają się wzajemnie. Są one zwykle oznaczane jako Q i $\bar{Q}$.

Przerzutnik zatrzaskowy S-R

Na rysunku A.24 widać powszechnie znaną konfigurację **przerzutnika S-R** zwanego też zatrzaskiem (przerzutnikiem zatrzaskowym) S-R. Układ ma dwa wejścia, S (*Set*, ustawienie) i R (*Reset*, przestawienie), oraz dwa wyjścia, Q i $\bar{Q}$. Składa się z dwóch bramek NOR połączonych ze sobą w układzie sprzężenia zwrotnego.

Pokażmy najpierw, że układ jest dwustabilny. Załóżmy, że $S = 0$ i $R = 0$ oraz że $Q = 0$. Na wejściu dolnej bramki NOR są wartości $Q = 0$ i $S = 0$. Wobec tego stan wyjścia $\bar{Q} = 1$ oznacza, że wartości na wejściach górnej bramki NOR są $\bar{Q} = 1$ i $R = 0$, co na wyjściu daje $Q = 0$. Stan układu jest więc wewnętrznie spójny i pozostanie stabilny tak długo, jak $S = R = 0$. Rozumując podobnie, dojdziemy do wniosku, że stan $Q = 1$, $\bar{Q} = 0$ jest również stabilny przy $R = S = 0$.

Układ może więc działać jako pamięć 1-bitowa. Możemy uważać wyjście Q za „wartość” bitu. Wejścia S i R służą do zapisu do pamięci wartości odpowiednio 1 i 0. Żeby to dostrzec, rozważmy stan $Q = 0$, $\bar{Q} = 1$, $S = 0$, $R = 0$. Załóżmy,

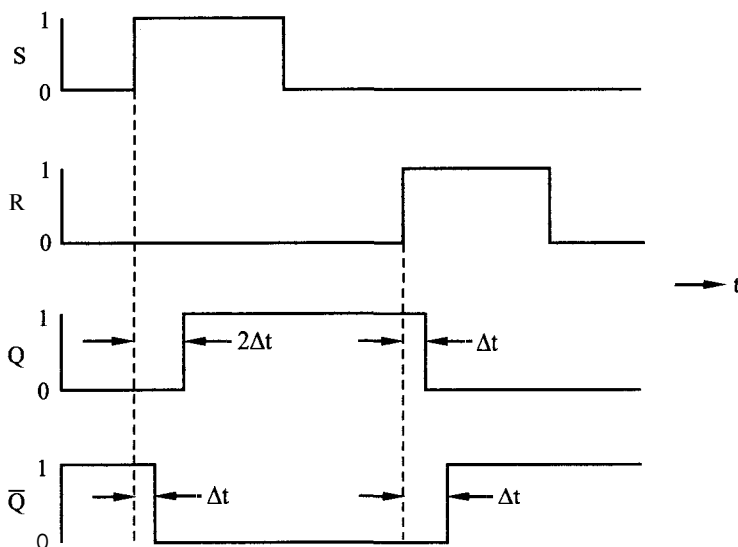


RYSUNEK A.24. Przerzutnik S-R typu zatrzask zrealizowany za pomocą bramek NOR

że stan S ulega zmianie na 1. Teraz na wejściach dolnej bramki NOR są wartości $S = 1$, $Q = 0$. Po pewnym opóźnieniu czasowym Δt , na wyjściu dolnej bramki NOR będzie $\overline{Q} = 0$ (patrz rys. A.25). Wobec tego wówczas na wejściach dolnej bramki NOR będą wartości $R = 0$, $\overline{Q} = 0$. Po następnym opóźnieniu Δt wartość wyjścia Q stanie się równa 1. Jest to znowu stan stabilny. Na wejściach dolnej bramki są teraz wartości $S = 1$, $Q = 1$, a więc wyjście pozostaje w stanie $\overline{Q} = 0$. Tak długo, jak $S = 1$ i $R = 0$, wartości wyjściowe pozostaną równe $Q = 1$, $\overline{Q} = 0$. Ponadto, jeśli S wróci do stanu 0, wartości na wyjściach pozostaną niezmiennione.

Wejście R realizuje przeciwną funkcję. Gdy R przyjmuje stan 1, wymusza to $Q = 0$ i $\overline{Q} = 1$ niezależnie od poprzedniego stanu Q i $\overline{Q}$. Jak poprzednio, powtórna stabilizacja następuje po opóźnieniu $2\Delta t$ (rys. A.25).

Zatrask S-R może być zdefiniowany za pomocą tablicy podobnej do tablicy prawdy, nazywanej **tablicą własności** (*characteristic table*), która pokazuje następny stan lub stany układu sekwencyjnego jako funkcję stanów bieżących i wartości wejściowych. W przypadku zatrasku S-R stan może być zdefiniowany przez wartość Q . W tabeli A.10a pokazano wynikającą stąd tablicę własności. Zauważmy, że stan $S = 1$, $R = 1$ jest niedozwolony, ponieważ prowadziłoby to do niespójnego stanu wyjść (Q i $\overline{Q}$ jednocześnie równe 0). Tablica ta może być wyrażona w sposób bardziej zwarty, co widać w części b tabeli A.10. Zachowanie zatrasku S-R jest pokazane w części c tej tabeli.



RYСУNEK A.25. Przebiegi czasowe zatrasku S-R zrealizowanego za pomocą bramek NOR

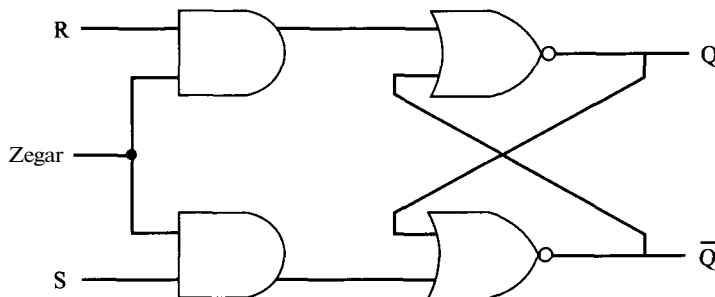
TABELA A. 10. Przerzutnik S-R

(a) Tablica własności			(b) Uproszczona tablica własności		
Bieżące wejścia	Stan bieżący	Stan następny	S	R	Q_{n+1}
SR	Q_n	Q_{n+1}	0	0	Q_n
00	0	0	0	1	0
00	1	1	1	0	1
01	0	0	1	1	–
01	1	0			
10	0	1			
10	1	1			
11	0	–			
11	1	–			

(c) Odpowiedź na szereg sygnałów wejściowych										
t	0	1	2	3	4	5	6	7	8	9
S	1	0	0	0	0	0	0	0	1	0
R	0	0	0	1	0	0	1	0	0	0
Q_{n+1}	1	1	1	0	0	0	0	0	1	1

Synchronizowany przerzutnik S-R

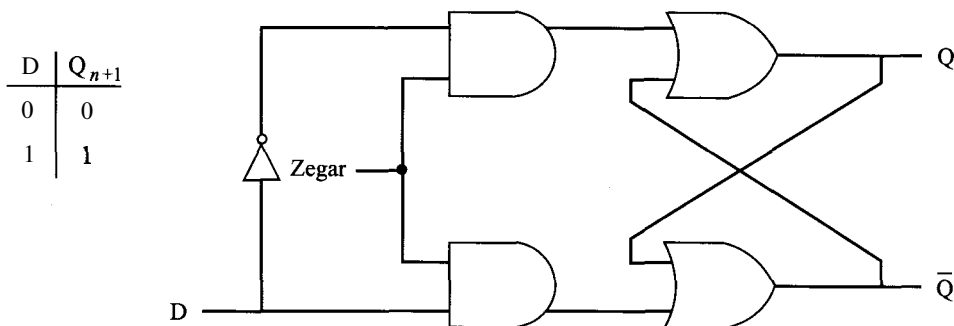
Zmiana wyjścia przerzutnika S-R następuje po krótkim opóźnieniu, jako reakcja na zmianę wejścia. Określamy to jako **działanie asynchroniczne**. Częściej jednak zdarzenia w komputerze są synchronizowane za pomocą impulsów zegarowych, zmiany następują więc tylko wtedy, gdy jest doprowadzony impuls zegarowy. Układ taki widać na rys. A.26. Urządzenie to jest określane jako **synchronizowany przerzutnik S-R**. Zauważmy, że stany z wejść R i S są doprowadzane do bramek NOR, tylko podczas trwania impulsu zegarowego.



RYSUNEK A.26. Synchronizowany przerzutnik S-R

Przerzutnik D

W przypadku przerzutnika S-R problemem jest to, że należy zapobiegać powstaniu sytuacji, w której $R=1$, $S=1$. Jednym ze sposobów osiągnięcia tego jest pozostawienie tylko jednego wejścia. Tak właśnie jest rozwiązany przerzutnik D. Na rysunku A.27 jest pokazana realizacja za pomocą bramek oraz tablica



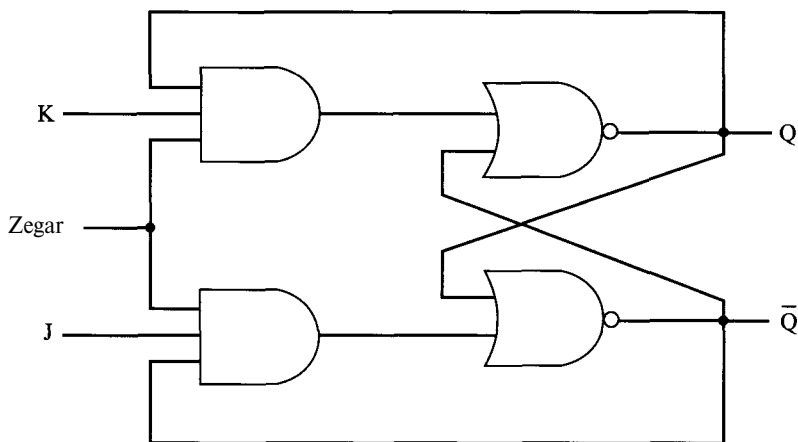
RYSUNEK A.27. Przerzutnik D

własności przerzutnika D. Za pomocą inwertera zagwarantowano, że niezegrowe wejścia do dwóch bramek AND są swoimi przeciwieństwami.

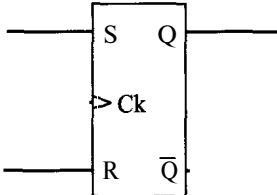
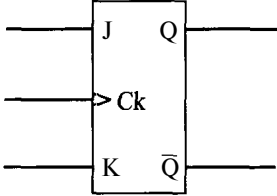
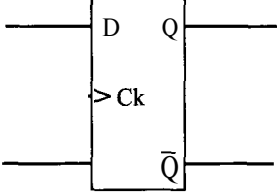
Przerzutnik D jest czasami określany jako **przerzutnik danych**, ponieważ jest on w rzeczywistości komórką pamiętającą jeden bit danych. Stan na wyjściu przerzutnika D jest zawsze równy ostatniej wartości doprowadzonej na wejście. Przerzutnik pamięta więc ostatni sygnał wejściowy. Jest on również określany jako przerzutnik opóźniający, ponieważ pojawienie się na wyjściu 0 lub 1 doprowadzonych do wejścia jest opóźnione o jeden impuls zegara.

Przerzutnik J-K

Jeszcze innym użytecznym przerzutnikiem jest przerzutnik J-K. Podobnie jak przerzutnik S-R ma on dwa wejścia. Jednak w tym przypadku wszystkie możliwe kombinacje wartości wejściowych są dopuszczalne. Na rysunku A.28 jest przedstawiona realizacja przerzutnika J-K za pomocą bramek, a na rys. A.29 –



RYSUNEK A.28. Przerzutnik J-K

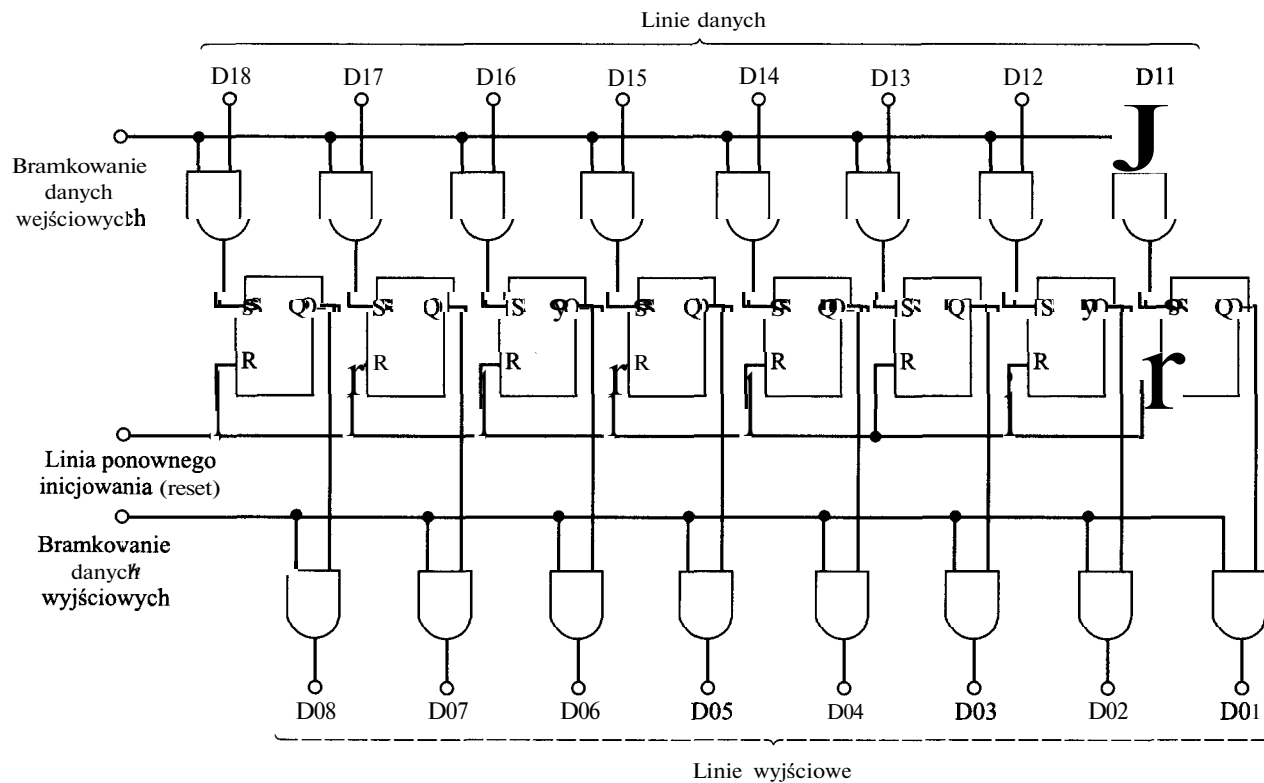
Nazwa	Symbol graficzny	Tablica stanów															
S-R		<table><tr><th>S</th><th>R</th><th>Q_{n+1}</th></tr><tr><td>0</td><td>0</td><td>Q_n</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>-</td></tr></table>	S	R	Q_{n+1}	0	0	Q_n	0	1	0	1	0	1	1	1	-
S	R	Q_{n+1}															
0	0	Q_n															
0	1	0															
1	0	1															
1	1	-															
J-K		<table><tr><th>J</th><th>K</th><th>Q_{n+1}</th></tr><tr><td>0</td><td>0</td><td>Q_n</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>$\frac{1}{Q_n}$</td></tr><tr><td>1</td><td>1</td><td>$\bar{Q}_n$</td></tr></table>	J	K	Q_{n+1}	0	0	Q_n	0	1	0	1	0	$\frac{1}{Q_n}$	1	1	$\bar{Q}_n$
J	K	Q_{n+1}															
0	0	Q_n															
0	1	0															
1	0	$\frac{1}{Q_n}$															
1	1	$\bar{Q}_n$															
D		<table><tr><th>K</th><th>Q_{n+1}</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	K	Q_{n+1}	0	0	1	1									
K	Q_{n+1}																
0	0																
1	1																

RYSUNEK A.29. Podstawowe przerzutniki

tablica własności (obok tablic odnoszących się do przerzutników S-R i D). Za-uważmy, że pierwsze trzy kombinacje są takie same, jak dla przerzutnika S-R. Bez sygnałów na wejściu wyjście jest stabilne. Samo wejście J umożliwia zrealizowanie funkcji ustawiania, powodując ustawienie na wejściu stanu 1; samo wejście K realizuje kasowanie (*reset*), powodując ustawienie na wyjściu stanu 0. Gdy oba wejścia (J i K) są równe 1, wykonywana funkcja jest określana jako kluczkowanie: stan wyjścia ulega odwróceniu. Jeśli więc Q jest równe 1, a do wejść J i K zostaną doprowadzone 1, to Q stanie się równe 0. Czytelnik może zweryfikować, że wdrożenie przedstawione na rys. A.28 rzeczywiście realizuje tę charakterystyczną funkcję.

Rejestry

Jako przykład zastosowania przerzutników, przeanalizujemy najpierw jeden z zasadniczych elementów procesora: rejestr. Jak wiemy, rejestr jest układem cyfrowym używanym wewnątrz procesora do przechowywania jednego lub wielu bitów danych. Powszechnie używane są dwa rodzaje rejestrów: równoległe i przesuwające.



RYСУNEK A.30. 8-bitowy rejestr równoległy

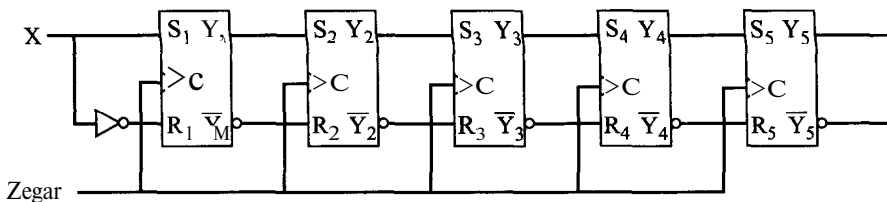
Rejestr równoległy

Rejestr równoległy składa się ze zbioru pamięci 1-bitowych, które mogą być jednocześnie odczytywane lub zapisywane. Służy do przechowywania danych. Rejestry, o których była mowa w tej książce, są rejestrami równoległymi.

Na rysunku A.30 jest pokazany rejestr 8-bitowy w celu zilustrowania działania rejestru równoległego. Zastosowano w nim przerzutniki zatraskowe S-R. Sygnał sterujący, oznaczony jako **bramkowanie danych wejściowych**, służy do sterowania zapisem do rejestru z linii sygnałowych DI 1÷D18. Linie te mogą być wyjściem multiplexera, co umożliwi ładowanie do rejestru danych z różnych źródeł. Wyjście jest sterowane w podobny sposób. Jako rozwiązanie dodatkowe przewidziano linię umożliwiającą łatwe zerowanie rejestru. Zauważmy, że nie mogłoby to tak łatwo być zrealizowane w przypadku rejestru zbudowanego z przerzutników D.

Rejestr przesuwający

Rejestr przesuwający przyjmuje i (lub) przekazuje informację szeregowo. Rozważmy na przykład rys. A.31, na którym widać 5-bitowy rejestr przesuwający zbudowany z synchronizowanych przerzutników S-R. Wprowadzanie danych następuje tylko do przerzutnika z lewej strony. Za każdym impulsem zegarowym dane są przesuwane o jedną pozycję w prawo, a bit położony najbardziej na prawo jest wyprowadzany na wyjście.



RYСУNEK A.31. 5-bitowy rejestr przesuwający

Rejestry przesuwające mogą być używane w interfejsach z szeregowymi urządzeniami wejścia-wyjścia. Ponadto mogą one być stosowane wewnątrz ALU do realizowania funkcji logicznych przesunięcia i obrotu. W tym przypadku mogą one wymagać również układów równoległego odczytu i zapisu.

Liczniki

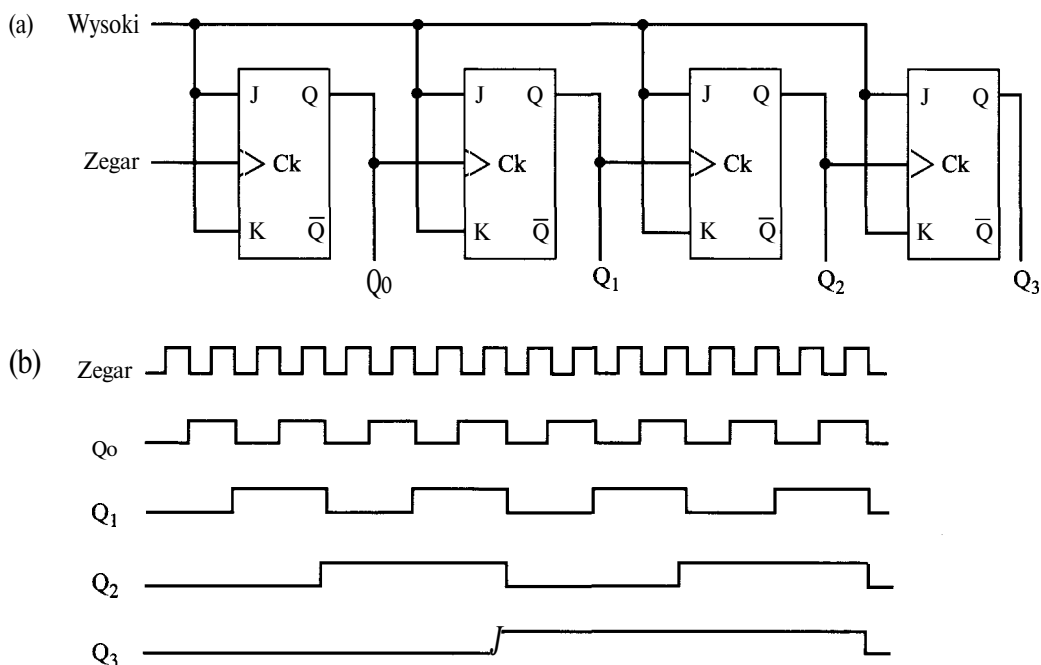
Inną użyteczną kategorią układów sekwencyjnych są liczniki. Licznik jest rejestrem, którego zawartość może być z łatwością inkrementowana o 1 modulo pojemność rejestru. Wobec tego rejestr wykonany z n przerzutników może liczyć do $2^n - 1$. Gdy zawartość licznika jest zwiększana poza jego wartość mak-

symalną, jest on ustawiany na 0. Przykładem licznika występującego w procesorze jest licznik programu.

Liczniki mogą być projektowane jako **asynchroniczne** lub synchroniczne, zależnie od sposobu pracy. Liczniki asynchroniczne są stosunkowo powolne, ponieważ **wyjście** jednego **przerzutnika** wyzwala zmianę stanu następnego **przerzutnika**. W liczniku synchronicznym wszystkie **przerzutniki** zmieniają stan jednocześnie. Ponieważ liczniki tego ostatniego rodzaju są o wiele szybsze, są one stosowane w procesorze. Celowe jest jednak rozpoczęcie dyskusji od opisu licznika **asynchronicznego**.

Licznik szeregowy

Licznik asynchroniczny jest również określany jako licznik szeregowy, ponieważ zmiana, która następuje w celu zwiększenia stanu, rozpoczyna się na jednym końcu, po czym przemieszcza się do drugiego. Na rysunku A.32 są przedstawione: wdrożenie 4-bitowego licznika zbudowanego z **przerzutników J-K** oraz diagram czasowy ilustrujący jego zachowanie. Diagram czasowy taktowania jest wyidealizowany - nie pokazuje on opóźnienia propagacji, które występuje przy przechodzeniu sygnałów przez szereg przerzutników. Wyjście lewego przerzutnika (Q_0) jest najmniej znaczącym bitem. Projekt ten mógłby oczywiście być rozszerzony do dowolnej liczby bitów przez dołączenie większej liczby przerzutników.



RYSunek A.32. Licznik szeregowy: (a) układ sekwencyjny; (b) przebiegi czasowe

W przedstawionej wersji stan licznika jest zwiększany za każdym impulsem zegarowym. Wejścia J i K każdego przerzutnika są utrzymywane na stałym poziomie 1. Oznacza to, że gdy wystąpi impuls zegarowy, stan na wyjściu Q zmieni się na przeciwny (z 1 na 0; z 0 na 1). Zauważmy, że zmiana stanu jest pokazana jako następująca podczas opadającego zbocza impulsu zegarowego; rozwiązanie to jest określane jako **przerzutnik przełączany zboczem sygnału**. Używanie przerzutników reagujących na zmianę impulsu zamiast na sam impuls umożliwia lepsze sterowanie taktowaniem złożonych układów. Wzór sygnałów wyjściowych tego licznika polega na cyklicznym powtarzaniu 0000, 0001, ..., 1110, 1111, 0000 itd.

Liczniki synchroniczne

Wadą licznika szeregowego jest opóźnienie wprowadzane podczas zmiany wartości, które jest **proporcjonalne** do długości licznika. Aby wyeliminować tę wadę, w procesorze stosuje się liczniki synchroniczne, w których wszystkie przerzutniki licznika jednocześnie zmieniają stan. W tym punkcie przedstawimy projekt 3-bitowego licznika synchronicznego. Zilustrujemy w ten sposób pewne podstawowe koncepcje w projektowaniu układów synchronicznych.

Licznik 3-bitowy wymaga użycia trzech przerzutników. Zastosujemy przerzutniki J-K. Oznaczmy niezanegowane wyjścia trzech przerzutników odpowiednio jako A, B i C, przy czym C reprezentuje najmniej znaczący bit. Pierwszym krokiem jest zbudowanie tablicy prawdy wiążącej wejścia J-K z wyjściami, co pozwoli na zaprojektowanie całego układu. Tablica taka jest pokazana na rys. A.33a. W pierwszych trzech kolumnach są podane możliwe kombinacje wyjść A, B i C. Wymieniono je w kolejności, w jakiej ukażą się przy zwiększaniu stanu licznika. Każdy wiersz zawiera bieżącą wartość A, B, C oraz stany wejść trzech przerzutników, jakie będą wymagane do osiągnięcia następnej wartości A, B, C.

Aby zrozumieć w jaki sposób buduje się tablice prawdy z rys. A.33a, pożyteczne może być przypomnienie tablicy własności przerzutnika J-K. Wygląda ona następująco:

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

W tej postaci tablica ukazuje wpływ, jaki wejścia J i K mają na wyjście. Rozważmy teraz następującą organizację tej samej informacji:

Q_n	J	K	Q_{n+1}
0	0	d	0
0	1	d	1
1	d	1	0
1	d	0	1

nym impulsie zegarowym. Tablica wzbudzenia pokazuje, że aby zachować wartość wyjścia O, musimy mieć na wejściu $J = 0$ oraz dowolną wartość na wejściu K. Aby nastąpiła zmiana z 0 na 1, wejścia muszą być równe $J = 1$ i $K = 1$. Wartości te są pokazane w pierwszym wierszu tablicy. Droga podobnego rozumowania można wypełnić pozostałą część tablicy.

Po utworzeniu tablicy prawdy z rys. A.33a widzimy, że tablica ta pokazuje wymagane wartości wszystkich wejść J i K jako funkcje bieżących wartości A, B i C. Za pomocą map Karnaugh'a możemy znaleźć wyrażenia Boole'a dla tych sześciu funkcji. Pokazano to w części b rysunku. Na przykład mapa Karnaugh'a dla zmiennej Ja (wejście J przerzutnika, który daje na wyjściu wartość A) prowadzi do wyrażenia $J_a = BC$. Po wyprowadzeniu wszystkich sześciu wyrażeń zaprojektowanie układu jest rzeczą prostą, co widać w części c rysunku.

A.5

POLECANA LITERATURA

Literatura dotycząca tego tematu jest ogromna, wobec czego następujące rekomendacje dotyczą jedynie próbek tego, co jest osiągalne. [FLOY90] i [REIS91] są pozycjami łatwymi do przestudiowania i jednocześnie w pełni pokrywają materiał tego dodatku. Bardziej rygorystyczne ujęcie, które warto polecić, zawarto w [MANO93] i [WAKE90]. Inną dobrą książką, o nieco innym rozkładzie akcentów niż w większości pozycji, jest [UNGE89].

FLOY90 Floyd T.: *Digital Fundamentals*. New York, Merrill/Macmillan, 1990.

MANO93 Mano M.: *Computer System Architecture*. Englewood Cliffs, Prentice Hall, 1993.

REIS91 Reis R.: *Digital Electronics Through Project Analysis*. New York, Merrill/Macmillan, 1991.

UNGE89 Unger S.: *The Essence of Logic Circuits*. Englewood Cliffs, Prentice-Hall, 1989.

WAKE90 Wakerly J.: *Digital Design Principles and Practices*. Englewood Cliffs, Prentice-Hall, 1990.

A.6

PROBLEMY DO ROZWIĄZANIA

A.1. Zbuduj tablice prawdy dla następujących wyrażeń Boole'a:

(a) $ABC + \overline{ABC}$	(b) $ACB + ABC + \overline{ABC}$
(c) $A(\overline{BC} + \overline{BC})$	(d) $(A + B)(A + C)(\overline{A} + \overline{B})$

A.2. Uprość następujące wyrażenia zgodnie z prawem przemienności:

- | |
|--|
| (a) $A \cdot \overline{B} + \overline{B} \cdot A + C \cdot D \cdot E + \overline{C} \cdot D \cdot E + \overline{C} \cdot D \cdot E + E \cdot \overline{C} \cdot D$ |
| (b) $A \cdot B + A \cdot C + B \cdot A$ |

$$(c) \quad (L \cdot M \cdot N)(A \cdot B)(C \cdot D \cdot E)(M \cdot N \cdot L)$$

$$(d) \quad F \cdot (K + R) + S \cdot V + W \cdot \bar{X} + V \cdot S + \bar{X} \cdot W + (R + K) \cdot F$$

A.3. Zastosuj twierdzenia De Morgana do następujących równań:

$$(a) \quad F = \overline{V + A + L}$$

$$(b) \quad F = \bar{A} + \bar{B} + \bar{C} + \bar{D}$$

A.4. Uprość następujące wyrażenia:

$$(a) \quad A = S \cdot T + V \cdot W + R \cdot S \cdot T$$

$$(b) \quad A = T \cdot U \cdot V + X \cdot Y + Y$$

$$(c) \quad A = F \cdot (E + F + G)$$

$$(d) \quad A = (P \cdot Q + R + S \cdot T)T \cdot S$$

$$(e) \quad A = \overline{\bar{D} \cdot \bar{D} \cdot E}$$

$$(f) \quad A = Y \cdot (W + X + \overline{\bar{Y} + \bar{Z}}) \cdot Z$$

$$(g) \quad A = (B \cdot E + C + F) \cdot C$$

A.5. Zbuduj operację XOR z podstawowych operacji Boole'a AND, OR i NOT.

A.6. Dane jest następujące wyrażenie Boole'a:

$$F = \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}C$$

(a) Wyprowadź wyrażenie równoważne, używając wyłącznie operacji NAND, i narysuj schemat logiczny.

(b) Wyprowadź wyrażenie równoważne, używając wyłącznie operacji NOR, i narysuj schemat logiczny.

A.7. Dysponując bramką NOR i bramkami NOT, narysuj schemat logiczny układu realizującego funkcję AND z 3 wejściami.

A.8. Napisz wyrażenie Boole'a dla 4-wejściowej bramki NAND.

A.9. Do sterowania 7-segmentowym wskaźnikiem cyfr dziesiętnych jest używany układ kombinacyjny, co pokazano na rys. A.34. Układ ma 4 wejścia doprowadzające kod 4-bitowy używany w upakowanej reprezentacji dziesiętnej ($0_{10} = 0000$, ..., $9_{10} = 1001$). Siedem wyjść określa, które segmenty są wzbudzone w celu wyświetlenia danej cyfry dziesiętnej. Zauważ, że niektóre kombinacje wejść i wyjść są niepotrzebne.

(a) Zbuduj tablicę prawdy dla tego układu.

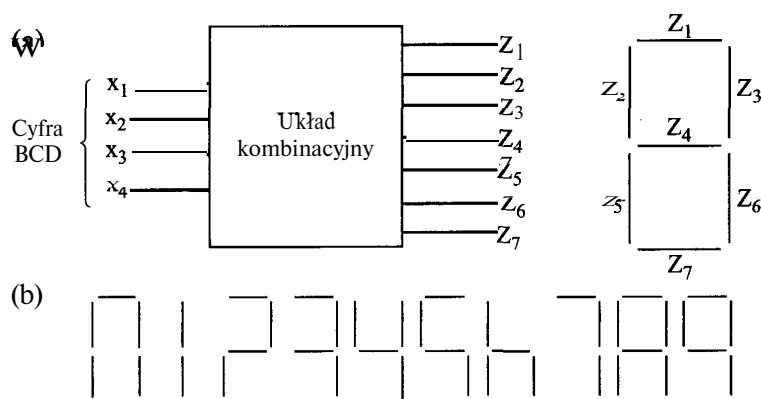
(b) Wyraż tablicę prawdy w formie sumy iloczynów.

(c) Wyraż tablicę prawdy w formie iloczynu sum.

(c) Wyprowadź wyrażenie uproszczone.

A. 10. Zaprojektuj multiplekser 8 do 1.

A. 11. Dodaj linię do schematu przedstawionego na rys. A. 15, tak żeby układ ten pełnił rolę demultipleksa.



RYSUNEK A.34. Przykład 7-segmentowego wyświetlacza na diodach świecących (LED)

A. 12. Kod Graya jest kodem binarnym liczb całkowitych. Różni się on od zwykłej reprezentacji binarnej tym, że między reprezentacjami dwóch dowolnych liczb istnieje tylko różnica jednego bitu*. Jest to przydatne w takich zastosowaniach, jak liczniki lub konwertery analogowo-cyfrowe, w których generowana jest sekwencja liczb. Ponieważ w określonej chwili ulega zmianie tylko jeden bit, nigdy nie występuje niejednoznaczność spowodowana przez niewielkie różnice taktowania. Oto pierwszych 8 elementów kodu:

Kod binarny	Kod Graya
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

Zaprojektuj układ, który przekształca kod binarny na kod Graya.

A.13. Zaprojektuj dekodery 5 x 32, używając 4 dekodery 3 x 8 (z wejściami zezwolenia) i jednego dekodera 2 x 4.

A. 14. Zaprojektuj realizację z wykorzystaniem układów PLA i ROM dla następującego zbioru funkcji kombinacyjnych:

$$\begin{aligned} Z_1 = & X_1 X_2 X_3 \bar{X}_4 + \bar{X}_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 + \bar{X}_1 \bar{X}_2 \bar{X}_3 X_4 + \bar{X}_1 \bar{X}_2 X_3 \bar{X}_4 + \\ & + \bar{X}_1 \bar{X}_2 X_3 X_4 + \bar{X}_1 X_2 \bar{X}_3 X_4 + \bar{X}_1 X_2 X_3 \bar{X}_4 \end{aligned}$$

$$Z_2 = X_1 X_2 X_3 X_4 + \bar{X}_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 + \bar{X}_1 \bar{X}_2 \bar{X}_3 X_4 + \bar{X}_1 \bar{X}_2 X_3 \bar{X}_4 + \bar{X}_1 \bar{X}_2 X_3 X_4$$

$$Z_3 = X_1 X_2 \bar{X}_3 X_4 + X_1 \bar{X}_2 X_3 X_4 + \bar{X}_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 + \bar{X}_1 \bar{X}_2 X_3 \bar{X}_4 + \bar{X}_1 X_2 \bar{X}_3 \bar{X}_4$$

* Chodzi o liczby kolejne (przyp. tłum.).

- A. 15. Opracuj realizację pełnego sumatora z rys. A.22, używając tylko 5 bramek (rada: niektóre z tych bramek to bramki XOR).
- A. 16. Rozważ rys. A.22. Załóż, że każda bramka powoduje opóźnienie 10 ns. Wobec tego wyjście sumatora jest ważne po 30 ns, a wyjście przeniesienia po 0 ns. Jaki jest całkowity czas operacji dodawania sumatora 32-bitowego zrealizowanego:
- (a) bez układu przeniesienia na bardziej znaczące pozycje, jak na rys. A.21?
 - (b) z układem przeniesienia na bardziej znaczące pozycje i przy użyciu sumatorów 8-bitowych, jak na rys. A.23?
- A. 17. Narysuj realizację przerzutnika D, używając tylko bramek NAND.

Słownik

Niektóre terminy w tym słowniku pochodzą z *American National Directory for Information Systems* [ANSI90]. Zostały one zaznaczone gwiazdką.

Adres bazowy* Wartość numeryczna używana jako odniesienie przy obliczaniu adresów podczas wykonywania programu komputerowego.

Adres bezpośredni* Adres, który określa miejsce przechowywania elementu danych, który ma być traktowany jako argument. Synonim *adresu zeropoziomowego*.

Adres bezwzględny* Adres w języku komputerowym identyfikujący miejsce przechowywania lub urządzenie bez użycia jakiegokolwiek odniesienia pośredniego.

Adres indeksowany* Adres, który jest modyfikowany przez zawartość rejestru modyfikacji przed lub podczas wykonywania rozkazu komputerowego.

Adres pośredni* Adres miejsca przechowywania, które zawiera adres.

Akumulator Nazwa rejestru procesora w jednoadresowym formacie rozkazu. Akumulator (AC) zawiera domyślnie jeden lub dwa argumenty rozkazu.

Argument* Obiekt, na którym wykonywana jest operacja.

ASCII Kod 7-bitowy używany do reprezentowania drukowalnych znaków numerycznych, alfabetycznych i specjalnych. Obejmuje również kody znaków sterujących, które nie są drukowane, ani wyświetlane, lecz określają pewną funkcję sterowania.

Bajt Osiem bitów. Określany również jako oktet.

Bit* W czystym binarnym systemie liczenia, jedna z cyfr 0 i 1.

Bit parzystości* Cyfra binarna dołączona do grupy cyfr binarnych w ten sposób, że suma wszystkich cyfr jest zawsze nieparzysta (bit nieparzystości) lub parzysta (bit parzystości).

Blok sterowania procesami Reprezentacja procesu w systemie operacyjnym. Jest to struktura danych zawierająca informację o własnościach i stanie procesu.

Blok stronicowy* Obszar pamięci głównej używany do przechowywania strony.

- Błąd strony** Ma miejsce, gdy strona zawierająca słowo, do którego nastąpiło odniesienie, nie znajduje się w pamięci głównej. Powoduje to przerwanie i wymaga, żeby system operacyjny dostarczył potrzebną stronę.
- Bramka** Układ elektroniczny, którego sygnał wyjściowy jest prostą funkcją Boole'a jego sygnałów wejściowych.
- Bufor*** Pamięć używana do skompensowania różnicy szybkości przepływu danych lub czasu następowania zdarzeń, podczas przesyłania danych z jednego przyrządu do drugiego.
- CD-ROM** Pamięć stała na dysku kompaktowym. Niewymazywalny dysk używany do przechowywania danych komputerowych. Standardowe dyski mają średnicę 12 cm i pojemność ponad 650 MB.
- Cykl adresowania pośredniego** Część cyklu rozkazu, podczas której procesor wykonuje operację dostępu do pamięci w celu przekształcenia adresu pośredniego na bezpośredni.
- Cykl pobierania** Część cyklu rozkazu, podczas której procesor pobiera z pamięci rozkaz przeznaczony do wykonywania.
- Cykl przerwania** Część cyklu rozkazu, podczas której procesor sprawdza, czy nie wystąpiło przerwanie. Jeśli w stanie zawieszenia znajduje się przerwanie dozwolone, to zachowuje on stan bieżący programu i wznowia przetwarzanie po zrealizowaniu programu obsługi przerwania.
- Cykl rozkazu** Przetwarzanie prowadzone przez procesor w celu wykonania pojedynczego rozkazu.
- Cykl wykonywania** Część cyklu rozkazu, podczas której procesor wykonuje operację określoną przez kod operacji.
- Czas cyklu pamięci** Odwrotność częstotliwości, z jaką mogą następować operacje dostępu do pamięci. Jest to minimalny czas między odpowiedzią na jedno zgłoszenie dostępu (odczytu lub zapisu) a odpowiedzią na następne zgłoszenie dostępu.
- Czas cyklu procesora** Czas potrzebny do wykonania najkrótszej, ściśle określonej mikrooperacji procesora. Jest to podstawowa jednostka czasu do mierzenia czasu trwania wszystkich działań procesora. Synonim *czasu cyklu maszynowego*.
- Dekoder*** Przyrząd mający pewną liczbę linii wejściowych, z których dowolna liczba może przenosić sygnały, oraz pewną liczbę linii wyjściowych, z których nie więcej niż jedna może przenosić sygnał, przy czym istnieje jednoznaczne przyporządkowanie między wyjściami a kombinacjami sygnałów wejściowych.
- Dostęp bezpośredni*** Możliwość uzyskania danych z przyrządu pamięciowego lub wprowadzenia danych do tego przyrządu w sekwencji niezależnej od względnego położenia tych danych, za pomocą adresu wskazującego fizyczną lokację danych.
- Dostęp bezpośredni do pamięci (DMA)** Postać wejścia-wyjścia, w której specjalny moduł, nazywany modułem DMA, steruje wymianą danych między

pamięcią główną a modulem wejścia-wyjścia. Procesor wysyła zapotrzebowanie na przesłanie bloku danych do modułu DMA. Przerwanie procesora następuje dopiero po zakończeniu transferu całego bloku.

Dysk kompaktowy (CD) Niewymazywalny dysk przechowujący informację audio w postaci cyfrowej.

Dysk magnetyczny* Płaska, okrągła płytką pokryta warstwą magnetyczną, na której powierzchni mogą być przechowywane dane wprowadzone metodą zapisu magnetycznego.

Dyskietka* Elastyczny dysk magnetyczny zamknięty w obudowie ochronnej. Synonim *dysku elastycznego*.

Emulacja* Imitowanie całości lub części jednego systemu przez inny system o charakterze przede wszystkim sprzętowym, dzięki czemu system imitujący akceptuje takie same dane, wykonuje takie same programy i uzyskuje takie same wyniki, jak system imitowany.

Format rozkazu Struktura rozkazu komputerowego jako sekwencji bitów. Format dzieli rozkaz na pola odpowiadające elementom składowym rozkazu (np. kod operacji, argumenty).

G Przedrostek oznaczający miliard.

Indeksowanie Metoda modyfikacji adresu za pomocą rejestrów indeksowych.

Indeksowanie automatyczne Forma adresowania indeksowego, w której wartość rejestru indeksowego jest automatycznie inkrementowana lub dekrementowana za każdym odniesieniem do pamięci.

Izolowane wejście-wyjście Metoda adresowania modułów wejścia-wyjścia i urządzeń zewnętrznych. Przestrzeń adresowa wejścia-wyjścia jest traktowana oddzielnie od przestrzeni adresowej pamięci głównej. Muszą być używane specjalne rozkazy maszynowe wejścia-wyjścia. Porównaj *wejście-wyjście odwzorowane w pamięci*.

Jądro Część systemu operacyjnego zawierająca jego podstawowe i najczęściej używane funkcje. Często jądro rezyduje w pamięci głównej na stałe.

Jednokrotny zapis, wielokrotny odczyt (WORM) Dysk, na którym łatwiej można dokonać zapisu niż na dysku CD-ROM, dzięki czemu dyski w jednym egzemplarzu o unikalnej zawartości są dostępne w handlu. Podobnie jak dysk CD-ROM, dysk taki po dokonaniu zapisu służy tylko do odczytu. Najpopularniejsze dyski mają średnicę 5,25 cala; ich pojemność wynosi wówczas 200÷800MB.

Jednostka arytmetyczno-logiczna (ALU)* Część komputera, która wykonuje operacje arytmetyczne, logiczne i pokrewne.

Jednostka centralna (CPU) Część komputera, która pobiera i wykonuje rozkazy. Składa się z jednostki arytmetyczno-logicznej, jednostki sterującej i rejestrów. Często jest określana po prostu jako procesor.

Jednostka sterująca Część procesora, która steruje jego operacjami, w tym operacjami ALU, przenoszeniem danych wewnątrz procesora oraz wymianą danych i sygnałów sterujących poprzez interfejsy zewnętrzne (np. magistralę systemową).

Język assemblerowy* Język komputerowy, którego rozkazy zwykle odpowiadają rozkazom komputera i który może zawierać udogodnienia w postaci makrorozkazów. Synonim *języka zależnego od komputera*.

Język mikroprogramowania Lista rozkazów służąca do tworzenia mikroprogramów.

K Przedrostek oznaczający $2^{10} = 1024$. Wobec tego 2 Kbit = 2048 bitów.

Kanał bajtowo-multiplekserowy* Kanał multiplekserowy, który przeplata przekazywane bajty danych. Patrz również *kanał multiplekserowy z blokową transmisją danych*. Rozwiązanie przeciwstawne - *kanał wybiórczy*.

Kanał multiplekserowy Kanał przeznaczony do jednoczesnej pracy z wieloma urządzeniami wejścia-wyjścia. Kilka urządzeń wejścia-wyjścia może przekazywać rekordy w tym samym czasie dzięki przeplatanej transmisji elementów danych. Patrz również *kanał bajtowo-multiplekserowy* i *kanał multiplekserowy z blokową transmisją danych*.

Kanał multiplekserowy z blokową transmisją danych Kanał multiplekserowy, który przeplata przekazywane bloki danych. Patrz również *kanał bajtowo-multiplekserowy*. Rozwiązanie przeciwstawne - *kanał wybiórczy*.

Kanał wejścia-wyjścia Stosunkowo złożony moduł wejścia-wyjścia, który uwalnia procesor od szczegółów operacji wejścia-wyjścia. Kanał wejścia-wyjścia wykona sekwencję rozkazów wejścia-wyjścia z pamięci głównej bez potrzeby angażowania procesora.

Kanał wybiórczy Kanał wejścia-wyjścia przeznaczony do współpracy z tylko jednym urządzeniem wejścia-wyjścia w określonym czasie. Gdy urządzenie wejścia-wyjścia zostało wybrane, kompletny rekord jest przekazywany bajt po bajcie. Rozwiązanie przeciwstawne - *kanał multiplekserowy z blokową transmisją danych* oraz *kanał multiplekserowy*.

Kod detekcyjny* Kod, w którym każdy znak lub sygnał spełniają określone reguły budowy, dzięki czemu odchylenia od tych reguł wskazują na obecność błędu.

Kod korekcyjny* Kod, w którym każdy znak lub sygnał spełniają określone reguły budowy (dzięki czemu odchylenia od tych reguł wskazują na obecność błędu) i w którym niektóre lub wszystkie wykryte błędy mogą być automatycznie poprawiane.

Kod operacji* Kod używany do reprezentowania operacji komputera.

Kod warunkowy Kod odzwierciedlający wynik poprzedniej operacji (np. arytmetycznej). Procesor może obejmować jeden lub wiele kodów warunkowych, które mogą być przechowywane oddzielnie wewnątrz procesora lub jako część większego rejestru sterowania. Znany także jako *znaczniki stanu*.

Licznik rozkazów Rejestr adresów rozkazów.

Lista rozkazów* Kompletny zbiór operatorów rozkazów komputera wraz z opisem rodzajów znaczeń, które mogą być przypisane ich argumentom. Synonim *listy rozkazów maszynowych*.

M Przedrostek oznaczający $2^{20} = 1\,048\,576$. Wobec tego 2 Mbit = 2×2^{20} bitów.

Magistrala Wspólna ścieżka komunikacji składająca się z jednej linii lub ze zbioru linii. W niektórych systemach komputerowych procesor, pamięć oraz urządzenia wejścia-wyjścia są połączone za pomocą wspólnej magistrali. Ponieważ linie są używane wspólnie przez wszystkie zespoły, tylko jeden z nich może transmitować w określonej chwili.

Magistrala systemowa Magistrala służąca do łączenia głównych zespołów komputera (procesora, pamięci, wejścia-wyjścia).

Mikrokomputer* System komputerowy, którego procesor jest mikroprocesorem. Podstawowy mikrokomputer zawiera mikroprocesor, pamięć i urządzenia wejścia-wyjścia, które mogą, lecz nie muszą znajdować się w tym samym mikroukładzie.

Mikrooperacja Elementarna operacja procesora, wykonywana podczas jednego impulsu zegara.

Mikroprocesor* Procesor, którego elementy zostały zminiaturyzowane w postaci jednego lub kilku układów scalonych.

Mikroprogram* Sekwencja mikrorozkazów znajdująca się w specjalnej pamięci, z której mogą być dynamicznie pobierane w celu wykonywania różnych funkcji.

Mikroprogramowana jednostka centralna Procesor, którego jednostka sterująca została wdrożona przy użyciu mikroprogramowania.

Mikrorozkaz* Rozkaz sterujący przepływem danych i szeregowaniem w procesorze na poziomie bardziej podstawowym niż rozkazy maszynowe. Pojedyncze rozkazy maszynowe i być może inne funkcje mogą być realizowane za pomocą mikroprogramów.

Moduł wejścia-wyjścia Jeden z głównych rodzajów zespołów komputera. Jest odpowiedzialny za sterowanie jednym lub wieloma urządzeniami zewnętrznymi (peryferyjnymi) oraz za wymianę danych między tymi urządzeniami a pamięcią główną i (lub) rejestrami procesora.

Multiplekser Układ kombinacyjny, który łączy wiele wejść z jednym wyjściem. W dowolnej chwili tylko jedno z wejść jest wybrane jako połączone z wyjściem.

Operator binarny* Operator, który reprezentuje operację na dwóch i tylko dwóch argumentach.

Operator jednoargumentowy* Operator, który reprezentuje operację na jednym i tylko jednym argumentem.

Oprogramowanie układowe* Mikroprogram przechowywany w pamięci stałej.

Ortogonalność Zasada, zgodnie z którą dwie zmienne lub wymiary są od siebie wzajemnie niezależne. W kontekście listy rozkazów termin ten jest używany do wskazania, że pozostałe składniki rozkazu (tryb adresowania, liczba argumentów, długość argumentu) są niezależne od kodu operacji (nie są przez ten kod wyznaczone).

Pakiet dysków* Zespół dysków magnetycznych, który może być usunięty jako całość z napędu, łącznie z pojemnikiem, od którego zespół musi być oddzielony podczas pracy.

Pamięć domenowa (pęcherzykowa) Magnetyczny przyrząd pamięciowy. Binarne jedynki i zera są reprezentowane przez obecność lub nieobecność mikroskopijnych domen magnetycznych zwanych *bąbelkami*.

Pamięć dynamiczna RAM Pamięć RAM, której komórki są zrealizowane w postaci kondensatorów. Dynamiczne pamięci RAM stopniowo tracą swoją zawartość, czemu zapobiega okresowe odświeżanie.

Pamięć główna* Pamięć adresowalna przez program, z której rozkazy i inne dane mogą być ładowane bezpośrednio do rejestrów w celu dalszego wykonywania lub przetwarzania.

Pamięć nieulotna Pamięć, której zawartość jest stabilna i nie wymaga stałego zasilania.

Pamięć o dostępie swobodnym (RAM) Pamięć, w której każda adresowalna lokacja ma unikatowy sposób adresowania. Czas dostępu do danej lokacji jest niezależny od sekwencji poprzednichostępów.

Pamięć podręczna* Specjalna pamięć buforowa, mniejsza i szybsza niż pamięć główna, używana do przechowywania kopii tych rozkazów i danych z pamięci głównej, które najprawdopodobniej będą potrzebne procesorowi jako następne i które zostały automatycznie uzyskane z pamięci głównej.

Pamięć pomocnicza Pamięć umieszczona poza samym systemem komputerowym, w tym dyskowa i taśmowa.

Pamięć skojarzeniowa* Pamięć, w której lokacje są identyfikowane przez ich zawartość lub częściowo przez ich zawartość, nie zaś przez ich nazwy czy położenie.

Pamięć tylko do odczytu (stała) (ROM) Pamięć półprzewodnikowa, której zawartość nie może być zmieniana bez zniszczenia tej pamięci. Pamięć niewymazywalna.

Pamięć sterująca Część pamięci zawierająca mikroprogram.

Pamięć ulotna Pamięć, w przypadku której w celu zachowania zawartości jest wymagane ciągłe zasilanie elektryczne. Po wyłączeniu zasilania przechowywana informacja jest stracona.

Pamięć wirtualna* Przestrzeń pamięci, która może być traktowana przez użytkownika systemu komputerowego jako adresowalna pamięć główna i w której adresy wirtualne są odwzorowane w zbiorze adresów rzeczywistych. Rozmiar pamięci wirtualnej jest ograniczony przez schemat adresowania systemu komputerowego oraz przez wielkość dostępnej pamięci pomocniczej, nie zaś przez rzeczywistą liczbę lokacji w pamięci głównej.

Paskowanie dysków Rodzaj organizacji danych w zespole dysków, polegający na tym, że logicznie sąsiadujące bloki danych (lub paski danych) są rozlokowane cyklicznie na kolejnych dyskach zespołu. Zbiór logicznie sąsiadujących pasków danych, odwzorowany w postaci dokładnie jednego paska na każdym dysku w zespole, jest określany jako pasek.

Podstawa naukowego systemu liczenia* W naukowym systemie liczenia jest to liczba, która jest podnoszona do potęgi określonej przez wykładnik, a następnie mnożona przez mantysę w celu otrzymania rzeczywistej reprezentowanej liczby (np. liczba 6,25 w wyrażeniu $2,7 \times 6,25^{1,5} = 42,1875$).

Podzespoły na ciele stałym* Podzespoły, których działanie zależy od sterowania zjawiskami elektrycznymi lub magnetycznymi w ciałach stałych (np. tranzystor, dioda półprzewodnikowa, rdzeń ferrytowy).

Półprzewodnik Stała substancja krystaliczna, taką jak krzem lub german, którego przewodnictwo elektryczne jest pośrednie między izolatorami a przewodnikami[♦]. Półprzewodników używa się do wytwarzania tranzystorów i innych podzespołów na ciele stałym.

Proces Wykonywany program. Proces jest sterowany i szeregowany przez system operacyjny.

Procesor* Jednostka funkcjonalna komputera, która interpretuje i wykonuje rozkazy. Procesor składa się przynajmniej z jednostki sterującej i jednostki arytmetycznej.

Procesor superskalarny Rodzaj procesora, który zawiera wiele potoków rozkazów dzięki czemu na tym samym etapie potoku można wykonywać więcej niż jeden rozkaz.

Procesor wejścia-wyjścia Moduł wejścia-wyjścia z własnym procesorem, który może wykonywać specjalistyczne rozkazy maszynowe wejścia-wyjścia, a także, w pewnych przypadkach, rozkazy maszynowe o ogólnym przeznaczeniu.

Procesor z przetwarzaniem superpotokowym Rodzaj procesora, w którym potok rozkazów składa się z bardzo małych etapów, w wyniku czego więcej niż jeden etap potoku może być wykonany podczas jednego cyklu zegara. Dzięki temu wielka liczba rozkazów może się jednocześnie znajdować w potoku.

Programowalna pamięć stała (PROM) Pamięć półprzewodnikowa, której zawartość może być ustalona jednorazowo. Proces zapisu jest realizowany elektrycznie i może być dokonywany przez użytkownika po wytworzeniu oryginalnego mikroukładu.

Programowalna tablica logiczna (PLA)* Tablica bramek, których połączenia mogą być programowane w celu realizowania określonej funkcji logicznej.

Programowane wejście-wyjście Forma wejścia-wyjścia, w której procesor wydaje modułowi wejścia-wyjścia rozkaz wejścia-wyjścia, po czym musi czekać na zakończenie operacji, zanim będzie kontynuował działanie.

* Bardziej istotną cechą półprzewodników jest wrażliwość ich przewodnictwa elektrycznego na pole elektryczne, magnetyczne, światło i temperaturę (*przyp.tłum.*).

- Przerwanie*** Zawieszenie procesu, takiego jak wykonywanie programu komputerowego, spowodowane przez zdarzenie zewnętrzne w stosunku do tego procesu i wykonywane w taki sposób, że proces może być wznowiony.
- Przerwanie dozwolone** Warunek, tworzony zwykle przez procesor, podczas którego odpowiada on na sygnały zapotrzebowania na przerwanie określonej klasy.
- Przerwanie zablokowane** Warunek, tworzony zwykle przez procesor, podczas którego ignoruje on sygnały zapotrzebowania na przerwanie określonej klasy.
- Przerzutnik*** Układ lub przyrząd zawierający elementy aktywne. Może on przyjmować w danym czasie jeden z dwóch stanów stabilnych. Synonim *układu dwustabilnego*.
- Przestrzeń adresowa** Zakres adresów (pamięci, wejścia-wyjścia), do których mogą następować odniesienia.
- Przetwarzanie jednoprocessorowe** Sekwencyjne wykonywanie rozkazów przez procesor lub niezależne używanie procesora w systemie wieloprocessorowym.
- Ramka strony** Część pamięci głównej używana do przechowywania strony.
- Rejestr adresowy pamięci (MAR)*** Rejestr w procesorze zawierający adres tej lokacji w pamięci, do której ma miejsce dostęp.
- Rejestr adresu rozkazu*** Specjalistyczny rejestr służący do przechowywania adresu następnego rozkazu przewidzianego do wykonania.
- Rejestr buforowy pamięci (MBR)** Rejestr zawierający dane odczytane w pamięci lub dane, które mają być zapisane w pamięci.
- Rejestr indeksowy*** Rejestr, którego zawartość może być użyta do zmodyfikowania adresu argumentu podczas wykonywania rozkazów komputerowych; może być również użyty jako licznik. Rejestr indeksowy może służyć do sterowania wykonywaniem pętli, do sterowania wykorzystaniem tablic, jako przełącznik, do przeglądania tablic lub jako wskaźnik.
- Rejestr ogólnego przeznaczenia (roboczy)*** Rejestr, zwykle jawnie adresowalny w ramach zbioru rejestrów, który może być używany do różnych celów, na przykład jako akumulator, jako rejestr indeksowy lub jako specjalny rejestr danych.
- Rejestr rozkazu*** Rejestr używany do przechowywania rozkazu przeznaczonego do interpretacji.
- Rejestry** Szybka pamięć wewnętrzna procesora. Niektóre rejestry są widzialne dla użytkownika, to znaczy dostępne dla programisty poprzez listę rozkazów maszynowych. Inne rejestry są używane wyłącznie przez procesor do celów sterowania.
- Rejestry sterujące** Rejestry procesora służące do sterowania jego działaniem. Większość z nich jest niewidzialna dla użytkownika.
- Rejestry widzialne dla użytkownika** Rejestry procesora, do których może się odnosić programista. Format listy rozkazów pozwala na to, żeby jeden lub

więcej rejestrów było przeznaczonych do przechowywania argumentów lub adresów argumentów.

Reprezentacja dopełnienia do dwóch Używana do reprezentowania całkowitych liczb binarnych. Dodatnia całkowita liczba binarna jest reprezentowana w notacji znak-moduł. Liczba ujemna jest reprezentowana przez dodanie 1 do reprezentacji uzupełnienia do 1 tej samej liczby.

Reprezentacja znak-moduł Używana do reprezentowania całkowitych liczb binarnych. W słowie N -bitowym lewy bit określa znak (0 = dodatni, 1 = ujemny), a pozostałych $N - 1$ bitów określa wielkość.

Rozkaz komputerowy* Rozkaz, który może być rozpoznany przez procesor komputera, dla którego został zaprojektowany. Synonim *rozkażu maszynowego*.

Skalar* Wielkość określana za pomocą pojedynczej wartości.

Skok bezwarunkowy* Skok, który następuje przy każdym wykonywaniu określonego rozkazu.

Skok warunkowy* Skok, który następuje tylko wtedy, gdy wykonywany jest określony rozkaz i gdy jednocześnie spełnione są określone warunki; w odróżnieniu od *skoku bezwarunkowego*.

Słowo stanu programu (PSW) Obszar w pamięci używany do wskazywania kolejności wykonywania rozkazów oraz do przechowywania i wskazywania stanu systemu komputerowego. Synonim *słowa stanu procesora*.

Statyczna pamięć o dostępie swobodnym Pamięć RAM, której komórki zostały zrealizowane za pomocą przerzutników. Statyczna pamięć RAM przechowuje dane tak długo, jak długo jest zasilana; nie jest wymagane okresowe odświeżanie.

Sterownik wejścia-wyjścia Stosunkowo prosty moduł wejścia-wyjścia, który wymaga szczegółowego sterowania ze strony procesora lub kanału wejścia-wyjścia. Synonim *sterownika urządzenia*.

Stos* Lista zbudowana i utrzymywana w ten sposób, że następna pozycja przewidziana do wydobywania jest tą **pozycją**, która ostatnio została na tej liście umieszczona (ostatnia weszła, pierwsza wychodzi - LIFO).

Strona* W systemie pamięci wirtualnej blok o ustalonej długości, który ma adres wirtualny i który jest przenoszony jako całość między pamięcią rzeczywistą a pamięcią pomocniczą.

Stronicowanie na żądanie* Transfer strony z pamięci pomocniczej do rzeczywistej w czasie, gdy jest potrzebna.

System operacyjny* Oprogramowanie sterujące wykonywaniem programów i wykonujące takie usługi, jak rozdzielanie zasobów, szeregowanie, sterowanie wejściem-wyjściem i zarządzanie danymi.

System reprezentacji liczb stałopozycyjnych* System liczenia o stałej podstawie, w którym miejsce przecinka jest domyślnie ustalone w ciągu cyfr za pomocą uzgodnionej konwencji.

Szyna adresowa Część magistrali systemowej używana do przesyłania adresu. Adres określa zwykle lokację w pamięci głównej lub urządzenie wejścia-wyjścia.

Szyna danych Część magistrali systemowej używana do przesyłania danych.

Szyna sterująca Część magistrali systemowej używana do przesyłania sygnałów sterujących.

Tablica niezależnych dysków twardych (RAID) Tablica dysków, w której część fizycznej pojemności pamięci jest używana do przechowywania nadmiarowej (redundantnej) informacji o danych użytkownika zapisanych w pozostałej części pamięci. Informacja nadmiarowa umożliwia odtworzenie danych użytkownika w przypadku, gdy jeden z dysków wchodzących w skład tablicy lub ścieżka dostępu do niego są uszkodzone.

Tablica prawdy* Tablica opisująca funkcję logiczną poprzez wymienienie wszystkich możliwych wartości wejściowych i wskazanie wartości wyjściowej odpowiadającej każdej z tych kombinacji.

Taśma magnetyczna* Taśma z magnetyczną warstwą powierzchniową, na której można dokonywać magnetycznego zapisu danych.

Transmisja danych Przenoszenie danych między urządzeniami. Termin ten zwykle obejmuje wejście-wyjście.

Tryb asynchroniczny Metoda, w której następowanie zdarzeń na magistrali zależy od występowania poprzednich zdarzeń.

Tryb synchroniczny Metoda, w której następowanie zdarzeń na magistrali jest wyznaczone przez zegar. Zegar wyznacza przedziały czasowe o równej szerokości, a zdarzenia rozpoczynają się tylko na początku przedziału.

Układ kombinacyjny* Przyrząd logiczny, którego wartości wyjściowe w dowolnej chwili zależą wyłącznie od wartości wejściowych w tym samym czasie. Układ kombinacyjny jest szczególnym przypadkiem układu sekwencyjnego, który nie umożliwia przechowywania.

Układ łańcuchowy* Metoda łączenia przyrządów służąca do określania priorytetu przerwań, polegająca na szeregowym połączeniu źródeł przerwań.

Układ scalony (IC) Niewielki kawałek ciała stałego, takiego jak krzem, na którym wytworzono zbiór elementów elektronicznych i ich połączeń.

Układ sekwencyjny Cyfrowy układ logiczny, którego stan wyjścia zależy od bieżącego stanu wejścia oraz od stanu samego układu. Układy sekwencyjne mają więc atrybut pamięci.

Układ wejścia-wyjścia sterowany przerwaniami Forma wejścia-wyjścia. Procesor wydaje rozkaz wejścia-wyjścia i kontynuuje wykonywanie następnych rozkazów. Wykonywanie to ulega przerwaniu przez moduł wejścia-wyjścia, gdy ten ostami zakończył pracę.

Urządzenie peryferyjne Jakiegokolwiek urządzenie w systemie komputerowym, które zapewnia komunikację procesora z otoczeniem, zewnętrzne w stosunku do tego procesora.

Uzupełnienie do jedyńki Używane do reprezentowania całkowitych liczb binarnych. Dodatnia liczba całkowita jest reprezentowana w postaci znak-

-moduł. Ujemna liczba całkowita jest reprezentowana przez odwrócenie każdego bitu w reprezentacji dodatniej liczby całkowitej o tej samej wielkości.

Wejście-wyjście Odnosi się do wejścia albo do wyjścia, albo do obu tych pojęć. Oznacza przenoszenie danych między komputerem a bezpośrednio dołączonym urządzeniem peryferyjnym.

Wejście-wyjście odwzorowane w pamięci Metoda adresowania modułów wejścia-wyjścia i urządzeń zewnętrznych. Zarówno w stosunku do pamięci głównej, jak i do wejścia-wyjścia używana jest jedna przestrzeń adresowa. W odniesieniu do odczytu/zapisu w pamięci oraz w urządzeniach wejścia-wyjścia są stosowane takie same rozkazy maszynowe.

Wektor* Wielkość określana zwykle za pomocą uporządkowanego zbioru skalarów.

Wieloprogramowanie* Tryb pracy polegający na przemianowym wykonywaniu dwóch lub większej liczby programów komputerowych przez jeden procesor.

Wymazywalny dysk optyczny Dysk, w którym zastosowano technologię optyczną, który jednak może być z łatwością wymazany i zapisany ponownie. Używane są dyski o średnicach 3,25 i 5,25 cala.

Zmienna globalna Zmienna zdefiniowana w jednej części programu komputerowego i używana przez przynajmniej jedną inną część tego programu.

Zmienna lokalna Zmienna zdefiniowana i używana tylko w jednej, określonej części programu komputerowego.

Zmiennopozycyjny system reprezentacji System liczenia, w którym liczba rzeczywista jest reprezentowana przez parę liczb. Pierwsza z nich jest liczbą rzeczywistą będącą iloczynem części stałopozycyjnej, druga zaś jest wartością uzyskaną przez podniesienie domyślnej podstawy do potęgi określonej przez wykładnik.

Literatura

- ABAD83 Abadir M., Reghabati H.: „Functional Testing of Semiconductor Random Access Memories”. *Computing Surveys*, September 1983.
- ACOS86 Acosta R., Kjelstrup J., Torng H.: „An Instruction Issuing Approach to Enhancing Performance in Multiple Functional Unit Processors”. *IEEE Transactions on Computers*, September 1986.
- ADAM91 Adamek J.: *Foundations of Coding*. New York, Wiley, 1991.
- AGAR89 Agarwal A.: *Analysis of Cache Performance for Operating Systems and Multiprogramming*. Boston, Kluwer Academic Publishers, 1989.
- AGER87 Agerwala T., Cocke J.: *High Performance Reduced Instruction Set Processors*. Technical report RC12434 (#55845). Yorktown, IBM Thomas J. Watson Research Center, January 1987.
- AGRA76 Agrawala A., Rauscher T.: *Foundations of Microprogramming: Architecture, Software and Applications*. New York, Academic Press, 1976.
- ALEX93 Alexandridis N.: *Design of Microprocessor-Based Systems*. Englewood Cliffs, Prentice Hall, 1993.
- ALSU90 Alsup M.: „The Motorola's 88000 Family Architecture”. *IEEE Micro*, June 1990.
- ALTN79 Altnether J.: *Error Detecting and Correcting Codes*. Intel Application Note AP-46, 1979.
- AMDA79 Amdahl G.: „The early chapters of the PCM story”. *Datamation*, February 1979.
- ANDE67 Anderson D., Sparacio F., Tomasulo F.: „The IBM System 360 Model 91: Machine Philosophy and Instruction Handling”. *IBM Journal of Research and Development*, January 1967.
- ANDE93 Anderson D., Shanley T.: *Pentium Processor System Architecture*. Richardson, Mindshare Press, 1993.
- ANDR80 Andrews M.: *Principles of Firmware Engineering in Microprogram Control*. Silver Spring, Computer Science Press, 1980.
- ANDR90 Andrews W.: „Futurebus+ Spec Completed - Almost”. *Computer Design*, February 1, 1990.
- ANDR91 Andrews W.: „Enhancing the Performance of Standard Buses”. *Computer Design*, September 1991.
- ANSI90 American National Standard Institute. *American National Standard Dictionary for Information Systems*. ANSI X3.172-1990, 1990.
- ARDE80 Arden B.: *What Can Be Automated?* Cambridge, The MIT Press, 1980.

- AZIM92 Azimi M., Prasad B., Bhat K.: „Two Level Cache Architectures”. *Proceedings COMPCON'92*, February 1992.
- BANE82 Banerji D., Raymond J.: *Elements of Microprogramming* Englewood Cliffs, Prentice Hall, 1982.
- BARR68 Barron D.: *Recursive Techniques in Programming*. New York, American Elsevier, 1968.
- BART85 Bartee T.: *Digital Computer Fundamentals*. New York, McGraw-Hill, 1985.
- BASH81 Bashe C. i in.: „The Architecture of IBM's Early Computers”. *IBM Journal of Research and Development*, September 1981.
- BASH91 Bashteen A., Lui L., Mullan J.: „A Superpipeline Approach to the MIPS Architecture”. *Proceedings, COMPCONSpring 91*, February 1991.
- BECK93 Becker M. i in.: „The PowerPC 601 Microprocessor”. *IEEE Micro*, October 1993.
- BELL70 Bell C. i in.: „A New Architecture for Minicomputers - the DEC PDP-11”. *Proceedings, Spring Joint Computer Conference*, 1970.
- BELL71a Bell C., Newell A.: *Computer Structures: Readings and Examples*. New York, McGraw-Hill, 1971.
- BELL71b Bell C., Newell A.: „The IBM 1401”, w [BELL71a].
- BELL78a Bell C., Mudge J., McNamara J.: *Computer Engineering: a DEC View of Hardware Systems Design*. Bedford, Digital Press, 1978.
- BELL78b Bell C., Newell A., Siewiorek D.: „Structural levels of the PDP-8”, w [BELL78a].
- BELL78c Bell C. i in.: „The evolution of the DEC system 10”. *Communications of the ACM*, January 1978.
- BHAN79 Bhandarkar D., Rothman S.: „The VAX-11, DEC's 32-bit Version of the PDP-11”. *Datamation*, February 1979.
- BLAH83 Blahut R.: *Theory and Practice of Error Control Codes*. Reading, Addison-Wesley, 1983.
- BOND83 Bond R.: „XA: the View from White Plains”. *Datamation*, May 1983.
- BOND94 Bondurant D.: „Low Latency EDRAM Main Memory Subsystem for 66 MHz Bus Operation”. *Proceedings, COMPCON'94*, March 1994.
- BOOT51 Booth A.: „A Signed Binary Multiplication Technique”. *Quarterly Journal of Mechanical and Applied Mathematics*, t. 4, p. 2, 1951.
- BRAD91a Bradlee D., Eggers S., Henry R.: „The Effect on RISC Performance of Register Set Size and Structure Versus Code Generation Strategy”. *Proceedings, 18th Annual International Symposium on Computer Architecture*, May 1991.
- BRAD91b Bradlee D., Eggers S., Henry R.: „Integrating Register Allocation and Instruction Scheduling for RISCs”. *Proceedings, Forth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.
- BREY95 Brey B.: *The Intel 32-bit Microprocessors: 80386, 80486 and Pentium*. Englewood Cliffs, Prentice Hall, 1995.
- BUCH86 Buchholz W.: „The IBM system 370 vector architecture”. *IBM Systems Journal*, nr 1, 1986.
- BUDD87 Buddine L., Young E.: *The Brady Guide to CD-ROM*. New York, Prentice Hall Press, 1987.
- BURG75 Burge W.: *Recursive Programming Techniques*. Reading, Addison-Wesley, 1975.
- BURK46 Burks A., Goldstine H., von Neumann J.: *Preliminary Discussion of the Logical Design of an Electronic Computer Instrument*. Report prepared for U.S. Army Ordnance Dept., 1946, przedrukowany w [BELL71a].

- CATA94 Catanzaro B.: *Multiprocessor system architectures*. Mountain View, Sunsoft Press, 1994.
- CHAI82 Chaitin G.: „Register Allocation and Spilling via Graph Coloring”. *Proceedings, SIGPLAN Symposium on Compiler Construction*, June 1982.
- CHEN94 Chen P. i in.: „RAID: High Performance, Reliable Secondary Storage”. *ACM Computing Surveys*, June 1994.
- CHOW86 Chow F. i in.: „Engineering a RISC compiler system”. *Proceedings, COMPCON Spring '86, March* 1986.
- CHOW87 Chow F. i in.: „How Many Addressing Modes Are Enough?” *Proceedings, Second International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1987.
- CHOW90 Chow F., Hennessy J.: „The Priority-Based Coloring Approach to Register Allocation”. *ACM Transactions on Programming Languages*, October 1990.
- CLAR80 Clark W.: „From Electron Mobility to Logical Structure: a View of Integrated Circuits”. *Computing Surveys*, September 1980.
- CLAR82 Clark D., Levy H.: „Measurement and Analysis of Instruction Use in the VAX-11/780.” *Proceedings, Ninth Annual Symposium on Computer Architecture*, April 1982.
- CLAR83 Clark D.: „Cache performance in the VAX-11/780.” *ACM Transactions on Computer Systems*, February 1983.
- CLAR85 Clark D., Emer J.: „Performance of the VAX-11/780 Translation Buffer: Simulation and Measurement.” *ACM Transactions on Computer Systems*, February 1985.
- CLIN81 Cline B.: *Microprogramming Concepts and Techniques*. New York, Petrocelli, 1981.
- CODY84 Cody W., i in.: „A Proposed Radix- and Word-Length-Independent Standard for Floating-Point Arithmetic”. *IEEE Micro*, August 1984.
- COHE81 Cohen D.: „On Holy Wars and a Plea for Peace.” *Computer*, October 1981.
- COLW85a Colwell R. i in.: „Computers, Complexity and Controversy”. *Computer*, September 1985.
- COLW85b Colwell R. i in.: „More Controversy About ‘Computers, Complexity and Controversy’”. *Computer*, December 1985.
- COME92 Come R., Watson G. (red.): „Memory Catches Up”. *IEEE Spectrum*, October 1992.
- CONN79 Connors W., Florkowski J., Patton S.: „The IBM 3033: An Inside Look”. *Datamation*, May 1979.
- CONW77 Conway J.: „Approach to Unified Bus Architecture Sidestepping Inherent Drawbacks”. *Computer Design*, January 1977.
- COOK82 Cook R., Dande N.: „An experiment to improve operand addressing.” *Proceedings, Symposium on Architecture Support for Programming Languages and Operating Systems*, March 1982.
- COON81 Coonen J.: „Underflow and Denormalized Numbers”. *IEEE Computer*, March 1981.
- COUT86 Coutant D., Hammond C., Kelley J.: „Compilers for the New Generation of Hewlett-Packard Computers”. *Proceedings, COMPCON Spring '86, March* 1986.
- CRA79 Cragon H.: „An evaluation of Code Space Requirements and Performance of Various Architectures”. *Computer Architecture News*, February 1979.
- CRA92 Cragon H.: *Branch Strategy Taxonomy and Performance Models*. Los Alamitos, IEEE Computer Society Press, 1992.
- CRAW90 Crawford J.: „The i486 CPU: Executing Instructions in one Clock Cycle.” *IEEE Micro*, February 1990.

- DATT93 Dattatreya G.: „A Systematic Approach to Teaching Binary Arithmetic in a First Course”. *IEEE Transactions on Education*, February 1993.
- DAVI87 Davidson J., Vaughan R.: „The Effect of Instruction Set Complexity on Program Size and Memory Performance”. *Proceedings, Second International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1987.
- DAWS86 Dawson W., Dobinson R.: „A framework for computer design”. *IEEE Spectrum*, October 1986.
- DAWS87 Dawson W., Dobinson R.: „Buses and Bus Standards”. *Computer Standards and Interfaces*, June 1987.
- DEDE94 Dedek J.: *Basics of SCSI*. Menlo Park, Ancot Corp., 1994.
- DEIT90 Deitel H.: *An introduction to Operating Systems*. Reading, Addison-Wesley, 1990.
- DENN68 Denning P.: „The Working Set Model for Program Behavior.” *Communications of the ACM*, May 1968.
- DEWA90 Dewar R., Smosna M.: *Microprocessors: A Programmer's View*. New York, McGraw-Hill, 1990.
- DIEF94a Diefendorf K., Oehler R., Hochsprung R.: „Evolution of the PowerPC Architecture”. *IEEE Micro*, April 1994.
- DIEF94b Diefendorf K., Silha E.: „The PowerPC User Instruction Set Architecture”. *IEEE Micro*, April 1994.
- DIJK63 Dijkstra E.: „Making an ALGOL Translator for the XI”. *Annual Review of Automatic Programming*, t. 4, Pergamon, 1963.
- DORA82 Doran R.: „The Amdahl 470V/8 and the IBM 3033: A Comparison of Processor Designs”. *Computer*, April 1982.
- DUBB78 Dubbey J.: *The Mathematical Work of Charles Babbage*. Cambridge, Cambridge University Press, 1978.
- DUBE91 Dubey P., Flynn M.: „Branch Strategies: Modeling and Optimization”. *IEEE Transactions on Computers*, October 1991.
- DUGA83 Dugan R.: „System/370 Extended Architecture: a Program View of the Channel Subsystem”. *Proceedings, Tenth Annual International Symposium on Computer Architecture*, June 1983.
- ECKE90 Eckert R.: „Communication Between Computers and Peripheral Devices - an Analogy”. *ACM SIGSCE Bulletin*, September 1990.
- EDEN83 Eden R., Livingston A., Welch B.: „Integrated Circuits: The Case for Gallium Arsenide”. *IEEE Spectrum*, December 1983.
- ELAS84 El-Asfoury S., Johnson O., King W.: *Computer Organization and Programming VAX-II*. Reading, Addison-Wesley, 1984.
- ELAY79 El-Ayat K.: „The Intel 8089: An Integrated I/O Processor”. *Computer*, June 1979.
- ELAY85 El-Ayat K., Agarwal R.: „The Intel 80386 - Architecture and Implementation”. *IEEE Micro*, December 1985.
- ENSL77 Enslow P.: „Multiprocessor Organization - A Survey”. *ACM Computing Surveys*, March 1977.
- ESPO92 Esponda M., Rojas R.: „A Graphical Representation of RISC Processors”. *Computer Architecture News*, September 1992.
- FELD94 Feldman J., Retter C.: *Computer Architecture*. New York, McGraw-Hill, 1994.
- FITZ81 Fitzpatrick D. i in.: „A RISCy Approach to VLSI.” *VLSI Design*, 4. kw. 1981. Przekład w *Computer Architecture News*, March 1982.

- FLOY72 Floyd M.: „Some computer organizations and their effectiveness”. *IEEE Transactions on Computers*, September 1972.
- FLOY90 Floyd T.: *Digital fundamentals*. New York, Merrill/Macmillan, 1990.
- FLYN71 Flynn M., Rosin R.: „Microprogramming: An Introduction and a View-Point.” *IEEE Transactions on Computers*, July 1971.
- FLYN72 Flynn M.: „Some computer organizations and their effectiveness”. *IEEE Transactions on Computers*, September 1972.
- FLYN85 Flynn M., Johnson J., Wakefield S.: „On instruction sets and their formats.” *IEEE Transactions on Computers*, March 1985.
- FLYN87 Flynn M., Mitchell C., Mulder J.: „And Now a Case for More Complex Instruction Sets”. *Computer*, September 1987.
- FOSS85a Fossum T., McElroy J., English B.: „New VAX Squeezes Mainframe Power Into Mini Package”. *Computer Design*, March 1985.
- FOSS85b Fossum T., Grundmann W., Błaha V.: „Floating-Point Processor for the VAX 8600”. *Proceedings, COMPCON Spring'85*, 1985.
- FOSS85c Fossum T., McElroy J., English W.: „An Overview of the VAX 8600 System”. *Digital Technical Journal*, August 1985.
- FOST67 Foster J.: *List processing*. New York, American Elsevier, 1967.
- FOX87 Fox G., Messina G.: „Advanced Computer Architectures”. *Scientific American*, October 1987.
- FRAI83 Frailey D.: „Word Length of a Computer Architecture: Definitions and Applications”. *Computer Architecture News*, June 1983.
- FREE88 Freese R.: „Optical disks become erasable”. *IEEE Spectrum*, February 1988.
- FUJI84 Fujitani L.: „Laser Optical Disk: the Corning Revolution in On-Line Storage”. *Communications of the ACM*, June 1984.
- FURH87 Furht B., Milutinovic V.: „A Survey of Microprocessor Architectures for Memory Management”. *Computer*, March 1987.
- GARR94 Garrett B.: „RDRAMs: A New Speed Paradigm”. *Proceedings, COMPCON'94*, March 1994.
- GEHR88 Gehringer E., Abullarade J., Gulyn M.: „A Survey of Commercial Parallel Processors”. *Computer Architecture News*, September 1988.
- GIBS88 Gibson S.: „Hardware Roundup: Large and Medium-Scale Systems”. *Computerworld*, September 19, 1988.
- GIFF87 Gifford D., Spector A.: „Case Study: IBM's System/360-370 Architecture”. *Communications of the ACM*, April 1987.
- GILL94 Gillig J.: „Endian-Neutral Software”. *Dr Dobbs's Journal*, October 1992.
- GJES92 Gjessing S. i in.: „A RAM Link for High Speed”. *IEEE Spectrum*, October 1992.
- GLAS91 Glass B.: „The MIPS R4000”. *Byte*, December 1991.
- GOLD91 Goldberg D.: „What Every Computer Scientist Should Know About Floating-Point Arithmetic”. *ACM Computing Surveys*, March 1991.
- GOOR89 Goor A.: *Computer Architecture and Design*. Reading, Addison-Wesley, 1989.
- GOSS89 Goss R.: „Motorola's 88000: Integration, Performance and Applications”. *Proceedings, COMPCOM Spring'89*, March 1989.
- GUST84 Gustavson D.: „Computer Buses - A Tutorial”. *IEEE Micro*, August 1984.
- GUTE88 Guterl F.: „Compact Disc”. *IEEE Spectrum*, November 1988.
- HAAV71 Haavind R.: „The Many Faces of Microprogramming”. *Computer Decisions*, September 1971.

- HABE85 Haber L.: „Multiprocessor Technology Means More Muscle, Less Fat”. *Mini-micro Systems*, June 1985.
- HAND93 Handy J.: *The Cache Memory Book*. San Diego, Academic Press, 1993.
- HAYE88 Hayes J.: *Computer Architecture and Organization*. Wyd. 2., New York, McGraw-Hill, 1988.
- HEAT84 Heath J.: „Re-evaluation of RISC I”. *Computer Architecture News*, March 1984.
- HEDG92 Hedge A.: „Detect/Correct Errors to Improve Data Reliability”. *Electronic Design*, June 11, 1992.
- HENN82 Hennessy J. i in.: „Hardware/Software Tradeoffs for Increased Performance”. *Proceedings, Symposium on Architectural Support for Programming Languages and Operating Systems*, March 1982.
- HENN84 Hennessy J.: „VLSI Processor Architecture”. *IEEE Transactions on Computers*, December 1984.
- HENN90 Hennessy J., Patterson D.: *Computer Architecture: a Quantitative Approach*. San Mateo, Morgan Kaufmann, 1990.
- HENN91 Hennessy J., Jouppi N.: „Computer Technology and Architecture: an Evolving Interaction.” *Computer*, September 1991.
- HEYW83 Heywood S.: „The 8086 - An Architecture for the Future”. *Byte*, June 1983.
- HIDA90 Hidaka H. i in.: „The Cache DRAM Architecture: A DRAM with an On-Chip Cache Memory”. *IEEE Micro*, April 1990.
- HIGB83 Higbie L.: „A Vector Processing Tutorial”. *Datamation*, August 1983.
- HIGB90 Higbie L.: „Quick and Easy Cache Performance Analysis”. *Computer Architecture News*, June 1990.
- HIGM67 Higman B.: *A Comparative Study of Programming Languages*. New York, American Elsevier, 1967.
- HILL64 Hill R.: „Stored Logic Programming and Applications”. *Datamation*, February 1964.
- HILL89 Hill M.: „Evaluating Associativity in CPU Caches”. *IEEE Transactions on Computers*, December 1989.
- HITC85 Hitchcock C., Brinkley H.: „Analyzing Multiple Register Sets”. *The 12th Annual International Symposium on Computer Architecture*, June 17-19, 1985.
- HOF95 Hof R.: „Intel: Far Beyond the Pentium”. *Business Week*, February 20, 1995.
- HOPK87 Hopkins M.: „A Perspective on the 801/Reduced Instruction Set Computer”. *IBM Systems Journal*, t. 26., nr 1, 1987.
- HUCK83 Huck T.: *Comparative Analysis of Computer Architectures*. Stanford University Technical Report No. 83-243, May 1983.
- HUGU91 Huguet M., Lang T.: „Architectural Support for Reduced Register Saving/Restoring in Single-Window Register Files”. *ACM Transactions on Computer Systems*, February 1991.
- HWAN93 Hwang K.: *Advanced Computer Architecture*. New York, McGraw-Hill, 1993.
- IBM94 International Business Machines, Inc.: *The PowerPC Architecture: A Specification for a New Family of RISC Processors*. San Francisco, Morgan Kaufmann, 1994.
- IEEE84 Institute of Electrical and Electronics Engineers: *IEEE Standard Graphic Symbols for Logic Functions*. ANSI/IEEE Std 91-1984, 1984.
- IEEE85 Institute of Electrical and Electronics Engineers. *IEEE Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE Std 754-1985, 1985.
- INTE81 Intel Corp.: „8085AH/8085AH-2/8085AH-1 3-bit HMOS Microprocessor”. 1981.
- INTE94a Intel Corp.: Pentium Family User’s Manual, Vol. 1: Data Book. Santa Clara, 1994.

- INTE94b Intel Corp.: *Pentium Family User's Manual, Vol. 3: Architecture and Programming Manual*. Santa Clara, 1994.
- JAME90 James D.: „Multiplexed Buses: The Endian Wars Continue”. *IEEE Micro*, June 1990.
- JEND83 Jendro J.: „Extending the Megabus”. *Mini-micro Systems*, September 1983.
- JOHN71 Johnson A.: „The Microdiagnostics for the IBM System/360 Model 30”. *IEEE Transactions on Computers*, July 1971.
- JOHN84 Johnson M., Kassel S.: *The Multibus Design Guidebook*. New York, McGraw-Hill, 1984.
- JOHN91 Johnson M.: *Superscalar microprocessor design*. Englewood Cliffs, Prentice-Hall, 1991.
- JONE91 Jones S.: „A Futurebus Interface from Off-the-Shelf Parts”. *IEEE Micro*, February 1991.
- JOUP88 Jouppi N.: „Superscalar Versus Superpipelined Machines”. *Computer Architecture News*, June 1988.
- JOUP89a Jouppi N., Wall D.: „Available instruction-level parallelism for superscalar and superpipelined machines”. *Proceedings, Third International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1989.
- JOUP89b Jouppi N.: „The Nonuniform Distribution of Instruction-Level and Machine-Level Parallelism and Its Effect on Performance”. *IEEE Transactions on Computers*, December 1989.
- KAEL91 Kaeli D., Emma P.: „Branch History Table Prediction of Moving Target Branches Due to Subroutine Returns”. *Proceedings, 18th Annual International Symposium on Computer Architecture*, May 1991.
- KANE92 Kane G., Heinrich J.: *MIPS RISC Architecture*. Englewood Cliffs, Prentice Hall, 1992.
- KARP87 Karp A.: „Programming for parallelism”. *Computer*, 1987.
- KATE83 Katevenis M.: *Reduced Instruction Set Computer Architectures for VLSI*. Rozprawa doktorska, Computer Science Department, University of California at Berkeley, October 1983, przedrukowane przez MIT Press, Cambridge, 1985.
- KNUT71 Knuth D.: „An Empirical Study of FORTRAN Programs”. *Software Practice and Experience*, t. 1, 1971.
- KNUT81 Knuth D.: *The Art of Computer Programming. Vol. 2: Seminumerical Algorithms*. Wyd. 2., Reading, Addison-Wesley, 1981.
- KOES78 Koestler A.: *Janus*. New York, Random House, 1978.
- KOP81 Kop H.: *Hard Disk Controller Design Using the Intel 8089*. Intel Application Note AP-122, 1981.
- KORN93 Koren L.: *Computer Arithmetic Algorithms*. Englewood Cliffs, Prentice Hall, 1993.
- KOZD80 Kozdrowicki E., Theis D.: „Second Generation of Vector Supercomputers”. *Computer*, November 1980.
- KRAF81 Kraff G., Toy W.: *Microprogrammed Control and Reliable Design of Small Computers*. Englewood Cliffs, Prentice-Hall, 1981.
- KRUT88 Krutz R.: *Interfacing Techniques in Digital Design with Emphasis on Microprocessors*. New York, Wiley, 1988.
- KRYD86 Kryder M. (red.): „Special Section on Magnetic Information Storage Retrieval”. *Proceedings of the IEEE*, November 1986.
- KRYD87 Kryder M.: „Data Storage Technologies for Advanced Computing”. *Scientific American*, October 1987.

- KUCK72 Kuck D., Muraoka Y., Chen S.: „On the Number of Operations Simultaneously Executable in Fortran-Like Programs and Their Resulting Speedup”. *IEEE Transactions on Computers*, December 1972.
- KUGA91 Kuga M., Murakami T., Tomita S.: „DSNS (Dynamically-hazard Resolved, Statically-code-scheduled, Nonuniform Superscalar): Yet Another Superscalar Processor Architecture”. *Computer Architecture News*, June 1991.
- LEE91 Lee R., Kwok A., Briggs F.: „The Floating-Point Performance of a Superscalar SPARC Processor”. *Proceedings, Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.
- LEVI82 Levine R.: „Supercomputers”. *Scientific American*, January 1982.
- LEVY78 Levy J.: „Buses, the Skeleton of Computer Structures”. W [BELL78a].
- LEVY89 Levy H., Eckhouse R.: *Computer Programming and Architecture: the VAX-11*. Bedford, Digital Press, 1989.
- LILJ88 Lilja D.: „Reducing the Branch Penalty in the Pipelined Processors”. *Computer*, July 1988.
- LILJ93 Lilja D.: „Cache Coherence in Large-Scale Shared-Memory Multiprocessors: Issues and Comparisons”. *ACM Computing Surveys*, September 1993.
- LUHN84 Luhn R.: „The Ups and Downs of Bubble Memory”. *Computerworld*, December 3, 1984.
- LUND77 Lunde A.: „Empirical Evaluation of Some Features of Instruction Set Processor Architectures”. *Communications of the ACM*, March 1977.
- LYNC93 Lynch M.: *Microprogrammed State Machine Design*. Boca Raton, 1993.
- MACG84 MacGregor D., Mothersole D., Moyer B.: „The Motorola MC68020”. *IEEE Micro*, August 1984.
- MALL75 Mallach E.: „Emulation Architecture”. *Computer*, August 1975.
- MALL79 Mallach E.: „The Evolution of an Architecture”. *Datamation*, April 1979.
- MALL83 Mallach E., Sondak N.: *Advances in Microprogramming*. Dedham, Artech House, 1983.
- MANO93 Mano M.: *Computer System Architecture*. Englewood Cliffs, Prentice Hall, 1993.
- MARC90 Marchant A.: *Optical Recording*. Reading, Addison-Wesley, 1990.
- MASH94 Mashey J.: „CISC vs. RISC (or what is RISC really)”. *USENET comp.arch newsgroup*, aniele 22850, January 1994.
- MASS94 Massiglia P. (red.): *The RAIDbook: A Sourcebook for Disk Array Technology*. St. Peter, The Raid Advisory Board, 1994.
- MAY94 May C. i in. (red.): *The PowerPC Architecture*. San Francisco, Morgan Kaufmann, 1994.
- MAYB84 Mayberry W., Efland G.: „Cache Boosts Multiprocessor Performance”. *Computer Design*, November 1984.
- MCEL85 McEliece R.: „The Reliability of Computer Memories”. *Scientific American*, January 1985.
- MCGE90 McGeady S.: „The i960CA Superscalar Implementation of the 80960 Architecture”. *Proceedings, COMPCON Spring '90*, March 1990.
- MCGE91 McGeady i in.: „Performance Enhancements in the Superscalar i960MM Embedded Microprocessor”. *Proceedings, COMPCON Spring '91*, February 1991.
- MEIK86 Meiklejohn W.: „Magneto-optics: A Thermomagnetic Recording Technology”. *Proceedings of the IEEE*, November 1986.
- MEIN87 Meindl J.: „Chips for Advanced Computing”. *Scientific American*, October 1987.

- MELE89 Melear C.: „The Design of the 88000 RISC Family”. *IEEE Micro*, April 1989.
- MILE92 Milenkovic M.: *Operating Systems: Concepts and Design*. New York, McGraw-Hill, 1992.
- MILU86 Milutinovic V.: „GaAs Microprocessor Technology”. *Computer*, October 1986.
- MIRA92 Mirapuri S., Woodacre M., Vasseghi N.: „The MIPS R4000 Processor”. *IEEE Micro*, April 1992.
- MOKH84 Mokhoff N.: „Magnetic Bubble Memories Making a Comeback”. *Computer Design*, November 1984.
- MOLD93 Moldovan D.: *Paralell Processing: From Applications to Systems*, San Mateo, Morgan Kaufmann, 1993.
- MORG92 Morgan D.: *Numerical Methods*. San Mateo. M&T Books, 1992.
- MORS94 Morse H.: *Practical Parallel Computing*. Boston, AP Proffesional, 1994.
- MORS78 Morse S., Pohlman W., Ravenel B.: „The Intel 8086 Microprocessor: A 16-bit Evolution of the 8080”. *Computer*, June 1978.
- MYER78 Myers G.: „The Evaluation of Expressions in a Storage-to-Storage Architecture”. *Computer Architecture News*, June 1978.
- NBS79 National Bureau of Standards: *I/O Channel Interface*. FIPS PUB 60-1, August 27, 1979.
- NCR90 NCR Corp.: *SCSI: Understanding the Smali Computer System Interface*. Englewood Cliffs, Prentice Hall, 1990.
- NOVI93 Novitsky J., Azimi M., Ghaznavi R.: „Optimizing Systems Performance Based on Pentium Processors”. *Proceedings COMPCON'92*, February 1993.
- NUTT92 Nutt G.: *Centralized and Distributed Operating Systems*. Englewood Cliffs, Prentice Hall, 1992.
- OBER88 Obermeier K.: „Side by Side”. *Byte*, November 1988.
- OMON94 Omondi A.: *Computer Arithmetic Systems: Algorithms, Architecture and Implementations*, Englewood Cliffs, Prentice Hall, 1994.
- PADE81 Padegs A.: „System/360 and Beyond”. *IBM Journal of Research and Development*, September 1981.
- PADE88 Padegs A. i in.: „The IBM System/370 Vector Architecture: Design Considerations”. *IEEE Transactions on Communications*, May 1988.
- PARK89 Parker A., Hamblen J.: *An introduction to Microprogramming With Exercises Designed for the Texas Instruments SN74ACT8800 Software Development Board*. Dallas, Texas Instruments, 1989.
- PATT82a Patterson D., Sequin C.: „A VLSI RISC”. *Computer*, September 1982.
- PATT82b Patterson D., Piepho R.: „Assessing RISCs in High-Level Language Support”. *IEEE Micro*, November 1982.
- PATT84 Patterson D.: „RISC Watch”. *Computer Architecture News*, March 1984.
- PATT85a Patterson D.: „Reduced Instruction Set Computers”. *Communications of the ACM*, January 1985.
- PATT85b Patterson D., Hennessy J.: „Response to 'Computers, Complexity and Controversy' ”. *Computer*, November 1985.
- PATT88 Patterson D., Gibson G., Katz R.: „A Case for Redundant Arrays of Inexpensive Disks (RAID)”. *Proceedings, ACM SIGMOND Conference of Management of Data*, June 1988.
- PATT94 Patterson D., Hennessy J.: *Computer Organization and Design: the Hardware/Software Interface*. San Mateo, Morgan Kaufmann. 1994.

- PAYN80 Payne M., Bhandarkar D.: „VAX Floating-Point: A Solid Foundation for Numerical Computation”. *Computer Architecture News*, June 1980.
- PEUT79 Peuto B.: „Architecture of a New Microprocessor”. *Computer*, February 1979.
- PHIL85 Philips D.: „The Z80000 Microprocessor”. *IEEE Micro*, December 1985.
- PIER84 Pierce R.: „Diskless Computers Emerge with Proper Mix of Firmware, Processor and Bubble Memory”. *Electronic Design*, October 13, 1984.
- POHL81 Pohl L., Shaw A.: *The Nature of Computation: An Introduction to Computer Science*. Rockville, Computer Science Press, 1981.
- POPE91 Popescu V. i in.: „The Metaflow Architecture”. *IEEE Micro*, June 1991.
- POTT94 Potter T. i in.: „Resolution of Data and Control-Flow Dependencies in the PowerPC 601”. *IEEE Micro*, October 1994.
- POUN93 Pountain D.: „Pentium: More RISC than CISC”. *Byte*, September 1993.
- PRIN91 Prince B.: *Semiconductor Memories*. New York. Wiley, 1991.
- PRZY88 Przybylski S., Horowitz M., Hennessy J.: „Performance trade-offs in Cache Design”. *Proceedings, Fifteenth Annual International Symposium on Computer Architecture*, June 1988.
- PRZY90 Przybylski S.: „The Performance Impact of Block Size and Fetch Strategies”. *Proceedings, 17th Annual International Symposium on Computer Architecture*, May 1990.
- PRZY93a Przybylski S.: „New DRAMs Improve Bandwidth”. *Microprocessor Report*, February 15, 1993.
- PRZY93b Przybylski S.: „DRAMs for New Memory Systems”. *Microprocessor Report*, March 8, 1993.
- PRZY94 Przybylski S.: *New DRAM Technologies*. Sebastopol. MicroDesign Resources, 1994.
- RADI83 Radin G.: „The 801 Minicomputer”. *IBM Journal of Research and Development*, May 1983.
- RAGA83 Ragan-Kelley R., Clark R.: „Applying RISC Theory to a Large Computer”. *Computer Design*, November 1983.
- RAUS80 Rauscher T., Adams P.: „Microprogramming: A Tutorial and Survey of Recent Developments”. *IEEE Transactions on Computers*, January 1980.
- REIS91 Reis R.: *Digital Electronics Through Project Analysis*. New York, Merrill/Macmillan, 1991.
- RIGA84 Riganati J., Schneck P.: „Supercomputing”. *Computer*, October 1984.
- ROSC94 Rosch W.: *The Winn L Rosch Hardware Bible*. Indianapolis, Sams, 1994.
- RUSS78 Russell R.: „The CRAY-1 Computer System”. *Communications of the ACM*, January 1978.
- RYAN93 Ryan B.: „RISC drives PowerPC”. *Byte*, August 1993.
- SATY81 Satyanarayanan M., Bhandarkar D.: „Design Trade-offs in VAX-11 Translation Buffer Organization”. *Computer*, December 1981.
- SCHL89 Schleicher D., Taylor R.: „Systems Overview of the Application System/400”. *IBM Systems Journal*, Nr 3, 1989.
- SCHM94 Schmit M.: „Optimizing Pentium Code”. *Dr Dobb's Journal*, January 1994.
- SEBE76 Sebern M.: „A Minicomputer-compatible Microcomputer System: The DEC LSI-11”. *Proceedings of the IEEE*, June 1976.
- SEGE91 Segee B., Field J.: *Microprogramming and Computer Architecture*. New York, Wiley, 1991.
- SERL86 Serlin O.: „MIPS, Dhrystones, and Other Tales”. *Datamation*, June 1, 1986.

- SHAN38 Shannon C.: „Symbolic Analysis of Relay and Switching Circuits”. *AIEE Transactions*, Vol. 57., 1938.
- SHAN94a Shanley T., Anderson D.: *PCI Systems Architecture*. Richardson, Mindshare Press, 1994.
- SHAN94b Shanley T.: *PowerPC System Architecture*. Richardson, Mindshare Press, 1994.
- SHER84 Sherburne R.: *Processor Design Tradeoffs in VLSI*. Tezy doktorskie, Report No. UCB/CSD 84/173, University of California at Berkeley, April 1984.
- SIER90 Sierra H.: *An Introduction to Direct Access Storage Devices*. Boston, Academic Press, 1990.
- SIEW82 Siewiorek B., Bell C., Newell A.: *Computer Structures: Principles and Examples*. New York, McGraw-Hill, 1982.
- SILB91 Silberschatz A., Peterson J., Galvin P.: *Podstawy systemów operacyjnych*. Warszawa, WNT 1993.
- SILB94* Silberschatz A., Galvin P.: *Operating System Concepts*. Reading, Addison-Wesley, 1994.
- SIMO69 Simon H.: *The Sciences of Artificial*. Cambridge, MIT Press, 1969.
- SIMP87 Simpson R., Hester P.: „The IBM RT PC ROMP Processor and Memory Management Unit”. *IBM Systems Journal*, t. 26., nr 4., 1987.
- SING94 Singhal M., Shivaratri N.: *Advanced Concepts in Operating Systems*. New York, McGraw-Hill, 1994.
- SMIT82 Smith A.: „Cache memories”. *ACM Computing Surveys*, September 1982.
- SMIT87a Smith A.: „Line (Block) Size Choice for CPU Cache Memories”. *IEEE Transactions on Communications*, September 1987.
- SMIT87b Smith A.: „Cache Memory Design: An Evolving Art”. *IEEE Spectrum*, December 1987.
- SMIT89 Smith M., Johnson M., Horowitz M.: „Limits on Multiple Instruction Issue”. *Proceedings, Third International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1989.
- SOHI90 Sohi G.: „Instruction Issue Logic for High-Performance Interruptable, Multiple Functional Unit, Pipelined Computers”. *IEEE Transactions on Computers*, March 1990.
- SOLA94 Solari E., Willse G.: *PCI Hardware and Software: Architecture and Design*. San Diego, Annabooks, 1994.
- STAL94 Stallings W.: *Data and Computer Communications*. Wyd. 4. Englewood Cliffs, Prentice Hall, 1994.
- STAL95 Stallings W.: *Operating Systems*. Wyd. 2. Englewood Cliffs, Prentice Hall, 1995.
- STEN90 Stenstrom P.: „A Survey of Cache Coherence Schemes of Multiprocessors”. *Computer*, June 1990.
- STEV64 Stevens W.: „The Structure of System/360, Part II: System Implementation”. *IBM Systems Journal*, Vol. 3., Nr 2., 1964. Przedruk w [BELL71A] i [SIEW82].
- STON80 Stone H. (red.): *Introduction to Computer Architecture*. Chicago, SRA, 1980.
- STON93 Stone H.: *High-Performance Computer Architecture*. Reading, Addison-Wesley, 1993.
- STRE78a Strecker W.: „Cache Memories for PDP-11 Family Computers”, w [BELL78a].

* Nakładem WNT ukazało się w 2000 roku tłumaczenie kolejnego piątego wydania amerykańskiego (*przyp.red.*).

- STRE78b Strecker W.: „VAX-11/780: A Virtual Address Extension to the DEC PDP-11 Family”. *Proceedings, National Computer Conference*, 1978.
- STRE83 Strecker W.: „Transient Behavior of Cache Memories”. *ACM Transactions on Computer Systems*, November 1983.
- STRI79 Stritter E., Gunter T.: „A Microprocessor Architecture for a Changing World: The Motorola 68000”. *Computer*, February 1979.
- SWAR90 Swartzlander E. (red.): *Computer Arithmetic, Vol. I i H. Los Alamitos*, IEEE Computer Society Press, 1990.
- TABA87 Tabak D.: *RISC Architecture*. New York, Wiley, 1987.
- TABA91 Tabak D.: *Advanced Microprocessors*. New York, McGraw-Hill, 1991.
- TAMI83 Tamir Y., Sequin C.: „Strategies for Managing the Register File in RISC”. *IEEE Transactions on Computers*, November 1983.
- TANE78 Tanenbaum A.: „Implications of Structured Programming for Machine Architecture”. *Communications of the ACM*, March 1978.
- TANE90 Tanenbaum A.: *Structured Computer Organization*. Englewood Cliffs, Prentice-Hall, 1990.
- TANE92 Tanenbaum A.: *Modern Operating Systems*. Englewood Cliffs, Prentice-Hall, 1992.
- TEEN93 Teener M.: „New Technology in the IEEE P1394 Serial Bus - Making it Fast, Cheap and Easy to Use”. *Proceedings, Hot Interconnects Symposium'93*, August 1993.
- TEJA85 Teja E.: *The Designer Guide to Disk Drives*. Reston, Reston Publishing, 1985.
- THOM94 Thompson T., Ryan B.: „PowerPC 620 Soars”. *Byte*, November 1994.
- TI90 Texas Instruments Inc.: *SN74ACT880 Family Data Manual*. SCSS006C, 1990.
- TJAD70 Tjaden G., Flynn M.: „Detection and Parallel Execution of Independent Instructions”. *IEEE Transactions on Computers*, October 1970.
- TOMA93 Tomasevic M., Milutinowic V.: *The Cache Coherence Problem Shared-Memory Multiprocessors: Hardware Solutions*. Los Alamitos, IEEE Computer Society Press, 1993.
- TOON81 Toong H., Gupta A.: „An Architectural Comparison of Contemporary 16-bit Microprocessors”. *IEEE Micro*, May 1981.
- TUCK67 Tucker S.: „Microprogram Control for System/360”. *IBM Systems Journal*, nr 4., 1967. Również w [MALL83].
- TUCK87 Tucker S.: „The IBM 3090 System Design with Emphasis on the Vector Facility”. *Proceedings, COMPCON Spring '87*, February 1987.
- TURN86 Turner R.: *Operating Systems: Design and Implementation*. New York, Macmillan, 1986.
- UNGE89 Unger S.: *The Essence Of Logic Circuits*. Englewood Cliffs, Prentice-Hall, 1989.
- VERN88 Vernon M., Manber U.: „Distributed Round-Robin and First-Come-First-Serve Protocols and Their Application to Multiprocessor Bus Arbitration”. *Proceedings, Fifteenth Annual International Symposium on Computer Architecture*, June 1988.
- VOEL88 Voelker J.: „The PDP-8”. *IEEE Spectrum*, November 1988.
- VOGL94 Vogley B.: „800 Megabyte Per Second Systems Via Use of Synchronous DRAM”. *Proceedings, COMPCON'94*, March 1994.
- VONN45 Von Neumann J.: *First Draft of a Report on the EDVAC*. Moore School, University of Pennsylvania, 1945.
- VRAN80 Vranesic Z., Thurber K.: „Teaching Computer Structures”. *Computer*, June 1980.
- WAKE90 Wakerly J.: *Digital Design Principles and Practices*. Englewood Cliffs, Prentice-Hall, 1990.
- WALL85 Wallich P.: „Toward Simpler, Faster Computers”. *IEEE Spectrum*, August 1985.

- WALL90 Wallis P.: *Improving Floating-Point Programming*. New York, Wiley, 1992.
- WALL91 Wall D.: „Limits of Instruction-Level Parallelism”. *Proceedings, Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.
- WARD90 Ward S., Halstead R.: *Computation Structures*. Cambridge, MIT Press, 1990.
- WATE86 Waters F. (red.): *IBM RT Personal Computer Technology*. IBM Corp. SA23-1057, 1986.
- WAYN92 Wayner P.: „Processor pipelines”. *Byte*, January 1992.
- WEBE67 Weber H.: „A Microprogrammed Implementation of EULER on IBM System/360 Model 30”. *Communications of the ACM*, September 1967.
- WEIN75 Weĳnberg G.: *An Introduction to General Systems Thinking*. New York, Wiley, 1975.
- WEIS84 Weiss S., Smith J.: „Instruction Issue Logic in Pipelined Supercomputers”. *IEEE Transactions on Computers*, November 1984.
- WEIS93 Weiss S.: „Optimizing a Superscalar Machine to Run Vector Code”. *IEEE Parallel & Distributed Technology*, May 1993.
- WEIS94 Weiss S., Smith J.: *POWER and PowerPC*. San Francisco, Morgan Kaufmann, 1994.
- WEIZ94 Weizer N., in.: *The Arthur D. Little Forecast on Information Technology and Productivity*. New York, Wiley, 1994.
- WIEC82 Wiecek L.: „A Case Study of VAX-11 Instruction Set Usage for Compiler Execution”. *Proceedings, Symposium on Architectural Support for Programming Languages and Operating Systems*, March 1982.
- WILK51 Wilkes M.: „The Best Way to Design an Automatic Calculating Machine”. *Proceedings, Manchester University Computer Inaugural Conference*, July 1951.
- WILK53 Wilkes M., Stringer J.: „Microprogramming and the Design of the Control Circuits in an Electronic Digital Computer”. *Proceedings of the Cambridge Philosophical Society*, April 1953. Również w [SIEW82].
- WILL90 Williams F., Steven G.: „Address and Data Register Separation on the M68000 Family”. *Computer Architecture News*, June 1990.
- WILS84 Wilson K.: „Science, Industry and the New Japanese Challenge”. *Proceedings of the IEEE*, January 1984.
- WIND85 Windsor W.: „IEEE Floating-Point Chips Implement DSP Architectures”. *Computer Design*, January 1985.
- WONG80 Wong C.: „Minimizing Expected Head Movement in One-Dimensional and Two-Dimensional Mass Storage Systems”. *Computing Surveys*, June 1980.
- YEN85 Yen W., Yen D., Fu K.: „Data Coherence in a Multicache System”. *IEEE Transactions on Computers*, January 1985.
- ZECH88 Zech R.: „Systems, Applications and Implications of Optical Storage”. *Proceedings, COMPCON Spring '88*, March 1988.

Skorowidz

A

AC 37
adres bazowy 294
- efektywny 313
- fizyczny 294
- liniowy 306
- logiczny 293
- wirtualny 306
adresowanie bezpośrednie 424
- natychmiastowe 423
- pośrednie 424
- rejestrowe 425
- -, pośrednie 426
- stosowe 429
- , tryby 421–422
- wielopoziomowe 425
- względne 426
- z przesunięciem 426
- z rejestrem podstawowym 427
akumulator 37, 374
algebra Boole'a 694
algorytm „zapisz raz” 170
- Bootha 335, 337
- FIFO 162
- LFU 162
- LRU 162
- zapisu 162
algorytmy zastępowania 162
ALU 321
analiza przepływu danych 57
antyżależność 555
arbitraż 96
architektura harwardzka 526

architektura komputera 19
- międzypiętrowa 94
- POWER 62
- PowerPC 62
- von Neumanna 70
arytmetyka komputera 29
- -, nieskończoność 357
- przedziałów 356
ASCII 221–222
assembler 402

B

bezpośredni dostęp do pamięci 87, 228,
243–245
bit parzystości 146–147
bity zabezpieczenia 354
blok kontrolny procesu 285
bloki 128
błąd strony 296
bramka 696
bufor sterowania 608
- TLB 299
- translacji adresów tablic stron 299
buforowanie danych 225
buforowy rejestr rozkazów 37

C

CD-ROM 209
-, pojemność dysku 210
cechy pamięci RAID 198
chybienie odczytu 666
- zapisu 668
cicha NaN 357, 358

CPU 24

cykl adresowania pośredniego 583

- magistrali 96
 - pobierania 73, 581
 - pośredni 461
 - przerwania 80, 583
 - rozkazu 37, 73, 461, 585-586
 - -, przepływ danych 463
 - -, stany 76-77
 - wykonywania 37, 73, 584
- czas cyklu pamięci 42, 130
- dostępu 129, 197
 - przeszukiwania 197

D

- dane logiczne 378
- dekoder 713
- demultiplexer 714
- długość rozkazu 436-439
- DMA 87, 228, 243-245
- , wykradanie cyklu 244
- dostęp bezpośredni 129
- sekwencyjny 129
 - skojarzeniowy 129
 - swobodny 129
- dynamiczna pamięć RAM 137
- dysk niewymienny 194
- WORM 212
- dyski magnetyczne 191
- -, czas dostępu 197
 - -, czas przeszukiwania 197
 - -, gęstość zapisu 192
 - -, opóźnienie obrotowe 197
 - typu Winchester 195
 - -, własności 194
 - z nieruchomą głowicą 194
 - z ruchomą głowicą 194
 - optyczne, rodzaje 208
 - wymienne 195
- dyskietka 195
- dyspozytor 284

E

- EDVAC 33
- efektywność dostępu 188
- elementy rozkazu maszynowego 362
- ENIAC 32

F

- firmware* 605
- flagi 456, 457
- format IEEE 754 347-348, 357
- rozkazu 370
- funkcja Boole'a 700
- -, mapy Karnaugh'a 703
 - -, postać iloczynu sum 701
 - -, - sumy iloczynów 700
 - -, realizacja NAND i NOR 710
 - -, upraszczanie algebraiczne 702
- funkcje komputera 22

G

- gęstość liczb zmiennopozycyjnych 345
- graf, kolorowanie 510
- grafy Kiviat'a 541-542
- granica stosu 413

H

- hierarchia pamięci 131

I

- IAS 34
- IBM 7094 42
- IBM system 360 48
- , komputery serii 700/7000 41
- IBR 37, 42
- indeksowanie 427
- wstępne 428
 - wtórne 428
- instrukcje wejścia-wyjścia 231
- Intel 80486, przetwarzanie potokowe 477
- Intel 8085 593-599
- interfejs równoległy 249
- SCSI 250
 - -, fazy 251
 - szeregowy 249
- IR 37, 457
- izolowane wejście-wyjście 231

J

- jednolita pamięć podręczna 166
- jednopoziomowa pamięć podręczna 165
- jednostka adresowalna 128
- arytmetyczno-logiczna 26, 321, 453

jednostka centralna 24
- sterująca 26, 453, 579–602, 608–614
- -, mikroprogramowana 608
- -, model 589
- -, sygnały sterujące 588
- transferu 128
jednostki NaN 349, 357
język assemblerowy 402–404
- mikroprogramowania 605

K

kanał multiplekserowy 247
- wejścia-wyjścia 227, 247
- wybiórczy 247
kanały danych 42
kod ASCII 221–222
- EBCDIC 378
- Hamminga 146
- poprawiania pojedynczego błędu 150
- SEC 150
- SEC-DEC 150
kody korekcyjne 146
- warunkowe 399, 456
kolejka długookresowa 288
- krótkookresowa 288
- pośrednia 291
- wejścia-wyjścia 288
kolorowanie grafu 510
komórka pamięci 139–140
komputer IAS 34
komputery drugiej generacji 41
- o zredukowanej liczbie rozkazów 30, 497, 511
konwersja 326, 364, 388
koordynacja asynchroniczna 97
- synchroniczna 96

L

liczby całkowite, arytmetyka 327–342
- -, dzielenie 339–342
- -, mnożenie 331–339
- -, sposób reprezentacji 322–327
- podwójnej precyzji 346
- pojedynczej precyzji 346
- zdenormalizowane 358
- zmiennopozycyjne, arytmetyka 349–359

liczby zmiennopozycyjne, arytmetyka, analiza dokładności 354
- -, zaokrąglanie 355
licznik programu 37, 73, 457
- synchroniczny 731
- szeregowy 730
liczniki 729–733
linie adresowe 90
- danych 90
- sterowania 90–91
lista rozkazów 29, 368
- -, projektowanie 375
logika cyfrowa 31
lokalność 504
- odniesień 184
LSI 51

Ł

łączenie w łańcuchy 673

M

magistrala 89
- Futurebus+ 109–125
- -, arbitraż 113–116
- Omnibus 51
- P1394 259
- -, konfiguracje 260
- PCI 99–109
- -, arbitraż 107
- -, linie sygnałowe 101, 102
- SCSI 250–259
- -, komunikaty 256
- -, przebiegi czasowe 253–255
- -, rozkazy 257
- systemowa 89
-, szerokość 90, 98
mała skala integracji 47
mapy Karnaugh 702, 703
MAR 37, 457
maszyna von Neumanna 33
MBR 36, 457
metody przewidywania rozgałęzień 473–477
mikrooperacje 580
mikroprocesor 54
- 32-bitowy 55
- 8086 55
- 16-bitowy 54

mikroprocesor 8-bitowy 54
 - ogólnego przeznaczenia 54
mikroprogramowanie 27
mikroprogramy programowe 625
 - układowe 625
mikrorozkaz 439, 605
 - LS-11, format 631
 - pionowy 610, 627
 - poziomy 606, 627
 -, wykonywanie 629
mikrorozkazy, metoda kodowania, funkcjonalna 626
 -, metoda kodowania, wg zasobów 626
 -, rodzaje 624
 -, szeregowanie 614–620
 -, taksonomia 622
 -, wykonywanie 620
mikrosekwenser 634, 637
 minikomputer 50, 801 62
MIPS R4000 529–537
 - -, lista rozkazów 530–532
 - -, potok rozkazów 533–537
 mnemoniki 370
 moduł wejścia-wyjścia 218, 224
 - -, struktura 226
 modyfikowanie adresu 38
 monitor 274
 Motorola 88000 523–529
 - -, architektura 525
 - -, lista rozkazów 524
 - -, zarządzanie rejestrami 528
 MQ 37
 multiplexer 42, 711
 multipleksowanie czasowe 95

N

najpierw najmłodszy bajt 416–419
 - najstarszy bajt 416–419
 NaN 349, 357
 nie oznacza liczby 349
 niedomiar dodatni 344
 - mantysy 349
 - ujemny 344
 - wykładnika 349
 nieskończoność 357
 norma IEEE Standard 754 346, 357
 notacja dwójkowa 362
 - dziesiętna 362
 - przyrostkowa 413
 - szesnastkowa 366
 - wrostkowa 413

O

obliczenia wektorowe 669
 odpytywanie łańcuchowe 237
 - za pomocą oprogramowania 237
 odwrotna notacja polska 413
 odwrócona tablica stron 299
 odwzorowanie „cienkokońcowe”
 416–419
 - „grubokońcowe” 416–419
 okna rejestrów 504
 okno rozkazu 554
 Omnibus 51
 operacje arytmetyczne 385
 - dotyczące stosu 411–413
 - logiczne 385–388
 -, rodzaje 381
 - transferu danych 383–384
 - wejścia-wyjścia 388
 operatory logiczne 695
 opóźnienie bramkowe 697
 - obrotowe 197
 opóźnione ładowanie 522
 oprogramowanie 71
 organizacja komputera 20
 - rejestrów 479–483, 486–490
 ortogonalność 444

P

pamięci stałe 138
 pamięć, bloki 128
 -, błąd przypadkowy 145
 -, - stały 145
 - błyskawiczna 139
 -, czas dostępu 129
 -DRAM 137, 144
 - -, organizacja 174
 - EEPROM 139
 - EPROM 139
 - główna 24
 - głównie do odczytu 139
 -Jednostka transferu 128
 -, kody korekcyjne 146
 -, korekcja błędów 145
 - na rdzeniach magnetycznych 52
 - nieulotna 130
 -, odświeżanie 142
 -, organizacja 130, 139

- pamięć podręczna 151–173
 - , algorytmy zastępowania 162–163
 - – DRAM 176
 - , efektywność dostępu 188
 - , funkcja odwzorowania 154
 - , jednolita 166
 - , lokalność odniesień 184
 - , odwzorowanie bezpośrednie 155
 - –, sekcijno-skojarzeniowe 159
 - , – skojarzeniowe 158
 - , podzielona 166
 - , poziomy 165
 - , rozmiar bloku 163
 - , spójność 163–164, 662–669
 - , stany 169, 666
 - , współczynnik trafień 186
 - , zewnętrzna 165
 - , pojemność 128
 - półprzewodnikowa 52
 - o dostępie swobodnym 137
 - PROM 138
 - RAM 137
 - , ramka 294
 - RAMLINK 178
 - RDRAM 178
 - rdzeniowa 52, 136
 - , rodzaje dostępu 129
 - ROM 138, 718
 - rozszerzona 136
 - rzeczywista 297
 - , segmentacja 302
 - SDRAM 176
 - , tryb pakietowy 177
 - , słowo 128
 - stała 130
 - sterująca 606
 - , strona 294
 - , stronicowanie 294
 - taśmowa 213
 - ulotna 130
 - wewnętrzna 28
 - wirtualna 296
 - , wydajność 129
 - , wymiana 290–291
 - , zarządzanie 290
 - zewnętrzna 29
- paralelizm na poziomie maszyny 551, 557
- paralelizm na poziomie rozkazu 547, 550
- partycje 292
- partycjonowanie 292
- PC 37, 457
- PDP-1 41
- PDP-10, formaty rozkazu 441
- PDP-11, formaty rozkazu 442–443
- PDP-8 50
- PDP-8, formaty rozkazu 440
- Pentium 60, 167–169, 303, 307, 378–379, 396–400, 429–433, 479–486, 567–572
 - , dobieranie rozkazów parami 569
 - , formaty rozkazu 444–447
 - , kody warunkowe 399
 - , organizacja pamięci 303
 - , – podręcznej 167–169
 - , – rejestrów 479–483
 - , przetwarzanie przerwań 484–486
 - , przewidywanie rozgałęzień 571
 - , rejestr EFLAGS 481
 - , – sterowania 482–483
 - , rodzaje danych 378–379
 - , – operacji 396–400
 - , segmentacja pamięci 303
 - , stronicowanie pamięci 306
 - , struktura potoku 568
 - , tryby adresowania 429–433
 - , zarządzanie pamięcią 303, 307
- PLA 715–718
- pobieranie na zakładkę 466
 - z wyprzedzeniem 466
- podprogram obsługi przerwania 80
 - standardowy 392
- podprogramy wielowejściowe 394
- podstawa stosu 413
- podział czasu 282
- podzielona pamięć podręczna 166
- pojemność pamięci 128
- połączenia procesora 27
 - systemu 24
- potok rozkazów 467–471
- PowerPC 62, 308–314, 400–402, 486–493, 559–567, 620 565

- PowerPC, formaty rozkazu 447–449
 - , organizacja pamięci podręcznej 172
 - , organizacja rejestrów 486–490
 - , potoki rozkazów 561
 - , przetwarzanie przerwania 490–493
 - , – rozgałęzień 563
 - , rodzaje danych 380–381
 - , – operacji 400–402
 - , rodzina 63
 - , schemat blokowy 559
 - , struktura potoku 560
 - , tryby adresowania 433–436
 - , zarządzanie pamięcią 308–314
 - prawa De Morgana 696
 - prawdziwa zależność danych 547
 - prawo Moore'a 56
 - procedura rekurencyjna 394
 - proces 283
 - , stany 284–285
 - wejścia-wyjścia 23
 - procesor 24
 - , organizacja rejestrów 454
 - , organizacja wewnętrzna 592
 - , struktura wewnętrzna 454
 - tablicowy 670, 676
 - wejścia-wyjścia 227, 247
 - wektorowy 676
 - procesory Intel 61
 - równoległe 31, 676, 684
 - –, ogólna klasyfikacja 687
 - –, rodzaje 685
 - superskalarne 30, 544
 - wyspecjalizowane funkcjonalnie 652
 - program sprzętowy 70
 - symboliczny 403
 - programowalne tablice logiczne 715–718
 - programowane wejście-wyjście 228
 - protokół MESI 666
 - –, graf przejść 667
 - przedział równorzędności 262
 - przekazywanie sterowania 389
 - przenoszenie danych 37
 - przepełnienie 329
 - dodatnie 344
 - mantysy 349
 - ujemne 344
 - wykładnika 349
 - przerwa izochroniczna 264
 - przerwania 78, 233, 484–486, 490–493
 - , przetwarzanie 233, 484–486, 490–493
 - wielokrotne 84
 - zablokowane 84
 - przerwanie wektorowe 238
 - przerzutnik 723
 - D 725
 - J–K 726
 - S–R 723
 - –, synchronizowany 725
 - przesunięcie arytmetyczne 336, 387
 - cykliczne 387
 - logiczne 386
 - przetwarzanie potokowe 465, 519
 - –, optymalizacja 519
 - –, pobieranie na zakładkę 466
 - –, – z wyprzedzeniem 466, 472
 - –, postępowanie z rozgałęzieniami 471
 - –, przewidywanie rozgałęzień 473–477
 - –, strategia 466–471
 - wewnątrz operacji 673
 - –, zwielokrotnione strumienie 471
 - przerwania 484–486, 490–493
 - równoległe 671
 - superpotokowe 545
 - tablicowe 676
 - wektorowe 31, 671
 - wieloprotokolowe 652
 - przewidywanie rozgałęzień 56, 473–477, 558
 - PSW 234, 457
- R**
- RAID 197
 - ramka 294
 - stosu 395
 - reguła odejmowania 330
 - przepełnienia 330
 - rejestr adresowy pamięci 37, 457
 - wejścia-wyjścia 71
 - adresu sterowania 608
 - buforowy pamięci 36, 457
 - EFLAGS 480
 - indeksowy 428
 - mnożenia-dzielenia 37
 - przesuwający 729
 - rozkazu 37, 73, 457
 - równoległy 729

rejestr wartości przedrostka 659
- wektorowy 673
rejestry 27, 321, 727–729
- adresowe 455
- danych 455
- indeksowe 455
- ogólnego przeznaczenia 455, 479
-, organizacja 458–461
- robocze 455, 479
- stanu 456
- sterowania 456
- widzialne dla użytkownika 455
reprezentacja przesunięta 343
- stałopozycyjna 327
- uzupełnienia do dwóch 323
- zmiennopozycyjna 342
- znak-moduł 322
rezydentny program zarządzający 274
RISC 30
RISC System/6000 62
rozgałęzienia bezwarunkowe 37
- warunkowe 37
rozkaz, cykl adresowania pośredniego 583
-, - pobierania 581
-, - pośredni 461
-, - przerwania 583
-, - wykonywania 584
- długość 436–439
-, etapy wykonywania 369
-, format 370, 436
-, przetwarzanie potokowe 465–471
-, - -, strategia 466–471
-, reprezentacja symboliczna 370
-, rodzaje 371
-, - argumentów 376
- rozgałęzienia warunkowego 390
- wywołania podprogramu standardowego 392
rozkazy arytmetyczne 372
- logiczne 372
- magistrali PCI 104
- maszynowe 368
- -, lista 368
- międzyprocesorowe IBM S/370 660
- o zmiennej długości 442
- pamięciowe 372
- PCI 104

rozkazy PDP-11 76
- pominięcia 391
- rozgałęzienia 372, 390
- testowania 372
- wejścia-wyjścia 229, 372

S

SCSI 250
segmentacja 302
sektor 192
skalowalność 688
słowo 128
- stanu programu 234, 457
- sterujące 605
spekulatywne wykonywanie rozkazów 57
spójność pamięci podręcznej 662–669
—, protokoły katalogowe 664
—, - podglądania 664, 665
—, protokół MESI 666
—, rozwiązania programowe 663
—, - sprzętowe 664
standard Futurebus+ 110–111
- PCI 101–103
stany procesu 284–285
statyczna pamięć RAM 138
sterowanie 389
- mikroprogramowe 31
- resztkowe 619
- wg Wilkesa 610–613
sterownik magistrali 96
- wejścia-wyjścia 227, 247
stopniowy niedomiar 358
stos 374, 413
strategia wydawania rozkazów 551
strona 294
stronicowanie 294
- na żądanie 296
struktura połączeń 28, 87
- systemu 21
struktury „cienkokońcowe” 416–419
- „grubokońcowe” 416–419
- wielomagistralowe 92
sumator 719
sygnalizacyjna NaN 357
sygnały stanu 220
- sterowania 220
symulacja pola ciągłego 669

synchroniczna pamięć DRAM 176
 system binarny 362
 - dziesiętny 362
 - jednoprogramowy 273
 - **konwersacyjny** 273
 - operacyjny 270
 - -, rodzaje 273
 - wsadowy 273
 systemy komputerowe, kategorie 686
 - liczbowe 362
 - procesorów równoległych 684
 - wieloprocessorowe, luźno powiązane 652
 - -, silnie powiązane 652
 - z przetwarzaniem równoległym 652
 szamotanie 296
 szeregowanie czasowe 283
 - krótkookresowe 284
 szeregowanie wysokiego poziomu 284
 szybkość przesyłania 130
 szyna adresowa 90
 - rozszerzenia 93

T

tablica prawdy 695
 - Quine'a-McKluskeya 702
 - stron 295, 297
 - wektorów przerwań 484
 — własności 724
 - wzbudzania 732
 TLB 299
 tożsamości algebry **Boole'a** 696
 trafienie odczytu 668
 - zapisu 668
 transmisja asynchroniczna 263
 - synchroniczna 263
 tryb z podziałem czasu 282
 tryby adresowania 421–422

U

układ kombinacyjny 699
 układy GSI 48
 - sekwencyjne 722
 - sterowania 220
UNIVAC I 40
UNIVAC II 40
 upakowanie 293

urządzenia peryferyjne 219
 - wejścia-wyjścia 86
 - zewnętrzne 219
 - -, sygnały stanu 220
 - -, - sterowania 220
 urządzenie peryferyjne 23

V

VLSI 51

W

wejście-wyjście 24, 29
 -, izolowane 231
 -, odwzorowane w pamięci 231
 -, sterowane przerwaniem 228
 wewnętrzna magistrala procesora 453
 węzeł nadrzędny cyklu 264
 wielka skala integracji 51
 wielokomputer 687, 688
 wieloprocessor 31, 652, 687, 688
 - **IBM 3033** 661
 wieloprocessorowe systemy operacyjne 657
 wieloprogramowanie 273, 277
wieloprzetwarzanie, przechowywanie bezwzględne 659
 -, - **rzeczywiste** 659
 wskaźnik stosu 413, 455
 wskaźniki segmentu 455
 współczynnik trafień 186
 współpraca z systemem operacyjnym 29
 wydawanie rozkazów 551
 wyjątek 484
 wymiana 290–291

Z

zagnieżdżenie 392
 zależność przepływu 547
 - wyjściowa 553
 - zapis-odczyt 547, 553
 - zapis-zapis 555
 zaokrąglenie do najbliższej 355
 - w kierunku $+\infty$ 355
 - w kierunku 0 355
 - w kierunku $-\infty$ 355
 zapis jednoczesny 163

zapis opóźniony 163

— z aktualizacją 665

- z unieważnieniem 665

zarządzanie pamięcią 290

—, segmentacja 302

—, stronicowanie 294

~~zastępowanie~~ pamięcią, wymiana 290–291

~~zasada~~ lokalności odniesień 133

~~zbiorek~~ 723

~~zbiór~~ funkcjonalnie pełny 697, 698

~~zinczniki~~ 322

~~zinc~~ 456



WYDAWNICTWA NAUKOWO-TECHNICZNE

ul. Mazowiecka 2/4, 00-048 Warszawa

tel. 826-72-71 do 79

Dział Marketingu i Sprzedaży

tel. 827-56-87, fax. 826-82-93

e-mail: marketing@wnt.com.pl

WNT Warszawa 2000. Wyd. I

Ark. wyd. 61,5. Ark. druk. 48,25

Symbol Et/83442/MEN

Cieszyńska Drukarnia Wydawnicza

Zam. nr 191/K-2000

Akronimy

ALU	Jednostka arytmetyczno-logiczna
ASCII	Znormalizowany amerykański kod wymiany informacji
ANSI	Amerykański Narodowy Instytut Normalizacji
BCD	Liczba dziesiętna kodowana binarnie
CD	Płyta kompaktowa
CD-ROM	Pamięć stała na płycie kompaktowej
CPU	Jednostka centralna (procesor)
CISC	Komputer o złożonej liście rozkazów
DRAM	Dynamiczna pamięć o dostępie swobodnym
DMA	Bezpośredni dostęp do pamięci
EPROM	Wymazywalna, programowalna pamięć stała
EEPROM	Elektrycznie wymazywalna, programowalna pamięć stała
HLL	Język wysokiego poziomu
I/O	Wejście-wyjście
IAR	Rejestr adresu rozkazu
IC	Układ scalony
IEEE	Instytut Elektrotechniki i Elektroniki*
IR	Rejestr rozkazów
LRU	Najmniej ostatnio używany
LSI	Wielka skala integracji
MAR	Rejestr adresowy pamięci

*Amerykańska instytucja normalizacyjna (przyp. red.)

MBR	Rejestr buforowy pamięci
MESI	Zmodyfikowany- wyłączny-wspólny nieważny
MMU	Jednostka zarządzania pamięcią
MSI	Średnia skala integracji
PC	Licznik rozkazów
PCI	System połączeń urządzeń peryferyjnych
PROM	Programowalna pamięć stała
PSW	Słowo stanu procesora
PCB	Blok sterowania procesem
RAID	Redundantna tablica niezależnych dysków
RALU	Rejestrowa jednostka arytmetyczno-logiczna
RAM	Pamięć o dostępie swobodnym
RISC	Komputer o zredukowanej liście rozkazów
ROM	Pamięć stała
SCSI	Interfejs małych systemów komputerowych
SMP	Wieloprzetwarzanie symetryczne
SRAM	Statyczna pamięć o dostępie swobodnym
SSI	Mała skala integracji
VLSI	Bardzo wielka skala integracji
WORM	Jednokrotny zapis, wielokrotny odczyt



Oto kolejna książka **Wydawnictw Naukowo-Technicznych** o budowie i działaniu nowoczesnych systemów komputerowych.

William Stallings - wybitny specjalista w tej dziedzinie - kładzie nacisk na projektowanie systemu pod kątem uzyskania jak największej wydajności. Omawia:

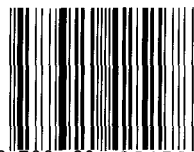
- historię komputerów;
- budowę systemu komputerowego, w tym strukturę wewnętrznych połączeń (magistrale), pamięci wewnętrzne i zewnętrzne oraz urządzenia wejścia-wyjścia;
- procesory, z uwzględnieniem arytmetyki komputerowej, listy rozkazów, struktury i organizacji rejestrów, przetwarzania potokowego, architektury RISC, CISC i superskalarnej;
- jednostkę sterującą i sterowanie mikroprogramowe;
- przetwarzanie z wykorzystaniem wieloprocesorów i procesorów równoległych.

Wartość dzieła podnoszą doskonale dobrane przykłady najważniejszych rozwiązań.

Rozwój technologii komputerowej ma dziś podstawowe znaczenie dla cywilizacji. Gorąco polecamy tę książkę **studentom elektroniki i informatyki**, bo to oni będą w przyszłości wytyczać kierunki tego rozwoju.

Cena 60,- zł

ISBN 83-204-2517-4



9 788 20 425178 >