CENTRUM KOMPUTEROWYCH SYSTEMÓW AUTOMATYKI I POMIAROW „MERA-ELWRO"
BIURO OBSŁUGI TECHNICZNEJ „ELWRO-SERWIS"—OŚRODEK SZKOLENIOWY

WPROWADZENIE DO JEDNOLITEGO SYSTEMU ELEKTRONICZNYCH MASZYN CYFROWYCH

WROCŁAW 1976

CENTRUM KOMPUTEROWYCH SYSTEMÓW AUTOMATYKI I POMIARÓW „MERA-ELWRO"
BIURO OBSŁUGI TECHNICZNEJ „ELWRO-SERWIS" — OŚRODEK SZKOLENIOWY

WPROWADZENIE DO JEDNOLITEGO SYSTEMU ELEKTRONICZNYCH MASZYN CYFROWYCH

Opracował: Thanasis Kamburelis

WROCŁAW 1976

M2g

Do użytku wewnętrznego

Eo.77/332/2
11.2 20,-



566889
II

WPM „WEMA", Warszawa 1976. Wydanie I. Nakład 2000+60 egz.
Format papieru A1. Ark. wyd. 17,71. Ark. druk. 14,5. Zam. 159/76-Z/C

OZGraf. Zam. 1768

1. Wstęp

1.1. Przeznaczenie skryptu

Skrypt ten stanowi wprowadzenie do struktury i zasad działania Elektronicznych Maszyn Cyfrowych Jednolitego Systemu (JS EMC) oraz opisuje konkretne jednostki centralne i urządzenia zewnętrzne wchodzące w skład JS EMC. Skrypt przeznaczony jest zarówno dla słuchaczy kursów technicznych — jednostek centralnych lub urządzeń zewnętrznych — jak i dla słuchaczy kursów programowania, którzy pragną zapoznać się z ogólnymi zasadami działania EMC Jednolitego Systemu.

Autor skryptu zakłada, że czytelnik jest ogólnie zaznajomiony z maszynami cyfrowymi, a zwłaszcza z terminologią podstawowych określeń i pojęć występujących w maszynach cyfrowych [12].

Część pierwsza skryptu dostarcza czytelnikowi ogólnych wiadomości o maszynach cyfrowych, w tym wiadomości o podstawach arytmetycznych, o strukturze logicznej i o zasadach działania maszyn cyfrowych. Część pierwsza — zawierająca rozdziały 1 i 2 — jest przeznaczona głównie dla słuchaczy niezaawansowanych w zakresie architektury logicznej maszyn cyfrowych i może być pominięta przez bardziej zaawansowanych czytelników bez ryzyka napotkania trudności w korzystaniu z rozdziałów części następnych.

Część druga (rozdziały 3 i 4) omawia podstawowe zasady działania EMC Jednolitego Systemu. W części tej opisuje się przyjętą strukturę danych i instrukcji oraz strukturę ważniejszych zespołów występujących w EMC Jednolitego Systemu.

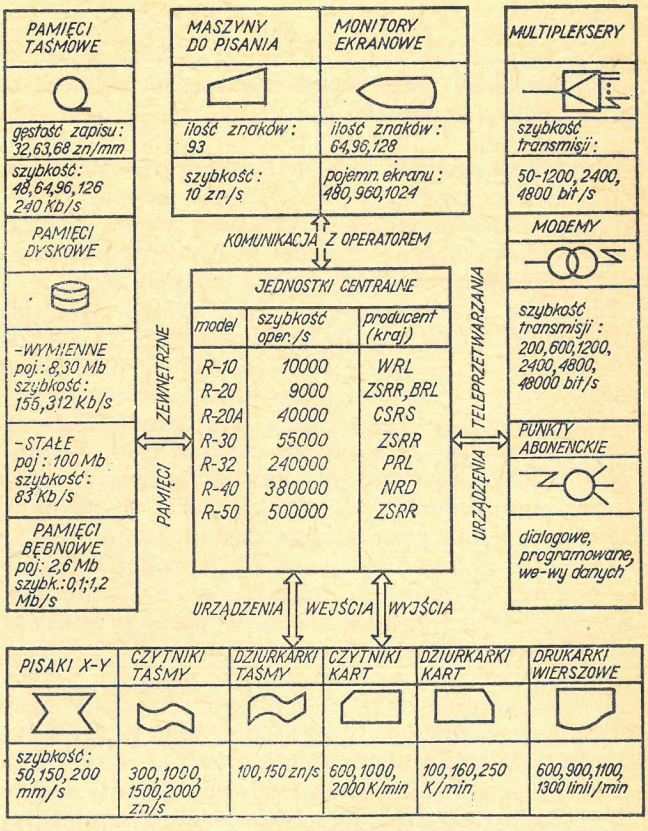
Część trzecia (rozdział 5) omawia charakterystykę jednostek centralnych JS EMC, zaś część czwarta (rozdziały od 6 do 10) omawia wybrane rozszerzone zasady działania jednostek centralnych, takich jak: pamięci buforowe, ogólna koncepcja maszyn wirtualnych, systemy wielomaszynowe i wieloprocesowe, problemy zgodności programowej i inne. Część czwarta opisuje również konkretne realizacje urządzeń zewnętrznych Jednolitego Systemu oraz uzyskane parametry techniczne tych urządzeń.

1.2. Ogólne wiadomości o EMC Jednolitego Systemu

Jednolity System Elektronicznych Maszyn Cyfrowych — składający się z szeregu jednostek centralnych o zróżnicowanej wydajności obliczeniowej, lecz o jednakowej architekturze logicznej i o wspólnych zasadach działania, oraz z bogatego zestawu urządzeń zewnętrznych — jest wynikiem współpracy naukowo-technicznej i przemysłowej krajów socjalistycznych.

Maszyny Jednolitego Systemu (RIAD) zostały tak pomyślane, aby mogły optymalnie zaspokajać potrzeby przetwarzania informacji w zakresie planowania i zarządzania gospodarką, automatyzacji sterowania procesami przemysłowymi, automatyzacji prac zawodowych, itp.

W opracowaniach EMC Jednolitego Systemu zastosowano najnowsze osiągnięcia krajów socjalistycznych z zakresu technologii, bazy elementowej, metod opracowania, wytwarzania oraz eksploatacji środków elektronicznej techniki obliczeniowej. Uwzględniono również wymagania nowych metod zastosowań, takich jak: dostęp do EMC za pośrednictwem środków telełączności, praca w trybie dialogowym, jednoczesny dostęp do centralnej EMC wielu użytkowników, praca w systemach wirtualnych, praca w systemach



Rys. 1.1. Środki techniczne EMC Jednolitego Systemu
Uwaga. Szybkości EMC od R-20 do R-50 są wzajemnie porównywalne, gdyż stosują identyczne formaty informacji oraz jednakową listę instrukcji użytkowych, natomiast maszyna R-10 ma odmienną i prostą listę rozkazów (właściwą dla minikomputerów)

Charakterystyka techniczna jednostek centralnych JS EMC Tabela 1.1

Parametr lub cecha	Jednostka centralna JS EMC						
	R-10	R-20	R-20A	R-30	R-32	R-40	R-50
1	2	3	4	5	6	7	8
1. Procesor							
— szybkość obliczeniowa, tys. oper./s	10	9	40	55	240	380	500
— zestaw instrukcji (specjalny, uniwersalny)	S	U	U	U	U	U	U
— długość przetwarzanych informacji w procesorze, bitów	16	8	8	32	32	64	64
— typ sterowania (M-mikroprogramowe, U — układowe)	M	M	M	M	M	U+M	U
— cykl procesora, ns	300	1000	300	600	300	200	160
2. Pamięć operacyjna							
— pojemność, Kbajtów	8—64	64—256	64—256	128—512	128—1024	256—1204	256—1024
— cykl pracy, μs	0,8	2	2	1,25	1,2	1,35	1,25
— ilość bloków logicznych	1	1	1	1	1	2—4	2—4
3. Kanały selektorowe							
— ilość	1	1—2	1—2	2—3	3	3—6	3—6
— szybkość przesyłania, Kbajtów/s	200	200	250	800	1100	1300	1300
— liczba podłączonych	256	256	160	256	256	256	256
4. Kanał multiplekserowy							
— szybkość przesyłania, Kbajtów/s, w trybie:							
selektorowym	140	100	220	300	250	670	670
multiplekserowym	30	16	35	40	110	110	110
— liczba podkanałów	127	128	16	128	128	196	196
5. Baza elementowa	TTL	TTL	TTL	TTL	TTL	TTL	ECL
6. Moc pobierana przez JC, kV·A	2	16	13	21	3,6	30	40

Uwaga. Szybkość pracy procesorów została obliczona metodą GIBSON 1 (przyjęta w JS EMC interpretacja tej metody jest podana w dodatku B)

wieloprocesorowych lub wielomaszynowych itp. Opracowano również odpowiednio bogate oprogramowanie systemowe i użytkowe w celu właściwego wykorzystania środków technicznych Jednolitego Systemu oraz jego nowoczesnych cech.

Pierwszą polską maszyną Jednolitego Systemu jest maszyna cyfrowa R-32. Maszyna R-32 — podobnie jak i maszyny R-20, R-30, R-40 i R-50 (patrz rys. 1.1) — należy do tzw. pierwszego pokolenia maszyny Jednolitego Systemu (RIAD I). Wyżej wymienione maszyny mają w pełni jednolitą architekturę logiczną, chociaż bazują na różnych rozwiązaniach konstrukcyjnych i różnych technologiach wytwarzania. Dzięki temu można było przyjąć również jednolite oprogramowanie na poziomie instrukcji maszynowych. Natomiast maszyny R-10 i R-20A nie są w pełni programowo zgodne z pozostałymi maszynami. Przyjęcie zgodnej architektury logicznej większości jednostek centralnych i urządzeń zewnętrznych oraz jednolitego oprogramowania stworzyło praktyczne możliwości skoncentrowania kadry naukowo-technicznej i potencjału przemysłowego krajów socjalistycznych (Bułgarii, Czechosłowacji, Niemieckiej Republiki Demokratycznej, Polski, Węgier i Związku Radzieckiego, a od początku 1974 r. także Kuby i Rumunii) nad realizacją wspólnego i ambitnego programu Elektronicznej Techniki Obliczeniowej, tj. Programu RIAD, który powinien w sposób optymalny zaspokoić potrzeby współpracujących krajów w zakresie informatyki. Przy czym już dzisiejsze potrzeby, a także prognozowane potrzeby informatyki są tak ogromne i tak zróżnicowane, że chyba żaden kraj (z wyjątkiem może Związku Radzieckiego) nie mógłby sobie pozwolić na samodzielny i szeroki program realizacji tych potrzeb. Oczywiście międzynarodowy podział pracy stwarza dodatkowe warunki do zmniejszenia ogólnych kosztów opracowań i wytwarzania oraz zwiększenia jakości wyrobów.

Jednolitość środków technicznych i systemów programowania stwarza również możliwość międzynarodowej współpracy i wymiany dorobku w zakresie systemów zastosowań informatyki w różnych dziedzinach gospodarki narodowej, co ma znaczenie nie mniejsze niż wspólny program budowy sprzętu komputerowego.

Wszystkie wyżej wymienione argumenty złożyły się na decyzję wyboru Jednolitego Systemu EMC RIAD.

O wielkości i złożoności programu RIAD świadczy fakt, że aczkolwiek koncepcja systemu RIAD powstała jeszcze w 1968 roku, to jednak na pierwsze praktyczne rezultaty współpracy trzeba było czekać do maja 1973 roku, kiedy to na specjalnej wystawie w Moskwie praktycznie

zademonstrowano system RIAD. W roku 1975 Jednolity System EMC (patrz rys. 1.1 i tabela 1.1) liczył 7 różnych jednostek centralnych: R-10, R-20, R-20A, R-30, R-32, R-40 i R-50 oraz około 130 różnych urządzeń zewnętrznych, które mogą współpracować z dowolną z tych jednostek centralnych, niezależnie od kraju ich opracowania i produkcji. Opracowany zestaw środków sprzętowych i programowych JS pozwala, dzięki modułowej konstrukcji, na tworzenie licznych konfiguracji użytkowych, zgodnie z wymaganiami użytkownika. Ta uniwersalność zastosowań EMC Jednolitego Systemu została zapewniona dzięki przyjęciu:

— współczesnej i zgodnej organizacji logicznej środków technicznych, gdzie w dużym stopniu zapewniono równoległość pracy poszczególnych elementów systemu liczącego;
— bogatego zbioru operacji zapewniających efektywne przetwarzanie dla różnych postaci wprowadzanych informacji;
— standaryzowanych zasad technicznej, logicznej i programowej współpracy jednostek centralnych z licznymi urządzeniami zewnętrznymi;
— modułowej struktury i konstrukcji występującej w różnych poziomach implementacji sprzętu (np. ilość kanałów, wielkość pamięci operacyjnej, ilość procesorów, ilość EMC w kompleksie obliczeniowym itp.);
— oprogramowania modułowego i wyposażonego w różnorodne funkcje i środki programowania.

Środki techniczne Jednolitego Systemu przechodzą ciągłą modernizację przemysłową, podlegają różnym istotnym rozszerzeniom strukturalnym i funkcjonalnym, a także co rok zwiększa się park produkowanych środków Jednolitego Systemu.

W ramach ustalonego podziału pracy Polska produkuje maszyny średniej klasy, w pierwszej kolejności maszyny typu R-32 oraz wybrane urządzenia zewnętrzne, np. drukarki, pamięci taśmowe, urządzenia taśmy papierowej, multipleksery, punkty abonenckie, systemy monitorowe itd.

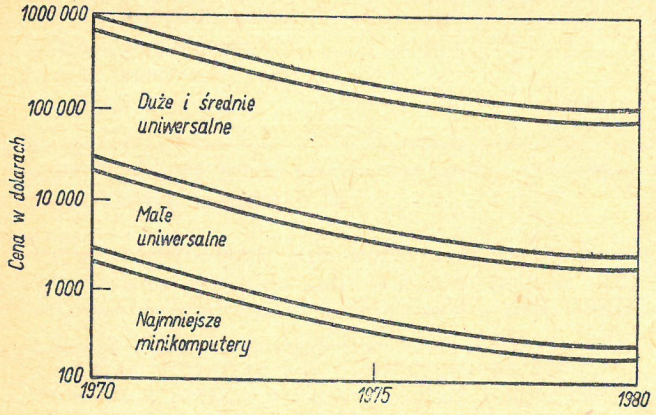
Polska maszyna Jednolitego Systemu R-32 pojawiła się nieco później od pozostałych modeli systemu RIAD (zademonstrowana została po raz pierwszy na MTP 1974 r., zaś jej produkcję seryjną rozpoczęto w 1975 r.), ale dzięki temu można było przy jej konstrukcji wykorzystać najnowsze krajowe rozwiązania konstrukcyjno-technologiczne oraz elementy scalone średniej skali integracji typu TTL. Przyjęta baza technologiczna i elementowa pozwoliła uzyskać wysokie współczynniki techniczno-ekonomiczne (np. stosunek szybkości do kosztów wytwarzania).

Dzięki wysokiej wydajności obliczeniowej maszynę R-32 można zaliczyć do klasy średnich lub dużych jednostek centralnych, zależnie od wielkości wbudowanej pamięci operacyjnej i ilości kanałów przesyłania informacji. Szybkość maszyny wynosi około 500 000 dodawań na sekundę, zaś średnia szybkość według mieszanki GIBSON I — ponad 240 000 operacji na sekundę; wielkość wbudowanej pamięci operacyjnej waha się od 128 Kb do 1024 Kb.

2. Podstawowe wiadomości o maszynach cyfrowych

2.1. Wprowadzenie

Elektroniczne maszyny cyfrowe znalazły w ostatnich latach szerokie zastosowanie w różnorodnych dziedzinach nauki, techniki i gospodarki narodowej. Wraz ze wzrostem zastosowań wzrasta również liczba zainstalowanych komputerów na świecie oraz ich szybkość działania, zaś ich względna cena maleje (rys. 2.1). Szacuje się, że pod koniec 1975 r. będzie na świecie około 250 000 komputerów, w tym: USA — 150 000, Japonia — 16 000, NRF — 14 000, Francja — 11 400, W. Brytania — 9800, ZSRR — 9000, PRL — 1200. Roczny wzrost zainstalowanych EMC w krajach zaawansowanych jest bardzo duży i wynosi przeciętnie 20—30%, czyli liczba komputerów podwaja się w ciągu 3—4 lat.

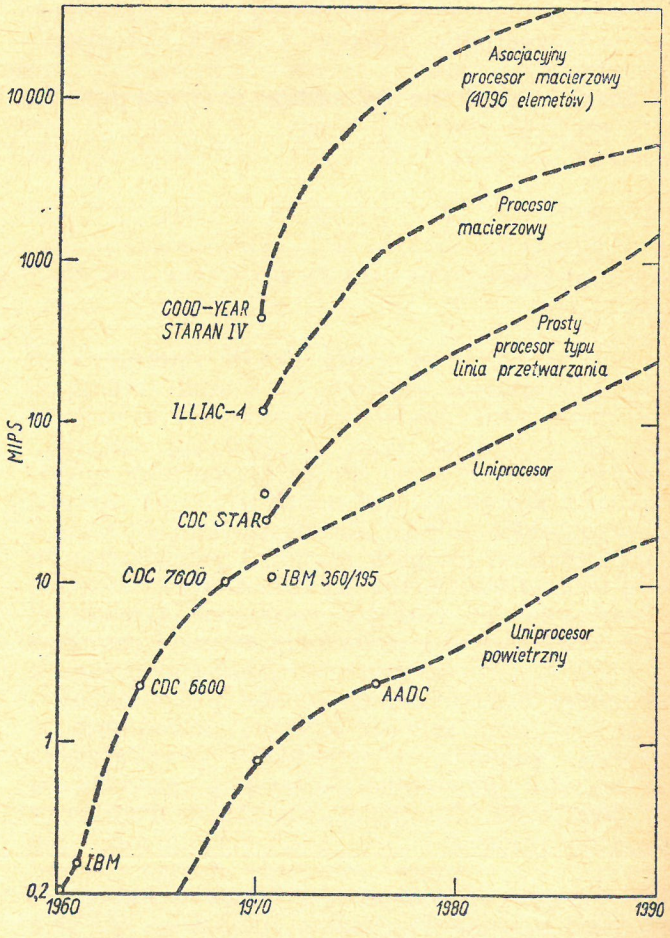


Rys. 2.1. Cena jednostki przetwarzania

Ostatnie 20 lat w technice komputerowej charakteryzowało się wzrostem prędkości działania, pojemności pamięci oraz niezawodności. Jednocześnie zmniejszeniu uległy rozmiary fizyczne, zużycie mocy oraz koszty. Wzrost prędkości obliczeniowej uzyskano dzięki zastosowaniu nowych elementów i technologii produkcji oraz nowej architekturze systemów komputerowych, jak: linie przetwarzania *(pipe-line)*, procesory macierzowe, procesory asocjacyjne oraz systemy pamięciowe z przeplotem *(interleaved)*. Jednocześnie wieloprogramowość i podział czasu wpłynęły na bardziej efektywne wykorzystanie systemów komputerowych.

Powstają nowe generacje elementów i technologie produkcji, które powodują skokowy wzrost szybkości działania komputerów. Technika komputerów drugiej połowy lat 70-tych przechodzi do tzw. czwartej generacji elementów, tj. do półprzewodnikowych elementów scalonych średniej i wielkiej skali integracji (MSI i LSI), gdzie w jednej standardowej kostce przechowuje się od 10 do 100 bramek logicznych — w przypadku elementów MSI — i od 100 do 1000 bramek logicznych w przypadku LSI.

Poprzednie techniczne generacje komputerów to: elementy przekaźnikowe (tzw. generacja zerowa), lampy elektronowe (generacja pierwsza), tranzystory (generacja druga) i układy scalone tzw. standardowej skali integracji, w których wmontowuje się do 10 bramek logicznych (generacja trzecia).



Rys. 2.2. Przewidywana wydajność procesorów

Szybkość obliczeniowa komputerów rośnie również nieustannie (rys. 2.2): od kilkunastu operacji do sekundę w przypadku komputerów przekaźnikowych (np. MARK z roku 1944), około 1000 operacji dla komputerów lampowych (np. ENIAC

w roku 1947) do około 100 milionów operacji na sekundę (tj. w MIPS) w przypadku tzw. uniprocesorów (tj. dla procesorów wykonujących jeden strumień instrukcji i jeden strumień danych) i około 1000 milionów w przypadku tzw. procesorów macierzowych (prognozowany postęp lat 1975—79), gdzie występuje jeden strumień instrukcji i wiele (np. 256) strumieni danych.

Wyżej podane informacje dotyczą w dużej mierze perspektywy rozwoju maszyn cyfrowych. Teraz powróćmy do omówienia podstawowych typów maszyn liczących, krótkiej historii powstania maszyn cyfrowych oraz do omówienia podstawowych arytmetycznych i logicznych zasad działania maszyn cyfrowych.

2.2. Maszyny cyfrowe a maszyny analogowe

Istnieją dwa podstawowe typy maszyn liczących: maszyny cyfrowe i maszyny analogowe.

Cyfrowa maszyna licząca wykonuje działania arytmetyczne (i inne operacje) na liczbach przedstawionych w postaci cyfrowej. Przykładem są różne typy kalkulatorów mechanicznych, w których cyfry dziesiętne są przedstawione za pomocą odpowiednich kół zębatych. W automatycznie liczących maszynach cyfrowych liczby są reprezentowane przez ciągi sygnałów elektrycznych. Na ogół wyróżnia się dwa stany sygnałów: istnienie sygnału o określonej wielkości (np. +5 V) lub brak sygnału. Istnienie sygnału oznacza cyfrę jeden (1), zaś brak sygnału cyfrę zero (0). Zatem wyróżnia się dwie cyfry: 0 i 1. Cyfry te nazywa się bitami, zaś układ liczenia binarnym (lub dwójkowym). Na przykład liczba 19 w układzie binarnym będzie przedstawiona w postaci ciągu bitów: 10011. Istotną zaletą maszyn cyfrowych jest możliwość uzyskania dowolnej dokładności obliczeń (np. przez zwiększenie ilości pozycji cyfrowych) oraz duża uniwersalność w zastosowaniach. W maszynie cyfrowej R-32 stosuje się 16 lub 32 bity do reprezentowania liczb całkowitych, co pozwala przedstawiać liczby z 4 lub 8 cyframi dziesiętnymi.

Natomiast zasada działania analogowych maszyn liczących nie opiera się na pojęciu cyfry. W analogowych maszynach liczby są reprezentowane przez wartość określonych wielkości fizycznych, takich jak długość, kąt, potencjał elektryczny itp. Operacje arytmetyczne są wykonywane przez zastosowanie określonych praw fizyki (np. prawo Ohm'a). Prostymi przykładami maszyny analogowej są: suwak rachunkowy, planimetr, integrator, analizatory równań różniczkowych lub liniowych algebraicznych itp.

Wadą maszyn analogowych jest ograniczona dokładność obliczeń i mała uniwersalność w zastosowaniach (aczkolwiek w wielu wyspecjalizowanych zastosowaniach maszyny analogowe są szybsze i efektywniejsze). Dla niektórych zastosowań korzystne jest zbudowanie maszyny liczącej w której stosuje się zarówno technikę cyfrową (w określonych jej blokach), jak i technikę analogową. Taka maszyna nazywa się hybrydową maszyną liczącą.

2.3. Historia maszyn liczących w świecie

Rozwój maszyn liczących (rachunkowych) wiąże się ściśle z rozwojem matematyki i techniki, z rozwojem potrzeb i możliwości zastosowań [2]. Pierwsze prymitywne narzędzia, które można by uznać za prototypy dzisiejszych maszyn matematycznych, powstały jednocześnie z tworzeniem się podstawowego pojęcia matematyki — liczby — i służyły w zasadzie do notowania ilości. Najdoskonalszym z nich jest starożytny abak, pierwowzór dzisiejszego liczydła. Abak i liczydło ułatwiały dodawanie i odejmowanie. Pierwsze narzędzie ułatwiające mnożenie i dzielenie skonstruował J. Napier (1550—1627). Urządzenie to jest specjalnie zapisaną na odpowiednich sztabkach tabliczką mnożenia. Pierwszą maszynę rachującą w dzisiejszym tego słowa znaczeniu skonstruował B. Pascal w 1642 r. Maszyna Pascala służyła tylko do dodawania i odejmowania i z niej wywodzą się używane dzisiaj arytmometry mechaniczne. Arytmometr wykonujący cztery działania (dodawanie, odejmowanie, mnożenie i dzielenie) został skonstruowany przez G. W. Leibniza w 1694 r. Podobny arytmometr zbudował A. Stern w Warszawie w 1813 r. Jego konstrukcji jest także maszyna do obliczania pierwiastków kwadratowych.

Dotychczas wymienione maszyny mogły wykonywać tylko pojedyncze działania arytmetyczne. Pierwszą próbę zbudowania maszyny wykonującej ciągi operacji arytmetycznych podjął Ch. Babbagge w 1812 r. Maszyna ta miała pracować „metodą różnic skończonych" Newtona, lecz budowy jej zaniechano. Dopiero Scheutz w 1853 r. skonstruował maszynę pracującą na wymienionej zasadzie. Babbagge opracował także koncepcję maszyny do obliczania dowolnych wyrażeń arytmetycznych, jednakże koncepcja ta, jak i poprzednia, nie została zrealizowana (idea „maszyny analitycznej" Babbagge'a została urzeczywistniona dopiero w nowoczesnych maszynach elektronicznych). Dalszym krokiem naprzód było zbudowanie przez H. Holleritha ok. 1880 r. tzw. maszyny rachująco-analitycznej. Dane nastawiano w niej nie ręcznie z klawiatury, lecz z odpowiednio dziurkowanych kart papierowych, co pozwalało na uzyskanie dużych szybkości rachowania. Maszyny Holleritha znalazły bardzo szerokie zastosowanie i znane są do dziś pod nazwą maszyn statystycznych.

Wszystkie maszyny mechaniczne, a o takich była dotąd mowa, mają dwie zasadnicze wady: stosunkowo małą szybkość działania i dość prymitywną konstrukcję. Zbudowanie maszyny szybciej rachującej, czy też rozwiązującej problemy bardziej skomplikowane, natrafiło na trudności nie do pokonania w mechanice. Dopiero w 1944 r. H. H. Aiken oraz G. Stibitz (St. Zjedn.), wykorzystując elementy teletechniki (przekaźniki, wybieraki itp.), zbudowali maszyny MARK-I oraz BELL-V, których możliwości przekraczały w sposób istotny możliwości maszyn mechanicznych. Nowoczesna technika pozwoliła na uzyskanie szybkości kilkunastu operacji na sekundę oraz na bardziej skomplikowaną konstrukcję. Do wielu jednak celów obie maszyny okazały się zbyt powolne w działaniu. W czasie II wojny światowej wyłonił się problem bardzo szybkiego wykonywania obliczeń balistycznych. Rozwiązanie mogła

przynieść tylko technika elektroniczna. Brak jakiegokolwiek doświadczenia w konstrukcji tego typu urządzeń elektronicznych nie wróżył szybkiego uzyskania pozytywnych rezultatów. Mimo to w St. Zjednoczonych zdecydowano się na budowę elektronicznej maszyny liczącej, którą ukończono w 1945 r. Wyniki okazały się rewelacyjne. Uzyskano szybkość liczenia kilku tysięcy operacji arytmetycznych na sekundę (maszynę tę nazwano ENIAC). Od tego czasu datuje się żywiołowy rozwój elektronicznych maszyn matematycznych. Rok później podobną, lecz mniejszą maszynę zbudowano w Kijowie. Jakkolwiek ENIAC była maszyną elektroniczną, nie uzyskano w niej własności urządzeń elektronicznych, a raczej przeniesiono na grunt elektroniki idee konstrukcji mechanicznej. Matematyk amerykański J. von Neumann wykazał, że elektronika daje zupełnie nowe możliwości konstruowania maszyn matematycznych; opracował szczegółowe zasady pracy elektronicznych maszyn matematycznych i zaczął swoje idee realizować praktycznie. Pierwszą taką maszynę zbudował w Wielkiej Brytanii M. V. Wilkes w 1949 r. Maszyna ta, zwana EDSAC, była znacznie mniejsza od ENIAC, mając przy tym o wiele większe możliwości matematyczne. Obecnie maszyny wykonują miliony operacji matematycznych na sekundę, jednak ich ogólna koncepcja nie wykracza w zasadzie poza ramy nakreślone przez von Neumanna. Tak duże szybkości liczenia uzyskano dzięki szybkiemu rozwojowi elektroniki. Istnieje pogląd, że dalszy rozwój maszyn matematycznych uzależniony jest od nowych koncepcji, a nie od ich dalszego doskonalenia technicznego.

2.4. Rozwój maszyn cyfrowych w Polsce

Pierwsze prace w Polsce nad maszynami matematycznymi rozpoczęły się w roku 1948, kiedy to powstała Grupa Aparatów Matematycznych w ówczesnym Państwowym Instytucie Matematycznym. Prace tej grupy obejmowały początkowo maszyny zarówno cyfrowe jak i analogowe. Pierwsze polskie przedsięwzięcie budowy maszyny cyfrowej, kierowane przez R. Marczyńskiego, dotyczyło maszyny EMAL-1, lecz ogromne wówczas trudności techniczne nie pozwoliły na jej pomyślne zakończenie. Pierwszą zbudowaną maszyną elektroniczną był analogowy Analizator Równań Różniczkowych ARR.

Pierwszą zakończoną elektroniczną maszyną cyfrową w kraju był historyczny już dzisiaj komputer XYZ, zbudowany w roku 1958 przez zespół pod kierunkiem L. Łukaszewicza. Komputer XYZ został następnie udoskonalony i pod nazwą ZAM-2 produkowany w niewielkiej serii, począwszy od 1961 roku, w Instytucie Maszyn Matematycznych w Warszawie.

O ile maszyna EMAL-1 miała być wzorowana na angielskim EDSAK-u, o tyle maszyna XYZ — nazywana też czasem ZAM-1 — była wzorowana na amerykańskiej maszynie IBM-701. Kod rozkazowy maszyny XYZ był zbliżony do kodu rozkazowego IBM-701, produkowanej seryjnie od roku 1953 do 1957. Nie była jednak maszyną równoległą jak IBM-701, lecz tylko szeregową. Wzorowanie polegało jedynie na koncepcji organizacyjnej, doborze odpowiednich operacji i sposobie przedstawiania liczb. Wszelkie natomiast rozwiązania konstrukcyjne były oryginalnymi polskimi rozwiązaniami. Maszyna XYZ potrafiła dodać w ciągu jednej sekundy około 1000 liczb.

Następna z polskich maszyn automatycznie liczących, to EMAL-2, potomek pechowego EMAL-1 — ale tylko z nazwy. Maszyna EMAL-2 była bowiem zupełnie odmiennej konstrukcji i organizacji.

Na wyróżnienie zasługuje również maszyna UMC minus 2, zbudowana na Politechnice Warszawskiej w roku 1960 według projektu logicznego Z. Pawlaka. Maszyna ta stała się następnie pierwszą krajową maszyną produkowaną pod nazwą UMC-1 przez Wrocławskie Zakłady Elektroniczne ELWRO.

Pierwszą krajową maszyną opracowaną w 1962 roku przez Biuro Konstrukcyjne Zakładów ELWRO i produkowaną przez te Zakłady była ODRA 1003, w której po raz pierwszy zastosowano u nas tranzystory oraz automatyczny zmienny przecinek. Zaś w ulepszonej jej wersji zastosowano oprócz pamięci bębnowej również pamięć ferrytową. Maszyna ODRA 1003 jest również ciekawa z punktu widzenia jej organizacji logicznej dobrze dostosowanej do pamięci bębnowej (o pojemności 8192 słów 39-bitowych). Przyjęto między innymi w tej maszynie system „jeden plus jeden adresowy", co oznacza, że kolejność pobierania i wykonania rozkazów nie jest s e k w e n c y j n a (tzw. rozkazy nie są pobierane w kolejności ich występowania w pamięci), lecz d o w o l n a. Dowolność oznacza to, że po wykonaniu pewnego rozkazu ciągu można przejść do wykonania dowolnego (a więc niekoniecznie następnego) rozkazu ciągu. Każdy rozkaz (39-bitowy) tej maszyny zawiera dwa adresy: jeden do określania adresu argumentu z pamięci, zaś drugi określa adres komórki, w której znajduje się kolejny rozkaz do wykonania. Zatem w maszynie ODRA 1003 zbędny jest konwencjonalny licznik rozkazów. System „jeden plus jeden adresowy" pozwala na optymalną organizację rozmieszczenia programu na pamięci bębnowej, minimalizując w ten sposób tzw. czas oczekiwania (związany z obrotem bębna).

Równolegle z EMC ODRA 1013 Zakłady ELWRO rozpoczęły od 1967 roku produkcję elektronicznego kalkulatora ODRA 1103, przeznaczonego do współpracy z maszynami analitycznymi (np. z reproducerem i tabulatorem). Kalkulator ODRA 1103 jest dziesiątkową, szeregowo-równoległą, jednoadresową maszyną cyfrową, wyposażoną w pamięć ferrytową i pamięć bębnową.

Kolejnym modelem maszyn produkowanych w ELWRO jest maszyna cyfrowa ODRA 1204, której produkcję seryjną rozpoczęto w 1968 r. Maszyna ODRA 1204 jest przeznaczona głównie do obliczeń naukowo-technicznych, jak również do sterowania procesami produkcyjnymi w czasie rzeczywistym. Pamięć operacyjna maszyny jest pamięcią ferrytową o pojemności 16 384 lub 32 768 słów 24-bitowych. Czas wykonywania typowych operacji w maszynie cyfrowej ODRA 1204 wynosi:

— dodawanie i odejmowanie w stałym przecinku — 16 μs
— dodawanie i odejmowanie w zmiennym przecinku — 145 μs
— dzielenie w stałym przecinku — 190 μs
— dzielenie w zmiennym przecinku — 1050 μs

ODRA 1204 jest pierwszą w Polsce maszyną mikroprogramowaną, dzięki czemu część centralna maszyny jest mała, mimo że lista rozkazów maszyny jest rozbudowana.

Maszyna ODRA 1204 została wyprodukowana w serii blisko 200 sztuk.

W latach 1965/66 w Instytucie Maszyn Matematycznych opracowano maszynę do przetwarzania danych ZAM-41, która została wyprodukowana następnie w ilości 18 sztuk.

W latach 1968/69 Zakłady ELWRO rozpoczęły produkcję maszyn systemu ODRA 1300 (wzorowanego na systemie ICL 1900), którego pierwszym przedstawicielem była maszyna ODRA 1304 (jeszcze tranzystorowa).

Natomiast w 1972 roku rozpoczęto seryjną produkcję nowoczesnych maszyn (zbudowanych na układach scalonych) ODRA 1325 i ODRA 1305, zaś w roku 1974 wykonano i uruchomiono serię prototypową pierwszej polskiej Elektronicznej Maszyny Cyfrowej Jednolitego Systemu R-32. Maszyna R-32 po niezbędnych badaniach i poprawkach weszła do seryjnej produkcji w roku 1975 (patrz również punkt 5.7).

Krótki okres produkcji komputerów w ELWRO (od 1963 r.) to ogromny skok nie tylko ilościowy, ale przede wszystkim jakościowy. Na przykład maszyna cyfrowa R-32 wykonuje ponad 4000 razy więcej dodawań na sekundę niż pierwsza produkcyjna maszyna UMC-1 (nie mówiąc o łącznej wydajności m.c. R-32, w której wszystkie zespoły funkcjonalne pracują jednocześnie). Natomiast koszt wykonania jednego miliona dodawań zmniejszył się z 500 zł do 0,0044 zł, czyli ponad 115 000 razy!

Tabela 2.0.1 pokazuje wzrost szybkości i ceny jednostek centralnych seryjnie produkowanych w ELWRO.

Oczywiście spadek kosztów wykonania określonej liczby operacji nastąpił nie tylko dzięki ogromnemu wzrostowi szybkości w minionym okresie, lecz również dzięki zwiększeniu niezawodności (z 8 na 23,5 godzin pracy efektywnej na dobę) oraz potanieniu elementów elektronicznych i zastosowaniu nowoczesnych technologii wytwarzania i procedur kontroli. Z powyższej tabeli widać, że najtańszą maszyną jest maszyna najdroższa, czyli R-32. Oczywiście inną sprawą jest wykorzystanie tak ogromnej dziennej mocy obliczeniowej tej maszyny.

W minionym okresie nastąpił również znaczny wzrost globalnej liczby komputerów produkcji ELWRO (tab. 2.0.2).

Tabela 2.0.1

Komputer (początek seryjnej produkcji)	Szybkość dodawań na sekundę	Wzrost szybkości	Średnia cena jedn. centr. w zł	Koszt mln dodawań (na 4K słów pamięci oper.) w zł
UMC-1 (1963)	100	1	3 400 000	500
ODRA 1003 (1964)	500	5	2 500 000	40
ODRA 1013 (1966)	1 000	10	3 000 000	24
ODRA 1103 (1967)	5 000	50	2 900 000	6
ODRA 1204/16K/ (1968)	60 000	600	2 700 000	0,08
ODRA 1304/32K/ (1970)	50 000	500	4 400 000	0,078
ODRA 1325/16K/ (1972)	280 000	2 800	2 100 000	0,0088
ODRA 1305/64K/ (1972)	370 000	3 700	8 400 000	0,0047
R-32 /256Kb/ (1975)	400 000	4 000	10 000 000	0,0044

Tabela 2.0.2

Rok	Ilość wyprodukowanych komputerów		Procentowy udział wartości produkcji sprzętu informatyki
	w danym roku	ogółem	
1960	1 (ODRA 1001)	1	0
1961	1 (ODRA 1002)	2	0
1962	2(ODRA 1003+UMC)	4	1,5
1963	16 (UMC+ODRA 1003)	20	18,4
1964	18 (UMC+ODRA 1003)	38	18,7
1965	32 (ODRA 1003)	70	23,0
1966	45 (ODRA 1013)	115	28,5
1967	70 (ODRA 1013+1103)	185	35,8
1968	104 (1103+1204+ELWAT)	289	38,0
1969	77 (1204+1103+ELWAT)	366	48,7
1970	60 (1204+1304)	426	56,9
1971	58 (1204+1304)	484	53,7
1972	76 (1304+1325)	560	61,4
1973	88 (1325+1305+R−32)	648	70,2
1974	105 (1325+1305+R−32)	753	74,3
1975	100 (1325+1305+R−32)	853	68,6

Uwaga. Zmniejszenie procentowego udziału sprzętu informatyki w 1975 r. jest spowodowane skokowym wzrostem ilości kalkulatorów (z 10 000 sztuk w roku 1974 do 47 000 w roku 1975)

2.5. Klasyfikacja techniczna środków liczących

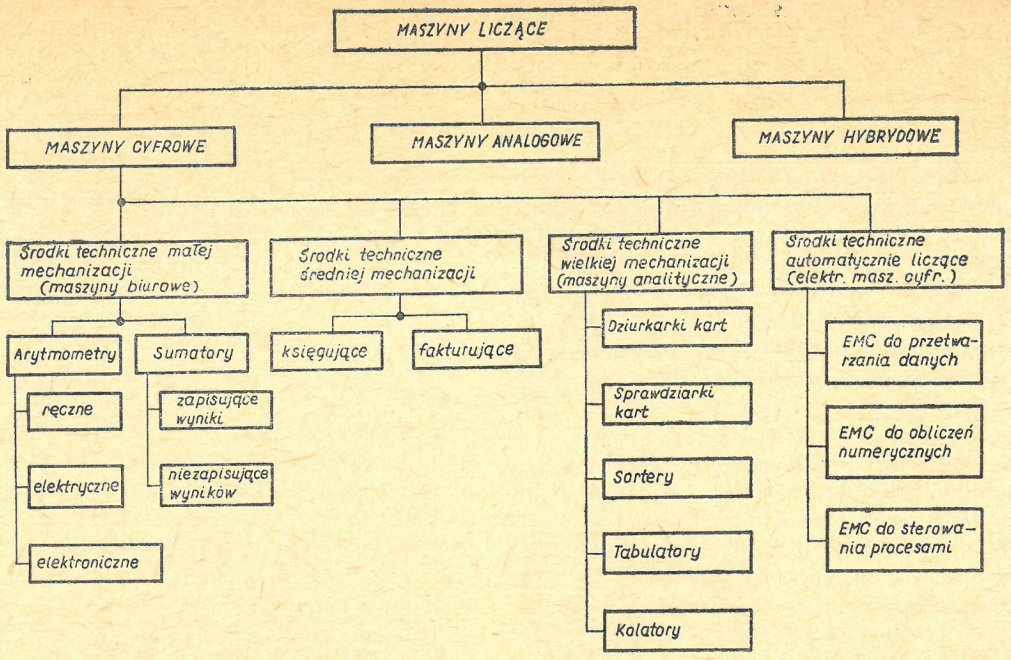
Techniczne środki liczące — tj. maszyny liczące — można podzielić na różne umowne grupy i podgrupy, stosując różne kryteria klasyfikacji. Na rys. 2.3 pokazano jeden z podziałów [5], [13]. Na najwyższym poziomie tej klasyfikacji podzielono środki obliczeniowe według rodzaju techniki obliczeniowej. W tym poziomie wyróżniono:

— maszyny oparte o technikę cyfrową,
— maszyny oparte o technikę analogową,
— maszyny hybrydowe, w których wykorzystuje się zarówno technikę cyfrową jak i analogową.

Na drugim poziomie umieszczono urządzenia liczące pogrupowane według kryterium stopnia mechanizacji lub automatyzacji obliczeń. W tym poziomie wyróżniono środki techniczne:

— małej mechanizacji,
— średniej mechanizacji,
— wielkiej mechanizacji, czyli półautomatyzacji,
— automatycznie liczące (tzn. programowane).

Jako dalsze kryterium przyjmuje się wykonywaną funkcję, czyli zalicza się do tej samej grupy urządzenia wykonujące te same funkcje. Urządzenia małej mechanizacji pozwalają mechanizować tylko pojedyncze czynności rachunkowe, np.



Rys. 2.3. Klasyfikacja technicznych środków liczących

dodawanie dwóch liczb zostaje zamienione naciśnięciem odpowiedniego klawisza arytmometru, przy czym liczby (argumenty operacji) nastawia się ręcznie w arytmometrze. Wynik operacji wykonanej w arytmometrze może być sygnalizowany w określonym rejestrze lub wydrukowany na pasku papieru.

Często w pracy zachodzi potrzeba wypełnienia różnych zestawień tablicowych, w których liczby sumowane są zarówno pionowo w kolumnach, jak i poziomo w wierszach. Ponadto zestawienia opisuje się różnymi tekstami (literowo-cyfrowymi).

Maszyny do księgowania. Są to maszyny dwudziałaniowe, wyposażone w kilka do kilkudziesięciu liczników, klawiaturę literowo-cyfrową o układzie jak w maszynie do pisania do drukowania dowolnych tekstów, klawiaturę cyfrową do wprowadzania danych liczbowych oraz klawiaturę sterującą do ręcznego sterowania pracą liczników.

Liczniki dzieli się ze względu na wykonywane funkcje na tzw. saldujące, rozróżniające sumy dodatnie i ujemne, saldujące kontrolne oraz zbiorcze i rejestrujące, służące do sumowania w wierszach i przechowywania danych.

Ponieważ sterowanie pracą maszyny księgującej jest skomplikowane, wprowadza się — oprócz sterowania ręcznego za pomocą klawiatury — możliwość sterowania automatycznego za pomocą tablicy programowej. Tablica ta pozwala nastawić pewien program działania maszyny, który jest następnie wykonywany automatycznie na żądanie operatora. Jest to szczególnie pomocne przy wielokrotnym wypełnianiu identycznych zestawień.

Maszyny do fakturowania. Jeśli w wypełnionym zestawieniu — przy zastosowaniu maszyn księgujących — występowały operacje mnożenia (które mają miejsce w różnych fakturach) oprócz maszyny do księgowania wymagało to stosowania arytmometru. Połączenie funkcji obu tych maszyn w jednym urządzeniu dało maszynę do fakturowania. Oprócz wszystkich możliwości maszyny do księgowania ma ona możliwość wykonywania mnożenia dzięki wyposażeniu w rejestry mnożnej, mnożnika i iloczynu. Wynik mnożenia może być wypisany w dowolnym miejscu zestawienia na formularzu lub przeniesiony do jednego z wewnętrznych liczników.

Maszyny średniej mechanizacji są produkowane jako mechaniczne z napędem elektrycznym bądź elektroniczne. W maszynach elektronicznych układy liczące są realizowane na tranzystorach lub na układach scalonych, zaś rejestry służące do przechowywania wprowadzanych liczb i sum częściowych są budowane na tranzystorach, na układach scalonych, na liniach opóźniających bądź na rdzeniach magnetycznych. W tej samej pamięci (rejestrach) przechowywany jest program działania maszyny (pamięć spełnia tu rolę również tablicy programowej), wprowadzony tam za pomocą klawiatury sterującej bądź literowo-cyfrowej.

Tak zbudowana maszyna do księgowania, jeśli dodatkowo pozwala wprowadzać dane automatyczne (np. z taśmy perforowanej lub kart magnetycznych) i zapisywać wyniki obliczeń, ma wszystkie cechy małej, specjalizowanej maszyny cyfrowej, mimo że wywodzi się z maszyn zaliczanych do średniej mechanizacji.

Maszyny analityczne. Do tej grupy zalicza się urządzenia służące do przygotowywania i przetwarzania danych przedstawionych w specyficznej postaci, tj. w postaci kart dziurkowanych.

Wprowadzenie kart dziurkowanych jako formy przedstawiania danych umożliwiło zautomatyzowanie ich wprowadzania do maszyn, eliminując konieczność wprowadzania ręcznego, które było szczególnie uciążliwe przy wielokrotnym opracowywaniu dużej ilości tych samych danych.

Maszyny analityczne umożliwiają w procesie przetwarzania danych mechanizację dużych grup pracochłonnych czynności, jak: sortowanie, grupowanie, obliczenia arytmetyczne, sporządzanie (drukowanie) zestawień. Umożliwiają one również dziurkowanie na kartach wyników pośrednich i końcowych do dalszego wykorzystania. Do tej

grupy zalicza się również urządzenia służące do przygotowywania danych w postaci kart.

Ze względu na format kart dziurkowanych urządzenia kartowe dzielą się przeważnie na pracujące za pomocą kart 80-kolumnowych (system Holleritha), o elektrycznym odczycie danych z kart i — pracujące za pomocą kart 90-kolumnowych (system Powersa), o mechanicznym odczycie danych z kart.

Z punktu widzenia funkcji maszyn analitycznych podzielić je można na następujące grupy:
— do przygotowywania kart dziurkowanych (dziurkarki, sprawdziarki i opisywacze oraz dziurkarki kart sumarycznych),
— do reprodukcji kart (reproducery),
— do porządkowania kart (sortery i kolektory),
— do obliczeń arytmetycznych i zestawiania wyników (tabulatory i kalkulatory).

Elektroniczne maszyny cyfrowe. Elektroniczne maszyny cyfrowe stanowią najbardziej zaawansowaną grupę wśród maszyn liczących (wiele innych rodzajów maszyn liczących rozwija się obecnie tylko dzięki temu, że spełniają one dla elektronicznych maszyn cyfrowych role urządzeń pomocniczych, służących do gromadzenia i przygotowywania danych). Głównymi cechami funkcjonalnymi elektronicznych maszyn cyfrowych są:
— zdolności do zapamiętywania praktycznie dowolnej ilości danych początkowych na dowolny okres czasu, co eliminuje konieczność wielokrotnego wprowadzania tych samych danych; dzięki temu stało się możliwe tworzenie systemów informacyjnych, których zadaniem, na równi z przeprowadzaniem pewnych obliczeń, jest m.in. przechowywanie zbioru danych, ich aktualizacja i odszukiwanie na żądanie użytkownika;
— możliwości zapamiętywania programu działania w tej samej postaci co dane i w tej samej pamięci, dzięki czemu jeden program może być przetwarzany pod kontrolą innego programu, a w szczególnym przypadku generowany przez inny program; umożliwiło to automatyzację programowania, dzięki której wąskie gardło, jakim bardzo szybko stało się programowanie maszyn cyfrowych, zostało przynajmniej częściowo pokonane;
— zdolności akceptowania danych i wydawania wyników w różnych postaciach i z bardzo dużą szybkością (na taśmie dziurkowanej, kartach, w postaci dokumentów, wykresów itp.).

2.6. Arytmetyka maszyn cyfrowych

W codziennej praktyce wykonujemy różne działania arytmetyczne na liczbach przedstawionych w dobrze nam znanym układzie dziesiętnym. Niemniej dziesiętny sposób przedstawiania liczb, tj. sposób w którym wyróżnia się dziesięć różnych znaków czy też cyfr, nie jest łatwy do realizacji w maszynach cyfrowych. Stosuje się w nich zatem inny sposób liczenia, opierający się o dwójkowy system przedstawiania liczb, w którym to systemie wyróżnia się dwie tylko cyfry, tj. zero (0) i jeden (1). Dlatego przed wykonywaniem działań na danych liczbowych zamienia się je z układu dziesiętnego na przyjęty w maszynie układ liczenia, tj. na układ dwójkowy lub na układ dziesiętny dwójkowo-kodowany.

Stosowane systemy liczenia dzielą się na systemy p o z y c y j n e i na n i e p o z y c y j n e. Niepozycyjnym systemem liczenia nazywa się taki system, w którym wartość danej cyfry nie zależy od jej położenia (pozycji) w ciągu cyfr przedstawiających liczbę. Dobrze nam znanym niepozycyjnym systemem liczenia jest tzw. rzymski system liczenia, w którym dla zapisu liczb stosuje się litery alfabetu łacińskiego.

W systemie rzymskim poszczególnym literom przyporządkowano następujące wartości:
I — jeden (1), V — pięć (5), X — dziesięć (10),
L — pięćdziesiąt (50), C — sto (100), D — pięćset (500),
M — tysiąc (1000), $\overline{\text{II}}$ — dwa tysiące (2000), $\overline{\text{III}}$ — trzy tysiące (3000), $\overline{\text{V}}$ — pięć tysięcy (5000), $\overline{\text{X}}$ — dziesięć tysięcy (10 000), $\overline{\text{XX}}$— dwadzieścia tysięcy (20 000), $\overline{\text{C}}$, — sto tysięcy (100 000), $\overline{\text{CCC}}$ — trzysta tysięcy (300 000), $\overline{\text{IXI}}$ — milion (1 000 000).

Dla przedstawienia na przykład liczby 1875 piszemy ciąg liter: MDCCCLXXV=1000+500+ +100+100+100+50+10+10+5, gdzie, jak widać, wartości poszczególnych liter, występujących w kolejności malejących wartości, zostają dodane do siebie.

W celu skrócenia zapisu niektórych liczb przyjmuje się dodatkową zasadę, która zakłada: jeśli litera przedstawiająca mniejszą liczbę zostanie umieszczona na lewo od litery przedstawiającej większą liczbę, to należy tę liczbę mniejszą odjąć od liczby większej (a nie dodawać!).
Na przykład: IX=10−1=9, XL=50−10=40, XC=100−10=90, ale zapis CX=100+10=110. Data 1975 może mieć MCMLXXV. Czyli każda dana litera przedstawia zawsze tę samą wartość, niezależnie od jej pozycji w ciągu.

Wykonywanie działań arytmetycznych na wielkościach przedstawionych w niepozycyjnym systemie liczenia jest sprawą trudną i złożoną. Dlatego też stosuje się powszechnie w praktyce i w maszynach cyfrowych pozycyjne systemy liczenia. Natomiast zastosowania niepozycyjnych systemów liczenia są ograniczone (np. w numeracji miesięcy, wieków, rozdziałów książki itp.).

W pozycyjnych systemach liczenia wartość cyfry zależy od jej położenia (pozycji) w ciągu cyfr oznaczających liczbę. Na przykład w ciągu 555 kolejne cyfry oznaczają: liczbę setek, liczbę dziesiątek i liczbę jednostek.

P o d s t a w ą systemu (układu) liczenia nazywa się liczbę równą ilości różnych znaków (cyfr) stosowanych w zapisie liczb w danym układzie liczenia. Podstawa układu liczenia, którą oznaczymy przez p, określa ile razy powiększa (pomniejsza) się wartość tej samej cyfry wpisanej w sąsiedniej pozycji.

Tabela 2.1 pokazuje różne systemy liczenia, tj. mające różne podstawy układu liczenia (od 2 do 16).

Wartość każdej cyfry pozycyjnego układu liczenia, np. układu dziesiętnego, zależy od jej pozycji, np. pierwsza z lewej cyfra 7 w liczbie 7527 oznacza siedem tysięcy, zaś ostatnia cyfra oznacza siedem jednostek. Liczba 7527,34 układu dziesiętnego może być rozwinięta w postaci:

$$7527{,}34 = 7 \cdot 10^3 + 5 \cdot 10^2 + 2 \cdot 10^1 + 7 \cdot 10^0 + 3 \cdot 10^{-1} + 4 \cdot 10^{-2}$$

Natomiast rozwinięcie liczby 57A3EF2 układu szesnastkowego układu liczenia ma postać:

$$57A3EF2 = 5 \cdot 16^6 + 7 \cdot 16^5 + A \cdot 16^4 + 3 \cdot 16^3 + E \cdot 16^2 + F \cdot 16^1 + 2 \cdot 16^0 = 5 \cdot 16^6 + 7 \cdot 16^5 + 10 \cdot 16^4 + 3 \cdot 16^3 + 14 \cdot 16^2 + 15 \cdot 16^1 + 2 \cdot 16^0$$

Analogicznie można zapisać rozwinięcie danej liczby x w układzie o postaci p i mającej ciąg $n+m+1$ — cyfr:

$c_n c_{n-1} \ldots c_1 c_0, c_{-1} c_{-2} \ldots c_{-m}$, czyli

$$x = c_n \cdot p^n + c_{n-1} \cdot p^{n-1} + \ldots + c_1 \cdot p^1 + c_0 \cdot p^0 + c_{-1} \cdot p^{-1} + c_{-2} \cdot p^{-2} + \ldots + c_{-m} \cdot p^{-m}$$

gdzie w miejscu c_i (dla $i = n, n-1, \ldots, 1, 0, -1, -2, \ldots, -m$) może wystąpić dowolna z przyjętych p-cyfr układu liczenia o podstawie p.

W naszym przykładzie liczba ma $n+1$ cyfr przed przecinkiem i m cyfr po przecinku.

Tabela 2.1

System liczenia	Podstawa systemu (p)	Cyfry stosowane w systemie
Dwójkowy	2	0,1
Trójkowy	3	0,1,2
Czwórkowy	4	0,1,2,3
Piątkowy	5	0, . . . , 4
Szóstkowy	6	0, . . . , 5
Siódemkowy	7	0, . . . , 6
Ósemkowy	8	0, . . . , 7
Dziewiątkowy	9	0, . . . , 8
Dziesiątkowy	10	0, . . . , 9
Jedenastkowy	11	0, . . . , 9, A
Dwunastkowy	12	0, . . . , 9, A, B
Trzynastkowy	13	0, . . . , 9, A, B, C
Czternastkowy	14	0, . . . , 9, A, B, C, D
Piętnastkowy	15	0, . . . , 9, A, B, C, D, E
Szesnastkowy	16	0, . . . , 9, A, B, C, D, E, F

Znaki pisarskie A, B, C, D, E, F oznaczają dalsze cyfry systemów liczenia o podstawie od 11 do 16 (którym przyporządkowano wartości 10, 11, 12, 13, 14, 15)

2.6.1. Liczby dwójkowe stałoprzecinkowe

Podstawą dwójkowego układu liczenia jest liczba 2 i wyróżnia się w nim tylko dwie cyfry o symbolach 0 i 1. Cyfrom tym przyporządkowano wartości odpowiednio zera i jedynki. Zatem każda liczba w układzie dwójkowym (zwanym też binarnym) jest zapisywana za pomocą ciągu zer i jedynek. Na przykład liczba 6 będzie zapisana w tym układzie w postaci: 110 $(= 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)$.

W celu oznaczenia liczb w różnych układach liczenia przyjmijmy oznaczenie $(x)_p$, gdzie x jest postacią zapisu danej liczby w układzie o podstawie p. Np. zapisy $(1110)_2$, $(112)_3$, $(32)_4$, $(14)_{10}$, $(E)_{16}$ oznaczają tę samą liczbę — tj. liczbę czternaście — w układach o podstawie 2, 3, 4, 10 i 16. Czyli zachodzą równości:

$$(1110)_2 = (112)_3 = (32)_4 = (14)_{10} = (E)_{16}$$

Tabela 2.2 podaje rozwinięcie liczb od 0 do 20 w różnych układach liczenia, tj. w układzie o podstawie 2, 3, 8, 10, 12 i 16.

Tabela 2.2

Podstawa układu	2	3	8	10	12	16
Rozwinięcia liczb	0	0	0	0	0	0
	1	1	1	1	1	1
	10	2	2	2	2	2
	11	10	3	3	3	3
	100	11	4	4	4	4
	101	12	5	5	5	5
	110	20	6	6	6	6
	111	21	7	7	7	7
	1000	22	10	8	8	8
	1001	100	11	9	9	9
	1010	101	12	10	A	A
	1011	102	13	11	B	B
	1100	110	14	12	10	C
	1101	111	15	13	11	D
	1110	112	16	14	12	E
	1111	120	17	15	13	F
	10000	121	20	16	14	10
	10001	122	21	17	15	11
	10010	200	22	18	16	12
	10011	201	23	19	17	13
	10100	202	24	20	18	14

Dotychczas podane przykłady przedstawienia liczb w różnych układach dotyczyły liczby nieujemnej. Dlatego nie było potrzeby wyróżnienia znaku liczby. W praktyce oczywiście wykonujemy działania na względnych wielkościach liczbowych i w tym celu w ciągu cyfr przedstawiających daną liczbę musi znajdować się również informacja o znaku (+ lub −) danej liczby. W układach dwójkowych wprowadza się na ogół dodatkową pozycję, na lewo od cyfr znaczących, do określania znaku liczb. Jeśli na przykład liczba dwójkowa ma N-1 cyfr znaczących, ponumerowanych od 1 do N-1, to wprowadza się pozycję o numerze 0, w której wpisuje się informację o znaku liczby (np. 0 dla oznaczenia liczb nieujemnych i 1 dla oznaczenia liczb ujemnych). Wówczas liczba ma N pozycji.

Przykład ogólny: $c_0 c_1 c_2 \ldots c_{N-1}$ (gdzie c_0 jest pozycją znakową).

Przykłady liczbowe: $(01101)_2 = (+13)_{10}$, $(11101)_2 = (-13)_{10}$.

W strukturach maszyn cyfrowych stosuje się najczęściej jeden z trzech dwójkowych systemów przedstawiania znaku liczby:

— system modułu i znaku,
— system negacji (uzupełnienie do jedynki),
— system uzupełnieniowy (uzupełnienie do dwójki).

Ponadto w maszynach cyfrowych, zależnie od sposobu przedstawienia liczb w słowie maszynowym, wyróżnia się trzy systemy przedstawiania liczb:

— system dwójkowy stałoprzecinkowy,
— system dwójkowy zmiennoprzecinkowy,
— system dziesiętny kodowany dwójkowo.

Przedstawienie liczb w maszynie w systemie dwójkowym stałoprzecinkowym polega na tym, że liczby przedstawione są w systemie dwójkowym, a ponadto każda liczba zapisywana jest w oddzielnym słowie (lub półsłowie) maszynowym, ma więc dla danej maszyny zawsze tą samą długość oraz ustalone położenie przecinka.

Przecinek liczby nie jest reprezentowany fi-

zycznie w słowie maszynowym, lecz jego położenie umownie ustala się pomiędzy dwoma kolejnymi bitami (np. zerowym i pierwszym). Najczęściej liczby traktowane są jako ułamkowe (wówczas przecinek umieszczony jest bezpośrednio przed najbardziej znaczącą cyfrą) lub całkowite (przecinek bezpośrednio po najmniej znaczącej cyfrze). W dalszym ciągu pracy liczby stałoprzecinkowe będą traktowane jako całkowite.

Znak liczby stałoprzecinkowej jest zapisywany w pozycji 0 słowa maszynowego. Jeśli w pozycji tej zapisany jest bit 0, oznacza to, że liczba jest dodatnia. Pozycje od 1 do N-1 zawierają wartość bezwzględną liczby; pozycja 1 jest najbardziej znaczącą pozycją liczby. Bit 1 na pozycji 0 oznacza, że liczba jest ujemna.

Poniżej opisano różne systemy przedstawiania liczb ujemnych.

A. System modułu i znaku. Pozycja 0 zawiera zawsze bit znaku, a pozycje od 1 do N-1 słowa (ciągu) zawierają wartość bezwzględną liczby; jeśli liczba nie jest ujemna, to w pozycji znakowej (zerowej) wpisuje się 0, jeśli zaś liczba jest ujemna, to w pozycji znakowej wpisuje się 1, np.

$(0101)_2=(+5)$; $(1101)_2=(-5)_{10}$;
$(0000)_2=(+0)_{10}$; $(1111)_2=(-7)_{10}$.

B. System negacji. Liczby dodatnie (i zero) mają 0 w pozycji znakowej, a bezwzględną wartość liczby zapisuje się w pozycjach od 1 do N-1. Natomiast liczby ujemne mają 1 w pozycji znakowej, a na pozycjach od 1 do N-1 słowa (ciągu) zapisuje się tzw. uzupełnienie jedynkowe wartości bezwzględnej liczby. Uzupełnieniem jedynkowym (X) danej liczby X nazywamy liczbę, która powstaje przez uzupełnienie do 1 każdej cyfry liczby X. Uzupełnienie to sprowadza się do zastąpienia każdej cyfry 0 cyfrą 1 i odwrotnie, czyli do wykonania na niej operacji negacji.

Przykłady: $(0101)_2=(+5)_{10}$; $(1010)_2=(-5)_{10}$;
$(0000)_2=(+0)_{10}$; $(1111)_2=(-0)_{10}$.

C. System uzupełnieniowy. Jeśli pozycja 0 zawiera bit 1, to na pozycjach od 1 do N-1 słowa zapisane jest tzw. uzupełnienie dwójkowe wartości bezwzględnej liczby. Uzupełnieniem dwójkowym (X^-) danej liczby n-cyfrowej (X) nazywamy różnicę pomiędzy liczbą 2^n i liczbą X. Zatem uzupełnienie dwójkowe każdej liczby jest o 1 większe od jej uzupełnienia jedynkowego. Właściwość tę wykorzystuje się do tworzenia uzupełnienia dwójkowego.

Przykłady: $(0101)_2=(+5)_{10}$; $(1011)_2=(-5)_{10}$;
$(0000)_2=(-0)$; $(1111)_2=(-1)_{10}$.

System uzupełnieniowy można również interpretować jako system liczbowy, w którym liczby dodatnie i ujemne zapisywane są jednolicie, bez wyróżniania bitu znaku, na pozycjach od 0 do N-1 słowa maszynowego.

Wybrany system przedstawiania liczb rzutuje przede wszystkim na algorytmy wykonywania działań w maszynie. Tabela 2.3 pokazuje algorytm dodawania i odejmowania liczb bez znaku. W tabeli tej litera a_i oznacza i-tą cyfrę jednego składnika, b_i oznacza i-tą cyfrę drugiego składnika, p_i oznacza przeniesienie (przy dodawaniu) lub pożyczkę (przy odejmowaniu) z mniej znaczącej pozycji $i+1$ na pozycję i-tą. Natomiast p_{i-1} oznacza analogiczne przeniesienie (pożyczkę) z pozycji i na pozycję i-1. Opisany tą tabelką algorytm odejmowania daje w rezultacie wartość bezwzględną różnicy, gdy odjemna jest większa od odjemnika; w przeciwnym przypadku otrzymuje się uzupełnienie dwójkowe tej wartości bezwzględnej.

Tabela 2.3

a_i	b_i	p_i	+		−	
			$(a+b)_i$	p_{i-1}	$(a-b)_i$	p_{i-1}
0	0	0	0	0	0	0
0	1	0	1	0	1	1
1	0	0	1	0	1	0
1	1	0	0	1	0	0
0	0	1	1	0	1	1
0	1	1	0	1	0	1
1	0	1	0	1	0	0
1	1	1	1	1	1	1

W każdym z trzech systemów przedstawiania liczb stosuje się różne algorytmy dodawania i odejmowania liczb względnych.

W systemie modułu i znaku jedną z metod jest tzw. metoda bezpośrednia. Polega ona na tym, że przed wykonaniem działania porównuje się znaki obu liczb. Jeśli liczby te należy dodać, a znaki są zgodne, wykonuje się dodawanie ich wartości bezwzględnych w sposób opisany w tabeli 2.3, a znakiem wyniku jest bit a_0. Gdy znaki są przeciwne, wykonuje się odejmowanie wartości bezwzględnych również zgodnie z tabelą 2.3 i bada się pożyczkę z pozycji 1. Jeśli pożyczka ta ma wartość 0, otrzymana różnica jest prawidłowa, a znakiem wyniku jest bit a_0; w przeciwnym przypadku tworzy się uzupełnienie dwójkowe otrzymanej różnicy, a znakiem wyniku jest bit b_0.

Jeżeli liczby należy odjąć, porównuje się bit a_0 z negacją bitu b_0 i postępuje się jak wyżej.

Inny sposób realizacji tej metody polega na tym, że odejmowanie wartości bezwzględnych opisane tabelą 2.3 zastępuje się dodawaniem do liczby a uzupełnienia dwójkowego liczby b. Jeżeli przy tej operacji przeniesienie p_0 ma wartość 1, wynik jest poprawny; w przeciwnym przypadku należy dokonać korekty wyniku polegającej na utworzeniu jego uzupełnienia jedynkowego. Obie te metody ilustrują poniższe przykłady.

Przykład. Odejmowanie liczb metodą bezpośrednią:

(a) $|a|=11000110$ $\quad |b|=01101100$

Etap 1: tworzenie różnicy według tabeli 2.3

$$\begin{array}{rl} 11000110 & |a| \\ -\ 01101100 & |b| \\ \hline 0\ 01011010 & r \end{array}$$

Etap 2: $p_0=0$, zatem wynik różnicy $|a|-|b|=r=01011010$

(b) $|a|=01101100$ $\quad |b|=11000110$

Etap 1: tworzenie różnicy według tabeli 2.3

$$\begin{array}{rl} 01101100 & |a| \\ -\ 11000110 & |b| \\ \hline 1\ 10100110 & r \end{array}$$

Etap 2: $p_o=1$, zatem należy utworzyć $\overline{X}$

$$\begin{array}{rr} 01011001 & \overline{r} \\ + \quad 1 & \\ \hline 01011010 & r \end{array}$$

Wynik $|a|-|b|=r=01011010$

Przykład. Odejmowanie liczb metodą dodawania uzupełnienia dwójkowego:

(a) $|a|=11000110 \quad |b|=01101100$

Etap 1: tworzenie uzupełnienia dwójkowego liczby b

$$\begin{array}{rr} 10010011 & |\overline{b}| \\ + \quad 1 & \\ \hline 10010100 & |b| \end{array}$$

Etap 2: tworzenie sum $|a|$ i $|b|$ według tabeli 2.3

$$\begin{array}{rr} 11000110 & |a| \\ +10010100 & |b| \\ \hline 101011010 & S \end{array}$$

Etap 3: $p_o=1$, zatem wynik $|a|-|b|=S=$ $=01011010$

(b) $|a|=01101100 \quad |b|=11000110$

Etap 1: tworzenie uzupełnienia dwójkowego liczby b

$$\begin{array}{rr} 00111001 & \overline{b} \\ + \quad 1 & \\ \hline 00111010 & |b| \end{array}$$

Etap 2: tworzenie sumy $|a|$ i $|b|$ według tabeli 2.3

$$\begin{array}{rr} 01101100 & |a| \\ +00111010 & |b| \\ \hline 010100110 & S \end{array}$$

Etap 3: $p_o=0$, zatem należy utworzyć $\overline{S}$

$$01011001$$

Wynik: $|a|-|b|=\overline{S}=01011001$

Najprostsze algorytmy dodawania i odejmowania liczb względnych realizowane są w systemie uzupełnieniowym.

Dodawanie dwu liczb w tym systemie, bez względu na ich znaki, odbywa się zawsze zgodnie z tabelą 2.3, a ponadto jest wykonywane jednolicie na pozycji znaku i pozycjach określających wartość bezwzględną liczby.

Odejmowanie dwu liczb jest wykonywane, również bez badania znaków, przez dodanie do odjemnej uzupełnienia dwójkowego odjemnika, bowiem w tym systemie utworzenie takiego uzupełnienia jest jednoznaczne ze zmianą znaku liczby. Wynik jest zawsze poprawny: nie wymaga dodatkowych kroków korekcyjnych. Wynika to z faktu, że liczba ujemna w tym systemie jest przedstawiona zawsze w postaci uzupełnienia dwójkowego.

Działania na znakach wykonuje się analogicznie jak na innych bitach liczb dzięki temu, że pozycje te można traktować jako pozycje cyfrowe.

Zakres liczby dwójkowej. Ze skończonej długości ciągu bitów, tj. słowa maszynowego, wynika ograniczoność zakresu liczb, które mogą być przedstawione w postaci stałoprzecinkowej. Jeśli długość słowa wynosi N, to zakres liczby całkowitej x zapisanej w takim słowie wynosi:

$-2^{N-1}+1 \leqslant x \leqslant 2^{N-1}-1$ dla systemu modułu i znaku oraz systemu negacji

$-2^{N-1} \leqslant x \leqslant +2^{N-1}-1$ dla systemu uzupełnieniowego

W maszynach Jednolitego Systemu, gdzie stosuje się 32-bitowe słowo (liczba długa) i 16-bitowe tzw. półsłowo (liczba krótka), przyjęto system arytmetyki uzupełnieniowej dla stałoprzecinkowych liczb. Zatem zakres 32- i 16-bitowych całkowitych liczb stałoprzecinkowych jest następujący:

$-2^{31} \leqslant$ L. CAŁKOWITA DŁUGA $\leqslant +2^{31}-1$;
czyli od -2147483648 do $+2147483647$
oraz
$-2^{15} \leqslant$ L. CAŁKOWITA KRÓTKA $\leqslant +2^{15}-1$;
czyli od -32768 do $+32767$

Jeśli wynik dodawania lub odejmowania wychodzi poza ten zakres, fakt ten jest wykrywany i sygnalizowany. Mówimy wówczas, że wystąpił nadmiar.

Mnożenie liczb stałoprzecinkowych. Mnożenie dwu liczb stałoprzecinkowych wykonuje się najczęściej metodą wielokrotnego dodawania mnożnej (krotności mnożnej), sterowanego kolejnymi cyframi lub grupami cyfr mnożnika.

Zasada najprostszej metody mnożenia dla dwu liczb bez znaku jest następująca. Niech X oznacza mnożną, Y — mnożnik. Mnożnik można przedstawić w postaci:

$$Y=2^{N-1}\cdot y_1+2^{N-2}\cdot y_2+\ldots+2^0\cdot y_{N-1}$$

gdzie $y_1, y_2 \ldots, y_{N-1}$ oznaczają jego kolejne cyfry znaczące, zaczynając od najbardziej znaczącej. Mamy zatem:

$$X\cdot Y=X\cdot(2^{N-1}\cdot y_1+2^{N-2}y_2+\ldots+2^0y_{N-1})=$$
$$=(\ldots(0+2^{N-1}X\cdot y^{N-1})\cdot 2^{-1}+2^{N-1}X\cdot y_{N-2})\cdot 2^{-1}+$$
$$+\ldots(2^{-1}+2^{N-1}X\cdot y_1)\cdot 2^{-1}$$

Wyrażenia $i_0=0, \quad i_1=i_0(+2^{N-1}X\cdot y_{N-1})\cdot 2^{-1}$

i ogólnie $i_n=(i_{n-1}+2^{N-1}X\cdot y_{N-n})\cdot 2^{-1}$ nazywamy iloczynami częściowymi. Ponieważ iloczyn dwu liczb $(N\text{-}1)$ cyfrowych może być liczbą $2N\text{-}2$ cyfrową, wszystkie iloczyny częściowe oraz iloczyn końcowy są liczbami podwójnej długości, tj. zawartość dwu słów maszynowych traktowana jest łącznie jako jedna liczba.

Ponieważ każda cyfra mnożnika Y, może być równa 0 bądź 1, iloczyn częściowy i_n powstaje z iloczynu częściowego i_{n-1} w następujący sposób:

$$i_n=i_{n-1}\cdot 2^{-1}, \quad \text{gdy} \quad y_{N-n}=0$$
$$i_n=(i_{n-1}+2^{N-1})\cdot X\cdot 2^{-1}, \quad \text{gdy} \quad y_{N-n}=1.$$

Łatwo zauważyć, że mnożenie dowolnej liczby dwójkowej przez 2^{-1} jest równoważne przesunięciu każdej cyfry tej liczby na pozycję sąsiednią mniej znaczącą słowa i wprowadzeniu cyfry 0 na zwolnioną pozycję najbardziej znaczącą.

Wyrażenie $i_{n-1}+2^{N-1}$ X oznacza dodanie mnoż-

nej (liczby N-1 cyfrowej) do bardziej znaczącej połowy iloczynu częściowego (liczby $2N$-2 bitowej). Iloczyn częściowy i_{N-1} jest równy szukanemu iloczynowi X i Y. Zatem powyższą formułę można ująć w następujący słowny przepis:

Etap 0. Wyzeruj zawartość rejestrów (słów) maszynowych, w których tworzony będzie iloczyn (czyli przyjmuje się $i_o=0$).

Etap n. Zbadaj cyfrę y_{N-n} mnożnika (dla $n=1, 2, \ldots, N$-1). Jeśli cyfra ta jest równa 1, to dodaj mnożną do zawartości słowa zawierającego bardziej znaczącą część iloczynu częściowego. Przesuń łączną zawartość obu słów o jedną pozycję w prawo. Jeśli zaś $y_{N-n}=0$, to przesuń iloczyn częściowy o jedną pozycję w prawo.

Jeśli mnożnik zawiera same jedynki, metoda ta wymaga N-1 dodawań i N-1 przesunięć. Aby realizację mnożenia w maszynie przyśpieszyć, zamiast pojedynczych cyfr mnożnika analizuje się grupy cyfr, dodając do iloczynu częściowego krotność mnożnej określoną przez taką grupę i przesuwając go o liczbę pozycji w grupie. Sposób ten stosuje się w różnych metodach wielobitowego mnożenia.

Przykład mnożenia dwóch stałoprzecinkowych liczb dodatnich:

Mnożna $X=01011$, zaś mnożnik $Y=01101$.
Postać ogólną mnożnika można zapisać następująco: $Y = y_0\, y_1\, y_2\, y_3\, y_4$ (czyli są to liczby 5-bitowe, zatem wskaźnik N-1=4).

Kroki mnożenia:

0. Przyjmuje się, że początkowy iloczyn częściowy jest równy zeru, czyli $i_o=0$
1. Cyfra $y_4=1$, zatem należy dodać do i_o wartość mnożnej X i przesunąć wynik w prawo o jedną pozycję:
   $i_o=00000$
   $X=01011$
   suma 01011 i suma przesunięta, tj. $i_1=001011$.
2. Cyfra $y_3=0$, zatem przesuwa się w prawo dotychczasowy iloczyn o jedną pozycję:
   $i_2 = 0001011$
3. Cyfra $y_2=1$, zatem iloczyn częściowy i_3 równa się:
   $i_2=0001011$
   $X=01011$
   suma=0110111, $i_3=00110111$
4. Cyfra $y_1=1$, zatem i_4 równa się:
   $i_3=00110111$
   $X=01011$
   suma=1001111, $i_4=010001111$

Zatem końcowy iloczyn $X\cdot Y=010001111$.

W dzieleniu liczb stałoprzecinkowych dzielna występuje zwykle w postaci liczby podwójnej długości, natomiast dzielnik jest liczbą pojedynczej długości. Podobnie jak mnożenie, dzielenie wykonuje się etapami. W pierwszym etapie porównuje się bardziej znaczącą połowę dzielnej z dzielnikiem w celu sprawdzenia czy dzielenie jest wykonalne, tzn. czy nie prowadzi do nadmiaru.

W następnym etapie przesuwa się dzielną o jedną pozycję w lewo (co odpowiada pomnożeniu jej przez 2) i od bardziej znaczącej połowy odejmuje się dzielnik. Jeśli różnica jest dodatnia, traktuje się ją jako nową dzielną (resztę częściową), a najbardziej znacząca cyfra ilorazu jest jedynką. Jeśli różnica jest ujemna, reszta częściowa pozostaje bez zmiany, a cyfra ilorazu jest zerem.

Etap ten powtarza się (N-1), razy, wyznaczając (N-1) kolejnych cyfr ilorazu. Reszta częściowa (N-1) jest właściwą resztą z dzielenia.

2.6.2. System dwójkowy zmiennoprzecinkowy

Podstawową wadą systemu stałoprzecinkowego jest stosunkowo mały zakres przedstawianych liczb, co bardzo komplikuje obliczenia, w których występują równocześnie liczby bardzo małe i bardzo duże. Aby możliwe było przedstawianie z jednakową precyzją, tj. tą samą ilością cyfr znaczących, zarówno liczb bardzo dużych jak i bardzo małych, konieczne było w takich obliczeniach dokonywanie operacji przeskalowywania, tj. zmiany położenia przecinka. Ostatecznie do każdej liczby zapisanej z możliwie dużą precyzją dołączona była informacja o położeniu przecinka w tej liczbie; w ten sposób powstał system zmiennoprzecinkowy przedstawiania liczb. Arytmetyka tych liczb była początkowo realizowana programowo, później w celu zwiększenia szybkości zaczęto w maszyny wbudowywać odpowiednie układy. Ponieważ ich koszt jest jednak dość duży, nie wszystkie maszyny są w takie układy wyposażone lub odbywa się to tylko na żądanie klienta.

Dwójkowa liczba zmiennoprzecinkowa A jest przedstawiana w postaci pary liczb stałoprzecinkowych $(m_a; w_a)$, z których pierwsza określa jej część ułamkową (mantysę), a druga wykładnik. Wartość liczby A określona jest przez wyrażenie:

$$A = m_a \cdot 2^{w_a} = (m_a; w_a)$$

Liczba m_a jest najczęściej liczbą ułamkową, której wartość bezwzględna jest zerem lub spełnia nierówność:

$$+\frac{1}{2} \leqslant |m_a| < +1$$

Liczbę, której mantysa spełnia te warunki, nazywamy znormalizowaną. Liczba w_a (wykładnik), która określa położenie przecinka, jest liczbą całkowitą należącą do pewnego przedziału liczb.

Przykłady przedstawiania dwójkowych lub zmiennoprzecinkowych liczb:
Liczbę 101, 11 (gdzie przecinek dzieli część całkowitą od ułamkowej) można przedstawić w kilku postaciach:
a) $0{,}10111\cdot 2^3=(0{,}10111;+3)=(0{,}10111;+11)$
b) $0{,}0010111\cdot 2^5=(0{,}0010111;+5)=$
$=(0{,}0010111;+101)$
Pierwszy przypadek przedstawia liczbę znormalizowaną, drugi zaś liczbę nieznormalizowaną.

Powyżej zostało podane określenie dwójkowej liczby zmiennoprzecinkowej. W analogiczny sposób można zdefiniować dziesiętną liczbę zmiennoprzecinkową. Będziemy mianowicie mówili, że liczba X jest dziesiętną liczbą zmiennoprzecinkową, gdy jest ona przedstawiona w postaci pary liczb m_x i w_x takich, że:

$$X = m_x \cdot 10^{w_x}$$

gdzie liczba m_x jest dziesiętnym ułamkiem właściwym, spełniającym nierówność:

$$\frac{1}{10} \leqslant |m_x| < +1$$

Natomiast liczba w_x jest liczbą całkowitą zaopatrzoną znakiem plus (+) lub minus (−).

Zatem każdą liczbę dziesiętną napisaną w postaci normalnej można przedstawić w postaci zmiennoprzecinkowej. Na przykład liczbę 3437,125 można przedstawić w kilku postaciach:

a) $0{,}3437125 \cdot 10^4 = (0{,}3437125; +4)$;
b) $0{,}0003437125 \cdot 10^7 = (0{,}0003437125; +7)$
c) $34{,}37125 \cdot 10^2 = (34{,}37125; +2)$
d) $3437125 \cdot 10^{-3} = (3437125; -3)$

Wszystkie powyższe cztery formy przedstawiania tej samej liczby (tj. 3437,125) są oczywiście poprawnymi formami wykładniczymi. Natomiast znormalizowaną liczbą zmiennoprzecinkową jest tylko pierwsza postać, gdyż moduł jej mantysy (ułamek) należy do przedziału:

$$\left[+\frac{1}{10}, +1\right]$$

Jeśli układ liczenia ma dowolną postawę p (gdzie p jest liczbą całkowitą), to liczby zmiennoprzecinkowe X w tym układzie określa się równością $X = x \cdot p^w$, gdzie x jest ułamkiem, a w liczbą całkowitą. Zakłada się przy tym, że: $|x| < +1$. Jeśli natomiast spełniona jest nierówność: $p^{-1} \leqslant |x| < +1$, to taką liczbę nazywa się znormalizowaną.

Wykonywanie działań arytmetycznych na liczbach zmiennoprzecinkowych sprowadza się do wykonywania działań na parach liczb (tj. mantysy i wykładnika). Algorytmy działań na parach liczbowych wynikające z przyjętej definicji mają niżej podaną postać.

Dodawanie zmiennoprzecinkowe. Załóżmy, że w układzie liczenia o podstawie p chcemy dodać dwie liczby X i Y, które przedstawione są za pomocą odpowiednich par:

$$X = x \cdot p^k \quad \text{i} \quad Y = y \cdot p^l$$

Zatem suma tych liczb może być zapisana w postaci:

$$X+Y = x \cdot p^k + y \cdot p^l = [x + y \cdot p^{l-k}] \cdot p^k = \\ = [x + y \cdot p^{-(k-l)}] \cdot p^k = (s; k)$$

gdzie s oznacza mantysę sumy.

Jeśli założymy, że wykładnik k jest większy lub równy wykładnikowi l, to formułę dodawania dwóch liczb zmiennoprzecinkowych można określić następująco: wykładnik sumy dwóch liczb równa się maksymalnemu wykładnikowi z dodawanych argumentów, tj. maksimum z wielkości k i l. Natomiast mantysa sumy dwóch liczb równa się sumie mantysy liczby o większym wykładniku i mantysy liczby o mniejszym wykładniku podzielonej przez liczbę p^{k-l} (dla $k \geqslant l$).

W strukturach maszyn cyfrowych operacja dzielenia przez p^{k-l} jest równoważna przesunięciu liczby (mantysy) w prawo o k-l pozycji danego układu liczenia.

Przykład z liczbami dziesiętnymi:
$X = 325{,}36 = (0{,}32536; +3)$; $Y = -4{,}25 = (-0{,}425; +1)$
$X+Y = (0{,}32536; +3) + (-0{,}425; +1) =$
$= (0{,}32536 - 0{,}00425; +3) = (0{,}32111; +3)$

Przykład z liczbami dwójkowymi:
$X = 101{,}11 = (0{,}10111; +11)$; $Y = 1{,}011 = (0{,}1011; +1)$
$X+Y = (0{,}10111; +11) + (0{,}1011; +1) = (0{,}10111 +$
$+ 0{,}001011; +11) = (0{,}110111; +11) = 110{,}111$

Przykład z liczbami szesnastkowymi:
Niech $X = 459B{,}3E = 0{,}459B3E \cdot 16^4 = (0{,}459B3E; +4)$
oraz $Y = 7A{,}6C = 0{,}7A6C \cdot 16^2 = (0{,}7A6C; +2)$
$X+Y = (0{,}459B3E; +4) + (0{,}7A6C; +2) =$
$= (0{,}459B3E + 0{,}007A6C; +4) =$
$= (0{,}4615AA; +4) = 4615{,}AA$

Uwaga: A, B, C, D, E i F to sześć ostatnich cyfr układu szesnastkowego i przyporządkowano im następujące wartości (według notacji układu dziesiątkowego):

$$A = 10_{10},\ B = 11_{10},\ C = 12_{10},\ D = 13_{10},\\ E = 14_{10} \text{ i } F = 15_{10}.$$

Odejmowanie zmiennoprzecinkowe. Odejmowanie jest analogiczne do dodawania i ma postać:

$$X - Y = x \cdot p^k - y \cdot p^l = [x - y \cdot p^{-(k-l)}] \cdot p^k = (r; k)$$

Gdzie przez r oznaczono różnicę $x - y \cdot p^{-(k-l)}$

Mnożenie zmiennoprzecinkowe. Algorytm mnożenia liczb zmiennoprzecinkowych jest prosty i ma postać:

$$X \cdot Y = x \cdot p^k y \cdot p^l = xy \cdot p^{k+l} = (x \cdot y; k+l)$$

Zatem mantysa iloczynu dwóch liczb zmiennoprzecinkowych równa się iloczynowi mantys tych liczb, zaś wykładnik iloczynu dwóch liczb równa się sumie wykładników tych liczb.

Dzielenie zmiennoprzecinkowe. Algorytm dzielenia liczb zmiennoprzecinkowych ma postać:

$$\frac{X}{Y} = xp^k : yp^l = \frac{x}{y} \cdot p^{k-l} = \left(\frac{x}{y}; k-l\right)$$

czyli mantysa ilorazu dwóch liczb zmiennoprzecinkowych równa się ilorazowi mantys tych liczb (dzielnej i dzielnika), zaś wykładnik ilorazu równa się różnicy wykładników dzielnej i dzielnika.

2.6.3. System dziesiętny kodowany dwójkowo

W omawianych poprzednio systemach konieczna była konwersja liczby z postaci alfanumerycznej, w jakiej są one wprowadzane do maszyny na postać stało- lub zmiennoprzecinkową, na której dokonywane są obliczenia oraz w kierunku odwrotnym przy wyprowadzaniu wyników. Translacja taka jest skomplikowana oraz czasochłonna i opłacalna tylko tam, gdzie na danych wejściowych przeprowadza się wiele skomplikowanych obliczeń. Natomiast w małych maszynach do przetwarzania danych, gdzie wykonuje się niewiele prostych obliczeń, translacja taka jest nieopłacalna i w jej miejsce stosuje się system dziesiętny przedstawiania liczb i odpowiednią arytmetykę.

System ten ustępuje poprzednim pod względem szybkości działania, a ponadto liczby systemu dziesiętnego zajmują na ogół więcej pamięci niż liczby poprzednich systemów. Niemniej przejście z postaci alfanumerycznej na ten system i odwrotnie jest bardzo proste i nosi odpowiednio nazwę operacji **„pakowania"** (postać alfanumeryczna → postać dziesiętna) i **„rozpakowania"** (postać dziesiętna → postać alfanumeryczna).

W systemie dziesiętnym każda cyfra dziesiętna i znak liczby przedstawiane są za pomocą grupy 4 bitów. Ponieważ na 4 bitach można zapisać 16 różnych kombinacji, dowolnych 10 z nich można wybrać do reprezentowania cyfr dziesiętnych. Otrzymujemy w ten sposób różne systemy kodowania dwójkowego cyfr dziesiętnych.

Najczęściej stosowane systemy, to **kod naturalny 8421** (wybrane kombinacje od 0000 do 1001) i **kod z nadmiarem 3** (wybrane kombinacje od 0011 do 1100). W kodzie naturalnym wybiera się dziesięć początkowych z 16 kombinacji, zaś w kodzie z nadmiarem 3 przyjmuje się tzw. środkowe kombinacje. Tabela 2.4. ilustruje te dwa sposoby kodowania cyfr dziesiętnych.

Tabela 2.4

Czterobitowa kombinacja dwójkowa	Przyporządkowana danej kombinacji dwójkowej cyfra dziesiętna w kodzie	
	naturalnym	z nadmiarem 3
0000	0	—
0001	1	—
0010	2	—
0011	3	0
0100	4	1
0101	5	2
0110	6	3
0111	7	4
1000	8	5
1001	9	6
1010	—	7
1011	—	8
1100	—	9
1101	—	—
1110	—	—
1111	—	—

Kombinacje dwójkowe oznaczone w powyższej tabeli przez znak — wykorzystuje się przeważnie do oznaczenia znaku liczby (np. kombinacja 1100 do oznaczenia znaku plus i kombinacja 1101 do oznaczenia znaku minus) lub do wykrycia błędnej pracy maszyny cyfrowej (np. kombinacje 1010, 1011, 1110 i 1111 traktowane są przez sprzęt maszyny jako błędne)

Dla kodu naturalnego liczbę dziesiętną N o n cyfrach przedstawić można ogólnie w postaci:

$$N=\sum_{i=0}^{n-1} d_i \cdot 10^i; \qquad d_i = 8 \cdot C_3^i + 4 \cdot C_2^i + 2 \cdot C_1^i + C_0^i$$

gdzie d_i oznacza cyfrę dziesiętną C_0^i, C_1^i, C_2^i, C_3^i, — cztery bity reprezentujące cyfrę d_i.

W maszynach Jednolitego Systemu, w przeciwieństwie do poprzednich dwu systemów, gdzie liczba przedstawiana była w jednym lub dwu słowach maszynowych, liczby dziesiętne mają zmienną długość określoną przez ilość znaczących cyfr. Z tego względu arytmetyka dziesiętna jest łatwiejsza do realizacji w maszynach o zmiennej długości słowa, czyli maszynach znakowych.

Znak liczby dziesiętnej umieszczany jest po najmniej znaczącej cyfrze. Wynika to stąd, że wszystkie działania na liczbach wykonywane są od najmniej znaczących cyfr, a przed rozpoczęciem działania konieczna jest znajomość znaku liczby.

Liczby ujemne systemu dziesiętnego przedstawiane są jednym z trzech sposobów: moduł-znak, uzupełnienie do dziewięciu (który odpowiada systemowi negacji dla liczb dwójkowych) i uzupełnienie do dziesięciu (odpowiadającym systemowi uzupełnieniowemu).

Dodawanie dwu liczb w systemie dziesiętnym odbywa się przez kolejne dodawanie par cyfr. Sumę dwu cyfr tworzy się przez dodawanie binarne dwu liczb dwójkowych 4-bitowych reprezentujących te cyfry. Otrzymana suma może zawierać się w przedziale 0÷19 i wymaga następnie przekodowania na odpowiednią cyfrę dziesiętną wyniku i przygotowania ewentualnego przeniesienia na następną grupę 4-bitową. Łatwość utworzenia cyfry wyniku z sumy binarnej i przeniesienia jest jednym z aspektów, który uwzględnia się przy wyborze kodu.

2.6.4. Zamiana liczb dziesiętnych całkowitych na dwójkowe

Proces zamiany postaci dziesiętnej liczby całkowitej na dwójkową wyjaśnimy na podstawie przykładu: liczba dziesiętna 27 ma być sprowadzona do postaci binarnej. Zauważmy, że liczbę 27 można przedstawić w postaci:

$$27=\frac{27}{2}\cdot 2=\left(13+\frac{1}{2}\right)\cdot 2=13\cdot 2^1+1\cdot 2^0,$$

zaś liczbę 13 można przedstawić:

$$13=\frac{13}{2}\cdot 2=\left(6+\frac{1}{2}\right)\cdot 2=6\cdot 2^1+1\cdot 2^0, \text{ zatem}$$

$$27=(6\cdot 2^1+1\cdot 2^0)\cdot 2^1+1\cdot 2^0=6\cdot 2^2+1\cdot 2^1+1\cdot 2^0$$

Postępując analogicznie z liczbą 6 i 3 otrzymamy:

$$27=1\cdot 2^4+1\cdot 2^3+0\cdot 2^2+1\cdot 2^1+1\cdot 2^0$$

Otrzymane cyfry dwójkowego rozwinięcia, tj. ciąg bitów 11011, dają wartość 27.

Powyższy proces można sformułować następująco: dzielimy daną całkowitą liczbę dziesiętną przez 2. Resztę z dzielenia (1 lub 0) zapisujemy w ciągu zaczynając z prawej strony. Dla otrzymania dalszych cyfr binarnych dzieli się przez 2 częściowy iloraz (pierwszy otrzymany bit reszty jest najmniej znaczącą pozycją liczby dwójkowej).

Przykład. Zamiana postaci dziesiętnej liczby 327 na dwójkową:

ilorazy częściowe	reszty
327	—
163	1
81	1
40	1
20	0
10	0
5	0
2	1
1	0
	1, tj. 101000111

2.6.5. Zamiana dziesiętnej liczby ułamkowej na dwójkową

Ułamek dziesiętny 0,62 można przekształcić:

$$0{,}62 = \frac{0{,}62}{2} \cdot 2 = \frac{1{,}24}{2} = 1 \cdot 2^{-1} + 0{,}24 \cdot 2^{-1};$$

$$0{,}24 = \frac{0{,}24}{2} \cdot 2 = \frac{0{,}48}{2} = 0 \cdot 2^{-1} + 0{,}48 \cdot 2^{-1}, \text{ zatem:}$$

$0{,}62 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0{,}48 \cdot 2^{-2}$ i ostatecznie:
$0{,}62 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + 0{,}92 \cdot 2^{-4}$
czyli zaokrąglony (do 4 bitów) ułamek dwójkowy: 0,1001.

Na podstawie przykładu można sformułować proces konwersji: mnoży się dany ułamek dziesiętny kolejno przez 2, wydzielając przeniesienia otrzymane na lewo od przecinka. Przeniesienia te tworzą kolejne bity poprzecinkowe liczby w postaci binarnej (poczynając od lewej).

Przykład. Zamiana dziesiętnej liczby ułamkowej 0,62 na postać dwójkową (4-bitową):

iloczyny częściowe	przeniesienia
0,62	—
1,24	1
0,48	0
0,96	0
1,92	1, czyli 0,1001

Uwaga. Liczbę mieszaną (np. liczbę 95,62) można przedstawić w postaci całkowitej (np. $9562 \cdot 10^{-2}$) lub ułamka właściwego (np. $0{,}9562 \cdot 10^{2}$) i pomnożyć — w systemie dwójkowym — przez wartość odpowiedniej potęgi 10 (np. 10^{-2} lub 10^{2}).

2.6.6. Zamiana całkowitej liczby dwójkowej na dziesiętną

Proces ma podobny przebieg jak w przypadku przejścia z postaci dziesiętnej do dwójkowej, z tą różnicą, że przekształcana liczba zostaje podzielona przez 10 (zamiast przez 2). Reszty z dzielenia liczby początkowej lub ilorazów częściowych tworzą ciąg cyfr dziesiętnych danej liczby (pierwsza reszta określa najmniej znaczącą cyfrę liczby).

Przykład. Sprowadzenie do postaci dziesiętnej liczby dwójkowej 10110111

Przypomnijmy, że $(10)_{10} = (1010)_2$

```
 10110111 | 1010          pierwsze dzielenie
-1010     | 10010
   1011
  -1010
     11,    reszta = 3 (cyfra jednostek)

 10010 | 1010              drugie dzielenie
-1010  | 1
 1000,   reszta = 8 (cyfra dziesiątek)

 1 | 1010                  trzecie dzielenie
-0
 1    reszta = 1 (cyfra setek)
```

Zatem $(10110111)_2 = (183)_{10}$

2.6.7. Zamiana ułamkowej liczby dwójkowej do postaci dziesiętnej

W tym przypadku mnoży się (w systemie dwójkowym) dany ułamek przez liczbę 10, wydzielając część otrzymaną na lewo od przecinka. Te kolejne wydzielone części tworzą ciąg cyfr dziesiętnych danego ułamka (od lewej).

Przykład. Sprowadzenie do postaci dziesiętnej ułamka dwójkowego 0,1101110

```
   0,1101110
        1010,        mnożenie pierwsze
  1 1011100
110 1110
1000,1001100,        zatem (1000)2 = (8)10

   0,1001100
        1010,        mnożenie drugie
  1 0011000
100 1100
0101,1111000,        zatem (0101)2 = (5)10

   0,1111000
        1010,        mnożenie trzecie
  1 1110000
111 1000
1001,0110000,        zatem (1001)2 = (9)10

   0,0110000
        1010,        mnożenie czwarte
   1100000
 11 0000
0011,1100000         zatem (0011)2 = (3)10
```

Ostatecznie jest: $(0{,}1101110)_2 = (0{,}8593)_{10}$

2.6.8. Kodowanie dwójkowe cyfr dziesiętnych

W wielu przypadkach — o których już była mowa — zachodzi potrzeba przechowywania w maszynie liczb w postaci dziesiętnej (np. dla maszyn pracujących w dziesiętnym systemie liczenia). Wówczas jest konieczne przedstawienie (kodowanie) każdej cyfry dziesiętnej za pomocą pewnej ilości cyfr dwójkowych. Istnieje wiele różnych sposobów kodowania cyfr dziesiętnych. W większości stosuje się cztery pozycje dwójkowe, z których każdej przyporządkowuje się pewną wartość lub „wagę". Często stosuje się jeden z następujących systemów:

A. 8-4-2-1;

cyfra dziesiętna	kombinacja kodowa wagi 8421
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

W systemie 8-4-2-1 oznaczenia kodowe mają tę samą wartość co normalne dwójkowe rozwinięcia cyfr dziesiętnych.

B. 2-4-2-1;

cyfra dziesiętna	kombinacja kodowa wagi 2421
0	0000
1	0001
2	0010
3	0011
4	0100
5	1011
6	1100
7	1101
8	1110
9	1111

czyli odrzucono sześć kombinacji „środkowych" z 16 kombinacji możliwych.

C. Kod „z nadmiarem trzy";

cyfra dziesiętna	kombinacja kodowa wagi 8421 plus 3
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

W tym kodzie przyjęto 10 „środkowych" kombinacji i odrzucono 6 „skrajnych" kombinacji.

Ostatnie dwa systemy kodowania mają wiele wygodnych cech w realizacji technicznej. Podamy dla przykładu dwie z nich:

1) uzupełnienie do dziewięciu dowolnej cyfry dziesiętnej może być dokonane za pomocą uzupełnienia (negacji) każdej cyfry dwójkowej, co jest bardzo proste w realizacji technicznej;
2) przy sumowaniu binarnym dwóch cyfr dziesiętnych cyfra sumy daje pięć cyfr binarnych, jeśli suma ta jest większa lub równa dziesięciu, lub daje cztery cyfry binarne, jeśli suma jest mniejsza od dziesięciu.

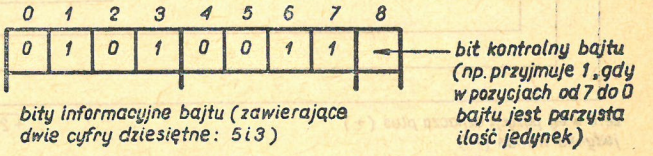
D. Kod „dwa z pięciu". Jest on szczególnie przydatny przy kontroli niezawodności pracy maszyny cyfrowej. Kontrola możliwa jest dzięki wprowadzeniu dodatkowego piątego bitu (co podraża sprzęt). Kod ten zakłada, że w każdej kombinacji kodowej przedstawiającej dowolną cyfrę dziesiętną występują dokładnie dwie jedynki (zatem większa lub mniejsza ilość jedynek jest uważana za błąd). Kod ma wagi 0,1,2,3,6. Pozwala on (w sensie wag) zakodować cyfrę trzy na dwa sposoby; jeden z tych sposobów został wykorzystany dla kodowania cyfry zero:

cyfra dziesiętna	kombinacja kodowa wagi 01236
0	01100
1	11000
2	10100
3	10010
4	01010
5	00110
6	10001
7	01001
8	00101
9	00011

2.6.9. Przedstawienie liczb w maszynach cyfrowych

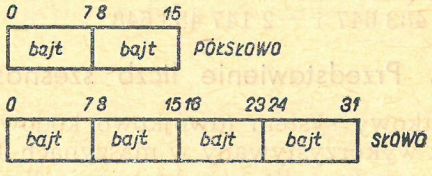
We współczesnych maszynach cyfrowych spotyka się różne typy liczb (np. liczby stałoprzecinkowe, liczby zmiennoprzecinkowe, liczby dziesiętne kodowane dwójkowo), na których wykonuje się różne operacje. Argumenty początkowe i wyniki tych operacji przechowuje się w tzw. rejestrach lub komórkach maszyny. Rejestry i komórki maszyny składają się z pewnej ilości elementów pamiętających. Każdy element może zapamiętać jeden z dwóch stanów: zero (0) lub jedynka (1). Zatem każdy element pamiętający może być traktowany jako pozycja liczby dwójkowej. Jeśli rejestr (lub komórka) maszyny składa się z n takich elementów pamiętających, to rejestr (komórka) może przechowywać n-bitowe liczby.

Z wielu względów wygodnie jest numerować (adresować) rejestry i komórki maszyny cyfrowej, przy czym adresowalną „porcją" informacji bywa przeważnie ciąg 8, 12, 16, 24, 32 lub 64 bitów. W maszynach Jednolitego Systemu najmniejszą adresowalną informacją jest tzw. bajt 8-bitowy, w którym można przechować jeden znak alfanumeryczny, dwie 4-bitowe cyfry dziesiętne, część parametru logicznego lub część liczby dwójkowej (rys. 2.4.). Technicznie każdy bajt zostaje rozszerzony o dodatkowy bit, (tzw. bit kontrolny), w którym przechowuje się pewną cechę związaną z kontrolą 8-bitowej informacji bajtu (np. cecha nieparzystości jedynek) lub z kontrolą informacji grupy bajtów (np. dla 8 bajtów).



Rys. 2.4. Struktura bajtu (w EMC Jednolitego Systemu)

Ciąg kolejnych bajtów tworzy większe jednostki informacji (rys. 2.5.):



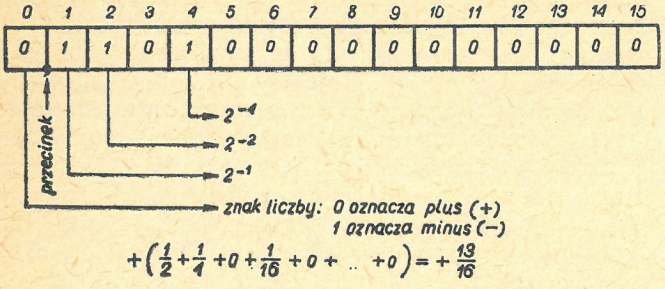
Rys. 2.5. Struktura półsłowa i słowa

— półsłowo, składające się z dwóch bajtów;
— słowo, składające się z czterech bajtów;
— słowo podwójne, składające się z ośmiu bajtów.

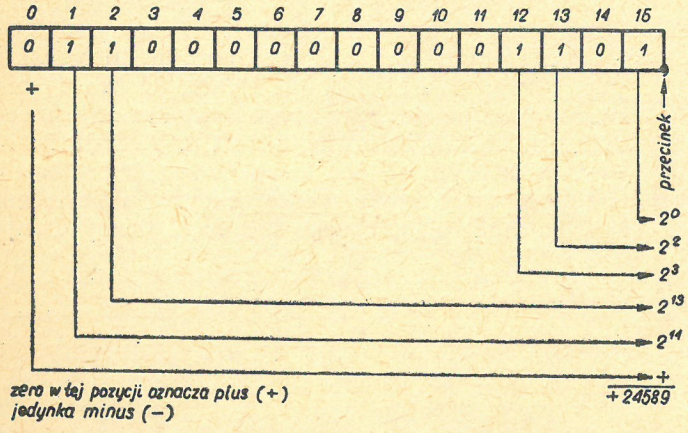
2.6.9.1. Przedstawienie liczb stałoprzecinkowych

Liczby stałoprzecinkowe mają, jak wiemy, ustaloną pozycję przecinka. Jeśli umowny przecinek ustalimy pomiędzy pozycjami 0 i 1 słowa maszynowego, to mamy do czynienia z liczbami ułamkowymi (krótkimi — jeśli zapisane są w półsłowie lub długimi — jeśli zapisane są pełnym słowem). Jeśli zaś przecinek występuje bezpośrednio po ostatniej pozycji słowa (lub półsłowa), mamy do

czynienia z liczbami całkowitymi jednosłowowymi (lub półsłowowymi). Rys. 2.6 i rys. 2.7 przedstawiają (odpowiednio) przykład zapisu liczby ułamkowej $\left(+\frac{13}{16}\right)$ i liczby całkowitej (+24589) w półsłowie.



Rys. 2.6. Zapis ułamka w półsłowie



Rys. 2.7. Zapis liczby całkowitej w półsłowie

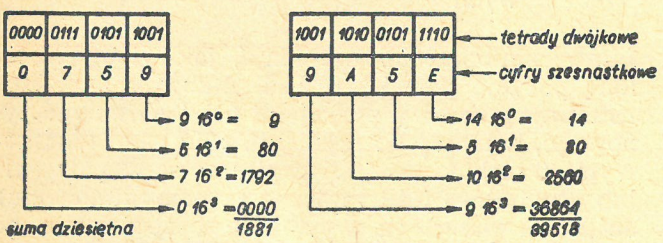
Maksymalna dodatnia liczba całkowita, którą można zapisać w jednym półsłowie, wynosi 32 767, natomiast najmniejsza ujemna liczba całkowita wynosi −32 768. Postać dwójkowa tych liczb jest następująca:

0111111111111111 = +32 767,
1000000000000000 = −32 768.

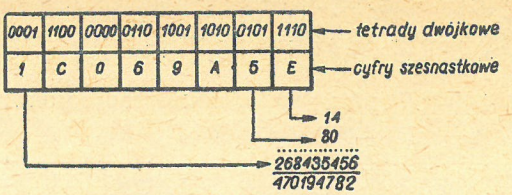
Maksymalna i minimalna liczba całkowita, które można zapisać w 32-bitowym słowie, wynoszą odpowiednio:
+2 147 483 647 i −2 147 483 648.

2.6.9.2. Przedstawienie liczb szesnastkowych

Szesnastkowy system (dwójkowo kodowany) jest również wykorzystywany w maszynach Jednolitego Systemu (np. dla przedstawienia liczb zmiennoprzecinkowych, kodów operacji itp.).
Układ szesnastkowy (heksadecymalny) ma 16 cyfr o symbolach: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F; którym to cyfrom przyporządkowano wartości od 0 do 15. Dla zapisu jednej cyfry układu szesnastkowego używa się 4 bity (tzw. tetrada) rejestru lub komórki, czyli półbajtu.



Rys. 2.8. Przedstawienie liczb szesnastkowych przez półsłowo

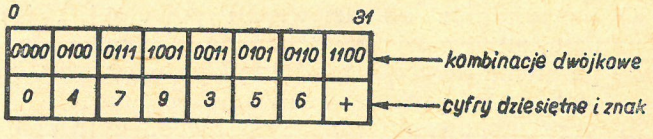


Rys. 2.9. Przedstawienie liczby szesnastkowej przez słowo

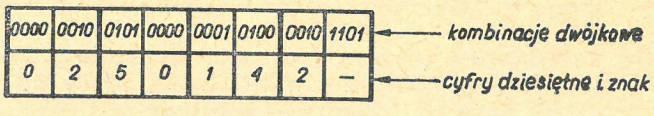
Rysunki 2.8 i 2.9 przedstawiają 4- i 8-cyfrową liczbę szesnastkową (bez znaku).
Znak liczby szesnastkowej można oczywiście zapisać tylko za pomocą dwóch cyfr, np. zero dla znaku plus (+) i jedynka dla znaku minus (−). Dlatego często przeznacza się w maszynach nie całą tetradę dla zapisu znaku liczby szesnastkowej, lecz tylko jedną pozycję rejestru.

2.6.9.3. Przedstawienie liczb dziesiętnych

Cyfry liczby dziesiętnej, tj. cyfry od 0 do 9, są kodowane za pomocą cyfr dwójkowych (tj. kombinacjami od 0000 do 1001). Zatem każda cyfra dziesiętna kodowana dwójkowo zajmuje pół bajtu (podobnie jak cyfry szesnastkowe). Rys. 2.10 i 2.11 podają przykłady rozmieszczenia liczb dziesiętnych w słowie. Liczby dziesiętne, których cyfry zostały rozmieszczone po dwie w jednym bajcie, nazywają się liczbami u p a k o w a n y m i. Znak upakowanej liczby dziesiętnej występuje zawsze w najmłodszym (tzn. w prawym) półbajcie. W tym półbajcie może wystąpić dowolna kombinacja dwójkowa.



Rys. 2.10. Rozmieszczenie liczby +479356 w słowie maszynowym



Rys. 2.11. Rozmieszczenie liczby −250142 w słowie maszynowym

Przyporządkowanie znakom liczby dziesiętnej określonych kombinacji dwójkowych zależy od przyjętego kodu wewnętrznego maszyny. W maszynach Jednolitego Systemu znakowi plus (+) przyporządkowuje się normalnie kombinacje *C* i *F* (czyli 1100 i 1111), zaś znakowi minus (−) kombinację *D* (czyli 1101). Pozostałe kombinacje spotykane w działaniach arytmetycznych są na ogół traktowane jako niepoprawne.

2.6.9.4. Symboliczne przedstawienie liczb dziesiętnych

Każda maszyna cyfrowa posiada własny tzw. kod wewnętrzny, który na ogół jest różny od kodów urządzeń zewnętrznych. W kodzie wewnętrznym przedstawia się w sposób jednoznaczny pełny repertuar znaków (np. cyfry, litery, znaki operatorów itd.) stosowanych w wewnętrznych działaniach danej maszyny.

Tabela 2.5

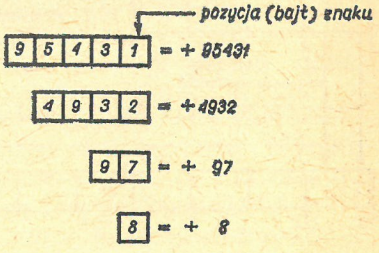
Kombinacja szesnastkowa (bajtu)	Kombinacja dwójkowa	Kodowany symbol (w kodzie wewnętrz.): cyfra bez znaku
F0	11110000	0
F1	11110001	1
F2	11110010	2
F3	11110011	3
F4	11110100	4
F5	11110101	5
F6	11110110	6
F7	11110111	7
F8	11111000	8
F9	11111001	9

Tabela 2.6

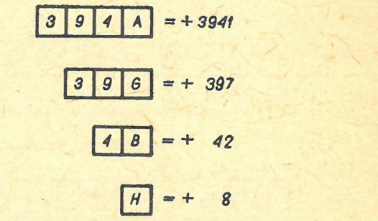
Dodatnie strefowe cyfry dziesiętne			
cyfra		kombinacja dwójkowa	symbol kodu wewnętrznego
dziesiętna ze znakiem plus	szesnastkowa		
+0	C0	1100 0000	
+1	C1	1100 0001	A
+2	C2	1100 0010	B
+3	C3	1100 0011	C
+4	C4	1100 0100	D
+5	C5	1100 0101	E
+6	C6	1100 0110	F
+7	C7	1100 0111	G
+8	C8	1100 1000	H
+9	C9	1100 1001	I

Ujemne strefowe cyfry dziesiętne			
cyfry		kombinacja dwójkowa	symbol kodu wewnętrznego
dziesiętna ze znakiem minus	szesnastkowa		
−0	D0	1101 0000	
−1	D1	1101 0001	I
−2	D2	1101 0010	K
−3	D3	1101 0011	L
−4	D4	1101 0100	M
−5	D5	1101 0101	N
−6	D6	1101 0110	O
−7	D7	1101 0111	P
−8	D7	1101 1000	Q
−9	D9	1101 1001	R

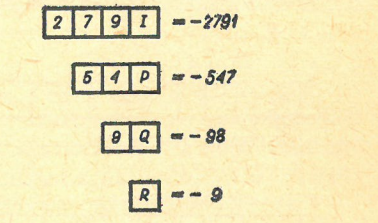
Tabele 2.5 i 2.6 przedstawiają tylko część kodu wewnętrznego maszyn cyfrowych Jednolitego Systemu. Jak widać z tych tabel, można również przedstawiać cyfry dziesiętne ze znakiem za pomocą odpowiednich symboli (liter). Zatem grupy takich symboli można traktować jako ciągi tekstów literowo-cyfrowych lub jako liczby dziesiętne (nazywane również liczbami rozpakowanymi lub nieupakowanymi). Każdy symbol kodu wewnętrznego składa się z 8 bitów, czyli zajmuje cały bajt. Cztery lewe bity symbolu tworzą tzw. strefę, zaś cztery prawe tworzą numer symbolu (w danej strefie). Przy zapisie dziesiętnym liczb rozpakowanych dopuszcza się trzy typy cyfr dziesiętnych (tabele 2.5 i 2.6): cyfry bez znaku, cyfry opatrzone znakiem (+) i cyfry opatrzone znakiem (−). Cyfry bez znaku traktowane są albo jako cyfry dodatnie (jeśli występują w pozycji znakowej), albo jako neutralne (gdy występują w ciągu cyfr przedstawiających daną liczbę).
Cyfry opatrzone znakiem (nazywane też cyframi strefowymi) występują normalnie w pozycji znakowej rozpakowanej liczby dziesiętnej (rys. 2.12, 2.13 i 2.14).



Rys. 2.12. Przykłady rozpakowanych liczb dziesiętnych zapisanych za pomocą tzw. cyfr bez znaku



Rys. 2.13. Przykłady rozpakowanych liczb dziesiętnych zapisanych za pomocą tzw. dodatnich strefowych cyfr



Rys. 2.14. Przykłady rozpakowanych liczb dziesiętnych zapisanych za pomocą tzw. ujemnych strefowych cyfr

2.6.9.5. Przedstawienie liczb zmiennoprzecinkowych

Liczby zmiennoprzecinkowe składają się, jak już wiemy, z pary liczb zwanych odpowiednio mantysą (m) i wykładnikiem (w) lub cechą. Liczby m i w danej liczby zmiennoprzecinkowej X dobiera się w ten sposób, aby zachodziła równość:

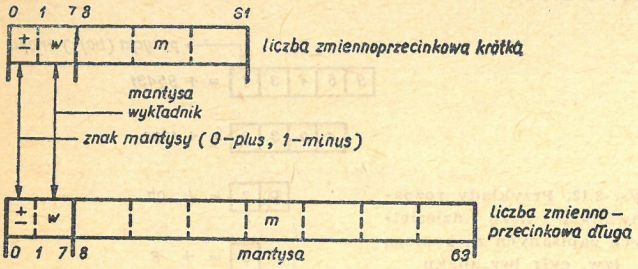
$$X = m \cdot p^w = (m;\ w)$$

gdzie p jest podstawą układu liczenia.

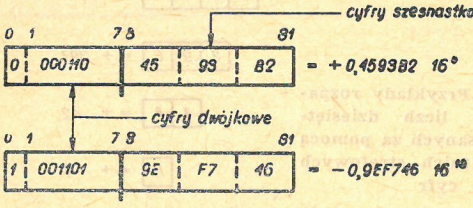
We współczesnych maszynach cyfrowych liczby zmiennoprzecinkowe definiowane są na ogół w układzie dwójkowym lub szesnastkowym (kodowanym dwójkowo). Ponieważ podstawa układu liczenia jest stała dla danej maszyny, nie zachodzi potrzeba zaznaczania jej wielkości w maszynie. Zatem w słowach maszynowych wystarczy określić tylko wielkość mantysy i wykładnika, czyli liczby m i w. Wielkość liczb m i w jest różna w różnych maszynach i zależy od założonej dokładności obliczeń (o której decyduje ilość cyfr mantysy) oraz od wymaganego zakresu obliczeń (który zależy od wielkości wykładnika).

W maszynach np. ODRA 1300 przyjmuje się: $p=2$, 38-bitową mantysę i 9-bitowy wykładnik (wraz ze znakiem). Dzięki temu zakres liczb zmiennoprzecinkowych wynosi od 2^{-256} do 2^{+255}, natomiast osiągana dokładność wynosi około 11 cyfr dziesiętnych. Natomiast w maszynach JS EMC przyjmuje się: $p=16$, 7-bitowy wykładnik

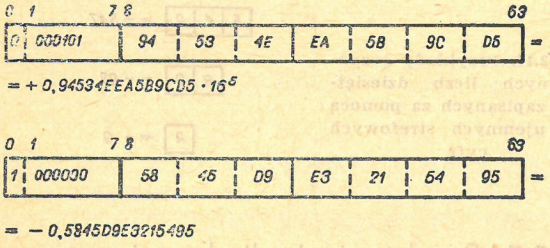
(wraz ze znakiem) i 24,56 lub 112-bitową mantysę (bez znaku). Znak mantysy zajmuje jeden bit i zawsze znajduje się w bajcie wykładnika (rys. 2.15, 2.16 i 2.17). W związku z tym zakres liczb zmiennoprzecinkowych maszyn Jednolitego Systemu wynosi od 16^{-64} do 16^{+63}, czyli od 10^{-78} do 10^{+75}.



Rys. 2.15. Przedstawienie krótkiej (jedno słowo) i długiej (dwa słowa) liczby zmiennoprzecinkowej w maszynach JS EMC



Rys. 2.16. Przykłady rozmieszczenia liczb zmiennoprzecinkowych w słowie JS EMC



Rys. 2.17. Przykłady rozmieszczenia liczb zmiennoprzecinkowych w podwójnym słowie JS EMC

Natomiast dokładność obliczeń jest różna i wynosi:

— 6 cyfr szesnastkowych (lub około 7 cyfr dziesiętnych) dla 24-bitowej mantysy,
— 14 cyfr szesnastkowych (lub około 17 cyfr dziesiętnych) dla 56-bitowej mantysy,
— 28 cyfr szesnastkowych (lub około 34 cyfr dziesiętnych) dla 112-bitowej mantysy.

Jak widać z powyższych przykładów, mantysa jest przedstawiana w arytmetyce „znak-moduł"; tzn. w pozycji zerowej słowa zapisuje się znak, zaś moduł mantysy w pozycjach 8—31 słowa, 8—63 (dla liczb długich) lub w pozycjach 8—63 dwóch kolejnych długich słów (dla liczb tzw. rozszerzonej precyzji). Przyjmuje się również, że wykładnik w jest wewnątrz maszyny zawsze dodatni. Oznacza to, że wykładnik „zewnętrzny", który przyjmuje liczby całkowite od -64 do $+63$, zostaje przez program sterujący maszyny zamieniony na wykładnik „wewnętrzny" według wzoru:

wykładnik wewnętrzny = wykładnik zewnętrzny + 64

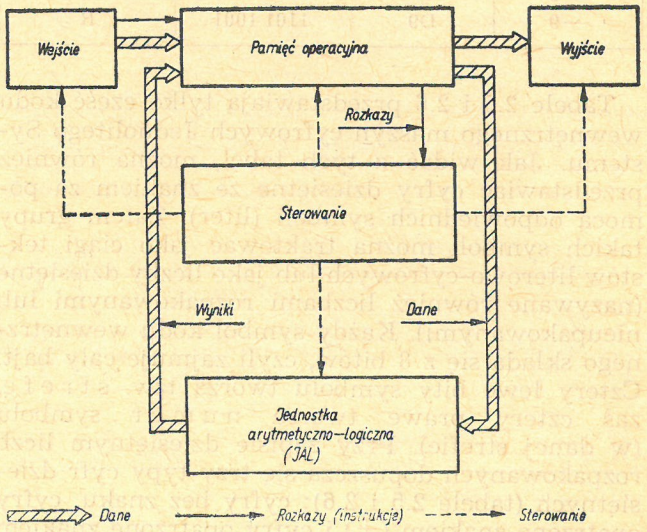
W ten sposób wykładnik wewnątrz maszyny przyjmuje wartości od 0 do $+127$, czyli jest zawsze dodatni. Zatem nie zachodzi potrzeba określania znaku wykładnika w słowie maszynowym.

2.7. Podstawowe zasady działania maszyny cyfrowej

Maszynę cyfrową można zdefiniować jako urządzenie do przetwarzania informacji. Przetwarza ona dane wejściowe w wyniki według określonego algorytmu (schematu) liczenia. Często danymi wejściowymi są liczby opisujące rozwiązywany problem lub różne wielkości wynikające z pomiarów parametrów procesu przemysłowego, a wynikami — liczby stanowiące rezultaty obliczeń lub sekwencje impulsów sterujących procesem. Jednakże zarówno w jednym jak i w drugim przypadku przekształcenie danych w wyniki jest dokonywane w maszynie według określonego programu. Program ten, mówiąc opisowo, określa co należy zrobić z danymi, by uzyskać wyniki. Program jest więc opisem procesu, który wykonany zostanie w maszynie na określonych danych i w wyniku którego wytworzone zostaną określone wyniki. Program może być ten sam dla bardzo różnych danych. Tak np. w przypadku najprostszego problemu obliczeniowego: dodania dwu liczb $s=a+b$, program sprowadza się do informacji „dodaj", gdzie danymi są dwie liczby a i b, za wynikiem — wynik dodawania, czyli liczba s. W przykładzie tym widać wyraźnie, że program jest ten sam dla dowolnych dwu liczb (a_1 i b_1, a_2 i b_2,...).

Program opisuje sposób realizacji procesu, czyli opisuje tzw. algorytm tego procesu.

Maszyny cyfrowe mogą rozwiązywać dowolny problem, jeżeli tylko zostanie on przedstawiony w postaci programu, a rozwiązanie jego nie przekracza technicznych możliwości maszyny. Cecha ta jest najistotniejszą cechą uniwersalnych maszyn cyfrowych. Tak więc, by rozwiązać problem przy pomocy uniwersalnej maszyny cyfrowej, należy oprócz danych przekazać do maszyny program. Dopiero wtedy, w wyniku pracy maszyny mogą zostać uzyskane wyniki. Oczywiście raz wprowadzony do maszyny program może być obliczany wielokrotnie i nie istnieje konieczność każdorazowego wprowadzania programu wraz z danymi. Jednakże przed przystąpieniem do obliczeń program musi zostać do maszyny wprowadzony.



Rys. 2.18. Schemat blokowo-funkcjonalny maszyny cyfrowej

Maszyny cyfrowe w zależności od swej wielkości i przeznaczenia stanowią systemy różniące się znacznie między sobą. Jednakże najogólniej budowę omawianych maszyn można przedstawić za pomocą schematu blokowego (rys. 2.18).
Przyjmuje się w uproszczeniu, że maszyna cyfrowa zawiera pięć bloków funkcjonalnych, przy czym bloki te w różnych maszynach mogą się znacznie różnić od siebie zarówno ze względu na swą budowę, jak i sposób wykorzystania. Blokami tymi są:

1) urządzenia wejścia (wejście),
2) pamięć operacyjna (lub główna),
3) jednostka arytmetyczno-logiczna (JAL), zwana często w uproszczeniu arytmometrem,
4) urządzenie sterujące (sterowanie),
5) urządzenie wyjścia (wyjście).

Bloki te powiązane są między sobą sygnałami elektronicznymi, które to sygnały mogą przedstawiać dane, rozkazy lub informacje sterujące.

2.7.1. Pamięć operacyjna

Pamięć operacyjna składa się z pewnej liczby rejestrów (np. N), z których każdy przeważnie ma długość słowa maszynowego (np. 32 bity). Rejestr, zwany też miejscem lub komórką pamięci operacyjnej, posiada swój numer będący liczbą z przedziału 0, N-1. Numer ten nazywa się adresem miejsca pamięci. Pamięć operacyjna posiada wyjście informacyjne (skąd pobiera się informacje), wejście informacyjne i wejście adresowe. Jej działanie polega na wykonaniu operacji odczytu i zapisu:

— operacja odczytu — na wyjściu informacyjnym ukazuje się słowo maszynowe, przechowywane w miejscu pamięci, którego adres został podany na wejście adresowe;
— operacja zapisu — słowo maszynowe podane na wejście informacyjne zapamiętane zostaje w miejscu pamięci, którego adres podany został na wejście adresowe.

W pamięci operacyjnej przechowywane są dane podlegające przetwarzaniu oraz program określający działanie maszyny. Dane mogą mieć postać liczb lub ciągów znaków alfanumerycznych przechowywanych w postaci słów maszynowych w miejscach pamięci. Program ma postać ciągu rozkazów. Każdy rozkaz jest oddzielnym słowem maszynowym i określa czynność lub grupę czynności wykonywanych przez maszynę. Typ informacji zawartej w słowie, tzn. czy słowo maszynowe zawiera dane czy rozkaz, zależy wyłącznie od interpretacji tego słowa przez maszynę. Dzięki temu program może być przedmiotem przetwarzania — analogicznie jak dane.

2.7.2. Jednostka arytmetyczno-logiczna

Po rozpoczęciu wykonywania programu dane z pamięci przesyłane są do jednostki arytmetyczno-logicznej (JAL). W jednostce tej wykonywane są operacje arytmetyczne i logiczne. Wyniki tych operacji, jako wyniki częściowe lub ostateczne, przechowywane są w rejestrach JAL, zwanych często akumulatorami lub rejestrami ogólnymi, albo przesyłane są do pamięci operacyjnej. Liczby, na których wykonywane są operacje arytmetyczne, przedstawiane są zazwyczaj w zapisie binarnym lub zapisie dziesiętnym kodowanym binarnie.

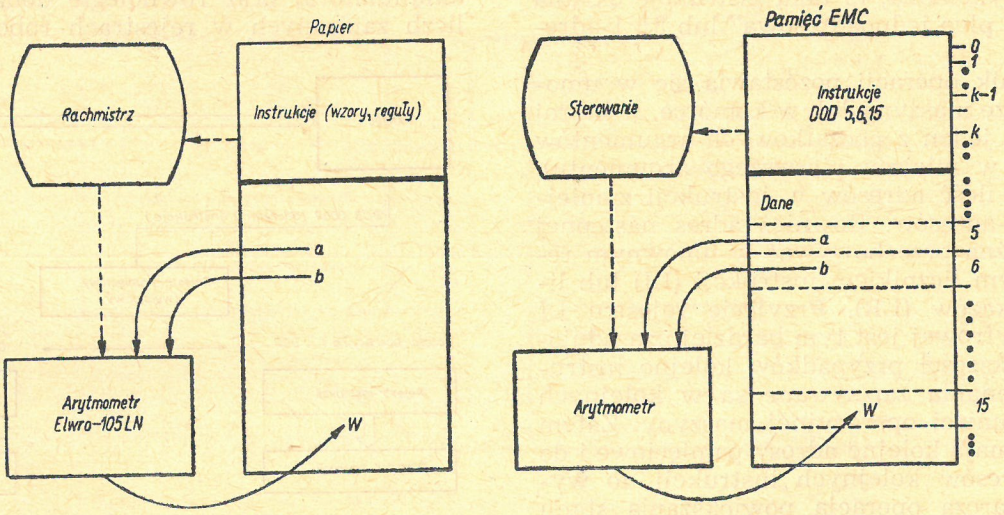
2.7.3. Sterowanie

Zadaniem sterowania jest odczytywanie kolejnych rozkazów (instrukcji) z pamięci i wytwarzanie na podstawie treści rozkazu, sygnałów sterujących, które powodują wykonanie odpowiednich czynności obliczeniowych w JAL (lub arytmometrze).

Zatem kolejność i kierunek przekazywania informacji (danych lub rozkazów) w maszynie jest określany przez blok sterowania. Praca samego bloku sterowania natomiast zależy głównie od sekwencji rozkazów programu wykonywanego.

Blok sterowania, po rozszyfrowaniu operacji pobranego rozkazu, przekazuje różne sygnały sterujące do JAL i innych bloków w celu otwarcia odpowiednich dróg przepływu danych pomiędzy blokami, zgodnie z funkcją (treścią) określoną przez dany rozkaz. Po zakończeniu wykonania funkcji danego rozkazu blok sterowania powoduje pobranie i dekodowanie następnego rozkazu programu z pamięci.

Funkcja bloku sterowania (zadawana przez program) często jest porównywana z pracą rachmistrza biurowego wyposażonego w prosty arytmometr (kalkulator). Rysunek 2.19 przedstawia sche-



Rys. 2.19. Praca rachmistrza i praca maszyny cyfrowej

mat pracy rachmistrza i prostej maszyny cyfrowej. W tradycyjnym systemie obliczania żądanej wielkości rachmistrz jest wyposażony w określoną „wiedzę matematyczną". Oznacza to, że rachmistrz zna wymagane reguły i wzory do rozwiązywania danego zagadnienia obliczeniowego. Zna zwłaszcza kolejność wykonania poszczególnych działań matematycznych oraz miejsce danych początkowych i pośrednich wyników.

W przypadku maszyny cyfrowej cała ta „wiedza matematyczna" zawarta jest w programie zapisanym w pamięci operacyjnej. Program składa się z wielu instrukcji (rozkazów) postępowania. Wykonując kolejno określony ciąg instrukcji otrzyma się rozwiązanie danego zagadnienia. Rys. 2.19 przedstawia schemat wykonania instrukcji DODAWANIE liczb a i b oraz zapisanie wyniku sumy w tej samej pamięci maszyny, w której znajdują się argumenty a i b.

Instrukcja dodawania składa się (w tym przykładzie) z czterech części: kodu operacji (DOD), adresu pierwszego argumentu operacji (5), adresu drugiego argumentu operacji (6) oraz z adresu wyniku operacji (15). Zatem zapis wzoru $w=a+b$, może być przedstawiony w maszynie w postaci instrukcji: DOD 5, 6, 15. Instrukcja ta oznacza dla maszyny cyfrowej następującą funkcję: DODaj do liczby zapisanej w komórce (pamięci) o adresie 5 liczbę zapisaną w komórce 6 oraz umieść wynik dodawania w komórce 15, po czym pobierz do wykonania kolejną instrukcję z pamięci (tj. instrukcję zapisaną bezpośrednią za instrukcją DODaj). Oczywiście w miejscu adresów 5, 6 i 15 mogą wystąpić dowolne inne adresy pamięci operacyjnej. Również w miejscu kodu operacji DOD może wystąpić inny kod określający jedną z dalszych instrukcji (operacji) maszyny cyfrowej, na przykład ODEjmowanie, MNOżenie, DZielenie, PrzeSuwanie, itd.

We współczesnych maszynach cyfrowych spotyka się od 100 do 200 różnych instrukcji, które tworzą tzw. listę instrukcji danej maszyny.

Strukturę opisanej instrukcji postaci DOD 5, 6, 15 można określić jako t r ó j a d r e s o w ą ze względu na występowanie w niej trzech niezależnych adresów, tj. adresu pierwszego argumentu, adresu drugiego argumentu i adresu wyniku operacji określonej przez dany kod. Jeśli do tych adresów dołączy się również adres następnej instrukcji do wykonania, to taką instrukcję określa się jako „trzy plus jednoadresową" lub 3+1-adresową.

Często wynik operacji pozostawia się w umownym rejestrze maszyny lub w komórce, w której znajdował się jeden z początkowych argumentów operacji (np. w miejscu pierwszego argumentu). Można zatem ilość adresów w instrukcji zmniejszyć do 2+1-adresów. Również adres następnej instrukcji można przechowywać w umownym rejestrze, zwanym licznikiem instrukcji (LI) lub licznikiem rozkazów (LR). Przyjęcie rejestru LI w maszynie cyfrowej jest tym bardziej uzasadnione, że w większości przypadków kolejne instrukcje do wykonania zapisywane są w kolejnych komórkach pamięci operacyjnej maszyny. Zatem instrukcje te mają kolejne adresy pamięciowe i do tworzenia adresów kolejnych instrukcji do wykonania wystarcza operacja powiększania stanu licznika instrukcji o jeden (lub o 2, 3, 4, 5 lub 6 — jeśli poprzednia instrukcja zajmuje odpowiednio więcej jednostek pamięciowych, np. bajtów). W maszynach np. Jednolitego Systemu instrukcje mają długość dwóch, czterech lub sześciu bajtów (tj. 8-bitowych pól). Zatem licznik instrukcji EMC Jednolitego Systemu będzie powiększany o 2, 4 bądź 6, przy tworzeniu adresu kolejnej instrukcji do wykonania.

Maszyny o powyższym sposobie rozmieszczenia instrukcji w pamięci nazywa się maszynami s e k w e n c y j n y m i.

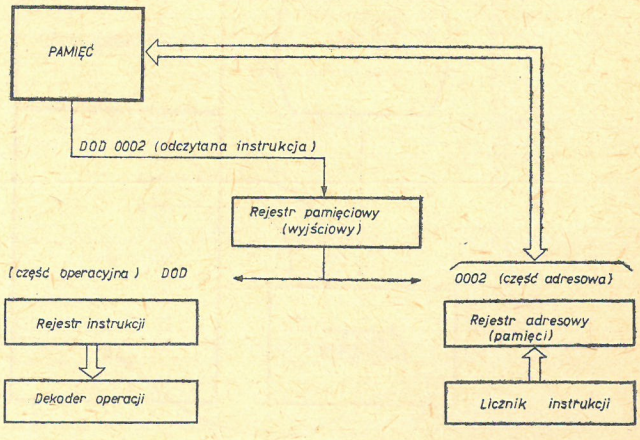
Większość współczesnych maszyn cyfrowych ma instrukcje o strukturze j e d n o a d r e s o w e j, co oznacza, że tylko jeden z argumentów ma swobodny adres; zaś drugi argument operacji, wynik operacji oraz adres następnej instrukcji do wykonania znajdują się w umownych (stałych) rejestrach. Przykładem mogą być maszyny typu ODRA 1300 oraz częściowo maszyny JS EMC.

Instrukcje jednoadresowe pozwalają wykonywać operacje np. w postaci: $(R) \pm (N) \rightarrow R$, gdzie R jest umownym rejestrem jednostki arytmetyczno-logicznej, zaś N — dowolnym adresem pamięci operacyjnej (nawias oznacza zawartość danego rejestru lub komórki). Często w maszynie cyfrowej takich „wyróżnionych" rejestrów jest więcej (np. 8 w maszynach ODRA 1300 i 16 w maszynach JS EMC) i wówczas zachodzi potrzeba adresowania także tych rejestrów. Niemniej adresy rejestrów wyróżnionych przebiegają bardzo małe zakresy (np. od 0 do 15) w stosunku do zakresów adresów pamięciowych
(np. 0, 1, 2, ... $2^{24}-1$).

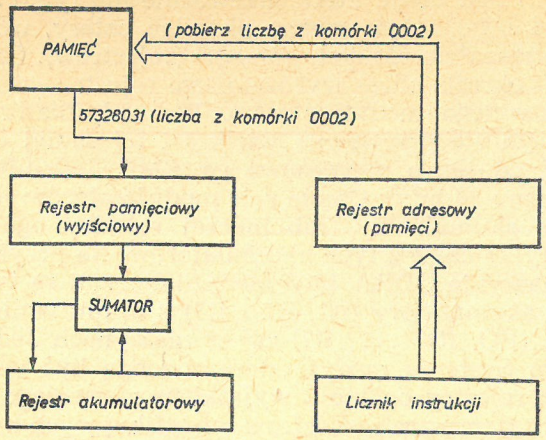
Adresy rejestrów umownych (akumulatorów lub ogólnych) nazywa się czasem adresami „połówkowymi" ze względu na ich ograniczony zakres.

Wykonanie instrukcji programu dzieli się na ogół na dwa odrębne etapy:
— pobranie instrukcji z pamięci do bloku sterowania oraz dokonanie ewentualnych modyfikacji jej adresów,
— wykonanie operacji określonej przez kod danej instrukcji oraz odpowiednie powiększenie licznika instrukcji (lub zmianę LI w przypadku tzw. instrukcji skokowych).

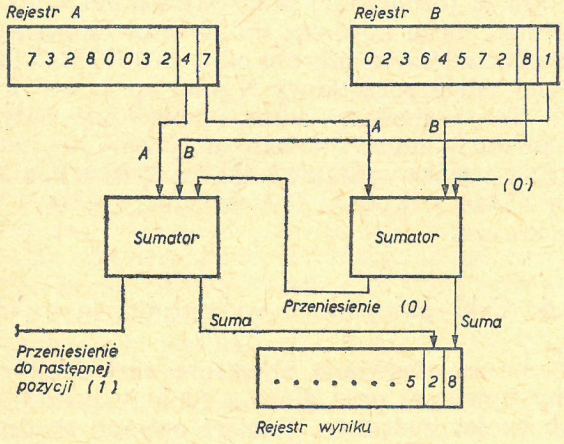
Rysunki 2.20, 2.21 oraz 2.22 pokazują odpowiednio proces pobrania instrukcji (DOD 0002), proces wykonania instrukcji (polegającej na odczytaniu zawartości komórki i zapisanie tej zawartości do akumulatora) oraz równoległe dodawanie dwóch liczb zapisanych w rejestrach roboczych A i B.



Rys. 2.20. Schemat pobrania instrukcji do sterowania



Rys. 2.21. Schemat wykonania instrukcji



Rys. 2.22. Schemat równoległego dodawania dwóch dziesiętnych liczb 10-cyfrowych (7328003247 i 0236457281)

2.7.4. Urządzenia zewnętrzne

Urządzenia zewnętrzne omawianej maszyny składają się z urządzenia wejściowego i urządzenia wyjściowego. Urządzenie wejściowe jest przeznaczone do wprowadzania danych i programów do pamięci operacyjnej maszyny. Urządzenie odczytuje informację z nośnika zewnętrznego (taśma dziurkowana, karty dziurkowane) lub z klawiatury maszyny do pisania i przesyła ją do pamięci.

Urządzenie wyjściowe jest przeznaczone do wprowadzania zawartości pamięci operacyjnej (wyników obliczeń lub programów). W tym celu urządzenie pobiera informację z pamięci (ewentualnie za pośrednictwem JAL) i zapisuje na nośniku zewnętrznym. Pracą urządzeń zewnętrznych steruje program za pomocą specjalnych instrukcji, tzw. instrukcji wejścia-wyjścia.

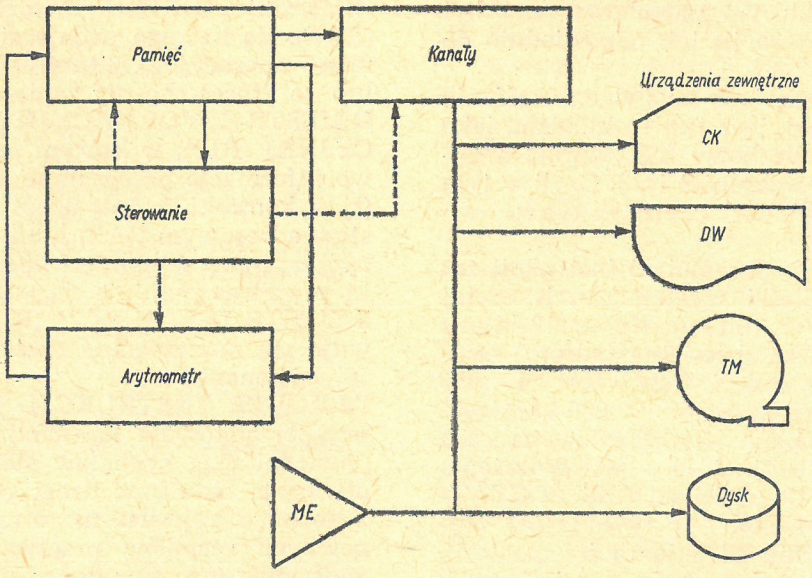
W początkowym okresie rozwoju maszyn cyfrowych bloki o nazwach wejście i wyjście były oddzielnymi zespołami i każdy z nich przeprowadzał przesyłanie informacji tylko w jednym kierunku. Natomiast we współczesnych maszynach cyfrowych wprowadza się zespoły, zwane k a n a ł a m i w e j ś c i a - w y j ś c i a (rys. 2.23), które mogą zarówno wprowadzać dane z urządzeń zewnętrznych (np. z czytnika kart) do pamięci operacyjnej, jak i wyprowadzać dane na określone urządzenia (np. drukarka wierszowa, pamięć taśmowa itp.).

2.7.5. Działanie maszyny cyfrowej

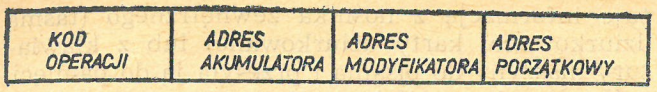
Działanie maszyny cyfrowej wyjaśnimy na podstawie uproszczonego schematu strukturalnego umownej maszyny, przedstawionego na rys. 2.25. Każda maszyna cyfrowa działa w następujący sposób: po wprowadzeniu programu do pamięci operacyjnej operator wpisuje do tzw. licznika instrukcji adres miejsca pamięci, w którym znajduje się pierwsza instrukcja programu do wykonania, i uruchamia maszynę, na przykład przyciskiem START. Maszyna rozpoczyna automatyczne działanie polegające na odczytywaniu i wykonywaniu kolejnych instrukcji, który to proces trwa aż do napotkania instrukcji STOP programu lub zatrzymania maszyny przez operatora, np. przyciskiem STOP.

Załóżmy, że w pamięci operacyjnej umownej maszyny zapisane są dane i instrukcje mające strukturę pokazaną na rys. 2.24.

Część KOD OPERACJI (np. 8-bitowa) instrukcji określa, jaką operację (funkcję) powinna maszyna



Rys. 2.23. Schemat blokowy prostej współczesnej EMC



Rys. 2.24. Format instrukcji przykładowej maszyny

wykonać (np. dodawanie, mnożenie, itp.). Każdej kombinacji bitów (tj. 256 kombinacji) w tym polu przypisana jest na stałe pewna grupa czynności (operacji). Wykaz tych czynności stanowi l i s t ę i n s t r u k c j i (lub rozkazów) danej maszyny.

Część ADRES AKUMULATORA służy do określenia adresu jednego z wewnętrznych rejestrów, zwanych akumulatorami lub rejestrami ogólnymi. W maszynie umownej przyjęto 16 takich akumulatorów oznaczonych odpowiednio przez RO, R1, ... R15. Zatem część instrukcji ADRES AKUMULATORA może mieć 4 bity i określać numery: 0, 1, 2, ... 15. W akumulatorach R (32-bitowych w przypadku maszyn JS EMC) przechowuje się często jeden z argumentów operacji przed operacją, zaś po operacji wynik tej operacji.

Część ADRES POCZĄTKOWY służy na ogół do określenia adresu komórki pamięci operacyjnej. Komórki pamięci są ponumerowane: 0, 1, 2, ... N (gdzie N-dowolna liczba całkowita). We współczesnych maszynach cyfrowych pojemność pamięci operacyjnej waha się przeważnie w granicach od 32 K do 1024 K słów (np. 32-bitowych). Czyli od 128 K do 4096 K znaków alfanumerycznych (gdzie K=1024). W komórce pamięci, której adres określa N, przechowuje się na ogół drugi argument operacji (a czasem także wynik po danej operacji).

Jeśli zatem przez symbol $\oplus$ oznaczymy dowolną operację, określoną częścią KOD OPERACJI omawianej instrukcji, zaś przez r i n odpowiednio zawartości akumulatora R i komórki o adresie N, to instrukcja umownej maszyny cyfrowej zawiera niezbędny opis (dla bloku sterowania maszyny) operacji do wykonania.

Przeważnie operacje te mają postać:

$$r \oplus n \rightarrow R \qquad \text{lub } r \oplus n \rightarrow N$$

Powyższy zapis oznacza wykonanie operacji typu $\oplus$ (np. dodawanie, mnożenie, sumowanie logiczne itp.) na argumentach liczbowych r i n oraz przesłanie wyniku tej operacji do akumulatora R (tzn. w miejscu, w którym przechowywany był pierwszy argument operacji) lub odpowiednio do komórki N.

W niektórych instrukcjach omawiana część nie określa adresu komórki, lecz jest traktowana jako pewien PARAMETR liczbowy (np. wprost argument operacji, liczba przesunięć itp.). Czyli w tym przypadku można wykonać operację typu: $r \oplus N \rightarrow X$.

Wielkość adresu (lub parametru) instrukcji zależy oczywiście od ilości występujących w tej części cyfr dwójkowych (bitów). Np. w 12-bitach można zapisać wielkości dziesiętne (adresy) od 0 do 4095. Natomiast pamięci współczesnych maszyn klasy średniej przekraczają 100 000 komórek. Zatem powstaje problem rozszerzenia zakresu adresów pamięci operacyjnej, lecz bez wydłużenia tej części instrukcji, tj. części ADRES POCZĄTKOWY, gdyż powodowałoby to wydłużenie długości instrukcji (zależnie od pojemności pamięci).

Do rozszerzenia zakresu adresów pamięciowych służy właśnie część ADRES MODYFIKATORA (MOD) instrukcji, która przeważnie ma tylko 2, 3 lub 4 bity, zatem można określać cztery, osiem lub szesnaście różnych adresów (rejestrów). Część MOD określa adres wyróżnionego rejestru maszyny zwanego modyfikatorem. W modyfikatorach pamięta się przeważnie wielkości tej samej długości co słowo maszynowe (w EMC Jednolitego Systemu 24 bity). Często rolę modyfikatorów spełniają akumulatory (lub inaczej rejestry ogólne) lub tylko część z tych akumulatorów. Na przykład w maszynie ODRA 1305 modyfikatorami są tylko trzy akumulatory (R1, R2 i R3) z ośmiu istniejących (RO, R1, ..., R7) zaś w maszynach Jednolitego Systemu modyfikatorami mogą być wszystkie rejestry ogólne z wyjątkiem pierwszego rejestru (tzn. w roli rejestrów modyfikujących mogą wystąpić rejestry R1, R2, ..., R15).

Główną funkcją modyfikacji jest rozszerzenie adresu początkowego komórki pamięci przez dodanie do pierwotnego adresu N zawartości modyfikatora M (oznaczonej przez m). Zatem suma ta, tj. $N+m$, może mieć np. do 24 bitów znaczących i określać adresy z zakresu od 0 do 16777215 (aczkolwiek adres początkowy N miał np. tylko 12 bitów). Tak zmieniony adres pamięciowy nazywa się adresem zmodyfikowanym. Często w jednej instrukcji może wystąpić kilka modyfikacji, z których każda powoduje dalsze zmiany adresu pamięciowego.

2.7.6. Cykl pobrania i wykonania instrukcji

Dotychczas omawiano strukturę instrukcji maszyny umownej oraz główną funkcję poszczególnych części instrukcji. Poniżej opisano strukturę rejestrową naszej „szkoleniowej" maszyny oraz ważniejsze powiązania pomiędzy rejestrami.

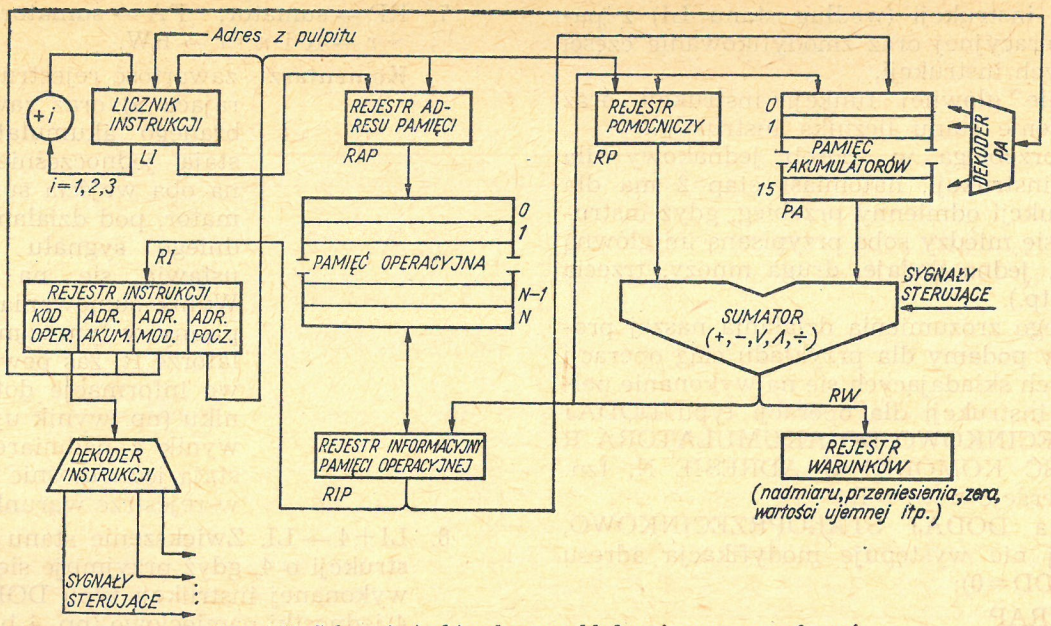
LICZNIK INSTRUKCJI (LI) jest to rejestr, w którym pamięta się adres komórki zawierającej kolejną instrukcję do wykonania. Zawartość LI jest przekazywana do rejestru adresu pamięci (patrz rys. 2.25) w celu wyboru określonej komórki pamięci, w której znajduje się dana instrukcja. Stan licznika instrukcji może być powiększony o jeden (lub o pewną stałą opisującą długość wykonywanej instrukcji liczoną np. w bajtach, półsłowach lub słowach).

REJESTR ADRESU PAMIĘCI OPERACYJNEJ (RAP), do którego podaje się adres komórki pamięci operacyjnej, której stan chcemy odczytać lub do której chcemy zapisać nową wielkość.

REJESTR INFORMACYJNY PAMIĘCI OPERACYJNEJ (RIP), w którym przechowuje się czasowo informacje przeczytane z pamięci operacyjnej (tj. z komórki, której adres znajduje się w rejestrze adresowym RAP) lub informacje, które mają być zapisane w pamięci operacyjnej (w komórce, której adres znajduje się w RAP).

REJESTR INSTRUKCJI (RI), w którym przechowuje się poszczególne części instrukcji w czasie jej wykonania.

DEKODER INSTRUKCJI jest to układ, który przyporządkowuje każdemu kodowi operacji odpowiedni ciąg sygnałów sterujących. Sygnały te otwierają określone drogi przesyłania informacji z jednego rejestru na drugi lub sterują pracą pewnych zespołów przetwarzających informacje wejściowe (np. sumator, półsumator, układ przesuwania itp.).



Rys. 2.25. Schemat strukturalny przykładowej maszyny cyfrowej

Często we współczesnych maszynach cyfrowych jeden początkowy sygnał sterowania inicjuje cały ciąg nowych układów sygnałów sterujących. Każdy nowy układ sygnałów sterujących jest powodowany pojawieniem się nowej tzw. mikroinstrukcji (lub mikrorozkazu). Mikroinstrukcje tworzą mikroprogramy sterujące, które są przechowywane w pamięci mikroprogramów maszyny (PAM). Zawartość PAM jest na ogół niezniszczalna, wówczas taka PAM nazywa się stałą (jak np. w maszynach R-32 i ODRA 1305), lecz coraz częściej stosuje się ładowalne pamięci mikroprogramów. Taka pamięć mikroprogramów pozwala na dynamiczną zmianę sterowania maszyny, co prowadzi do zmiany funkcji maszyny.

PAMIĘĆ AKUMULATORÓW (R0, R1, . . . , R15) składa się z 16 rejestrów, w których przechowuje się pewne robocze argumenty operacji, wyniki robocze oraz modyfikatory (indeksy) programu. W czasie zapisu lub odczytu informacji z pamięci akumulatorów odwoływania dotyczą tego akumulatora, którego adres podaje się na wejściu dekodera pamięci akumulatorów (tj. dekodera PA).

REJESTR POMOCNICZY (RP), w którym w czasie operacji przechowuje się jeden z początkowych argumentów operacji, zaś po operacji można też przechować wynik operacji (np. otrzymany z sumatora).

SUMATOR jest to układ, w którym wykonuje się różne proste operacje arytmetyczne (np. $p+r$, $p-r$, $p+r+1$) oraz operacje logiczne (np. $p \vee r$, $p \wedge x$, $p \div x$) na argumentach p i r podawanych na wejściu sumatora odpowiednio z rejestru pomocniczego RP i wybranego rejestru pamięci akumulatorów PA. Wybór operacji w sumatorze dokonywany jest za pomocą podawanych do niego sygnałów sterujących.

Poniżej wymieniono drogi przesyłań pomiędzy rejestrami naszej umownej maszyny. Z rysunku 2.25 wynika, że istnieją następujące przesyłania międzyrejestrowe:

1. LI do RAP (licznik instrukcji do rejestru adresu pamięci)
2. LI do półsumatora (danie jedynki do LI lub innej stałej wielkości)
3. Półsumator do LI (zapis do LI zmodyfikowanego stanu)
4. RAP do PAO (przekazanie adresu z RAP do układów deszyfrujących pamięci operacyjne)
5. PAO do RIP (odczyt informacji z pamięci operacyjnej)
6. RIP do RI (i deszyfracja instrukcji)
7. RI do RP (adres instrukcji do RP)
8. RI do RAP (adres instrukcji do RAP)
9. RI do dekodera PA (adresy R — instrukcji do dekodera PA)
10. RIP do PA (zapis stanu rejestru RI do wybranego akumulatora)
11. RIP do RP (zapis stanu rejestru RI do rejestru pomocniczego)
12. PA do sumatora (przesłanie zawartości wybranego akumulatora do prawego wejścia sumatora)
13. RP do sumatora (przesłanie stanu rejestru RP do lewego wejścia sumatora)
14. Sumator do RIP (przesłanie wyniku z sumatora do rejestru informacyjnego pamięci, czyli zapis do PAO)
15. Sumator do RP (przesłanie wyniku z sumatora do rejestru RP)
16. Sumator do PA (zapis wyniku z sumatora do wybranego rejestru pamięci akumulatorowej)
17. Sumator do RW (zapis warunku wyniku z sumatora)

Wykonanie bardziej złożonej operacji można rozpatrywać jako ciąg wyżej opisanych przesyłań międzyrejestrowych oraz operacji elementarnych wykonanych w sumatorze oraz w innych układach przetwarzających (np. półsumatorach, licznikach, w układach przesuwania itp.). W rzeczywistości operacje zadawane przez instrukcje maszynowe (np. dodawanie lub mnożenie dwóch liczb) są traktowane jako operacje złożone i ich wykonanie składa się z wykonania pewnej ilości takich operacji elementarnych.

Wszystkie czynności związane z pobraniem jednej instrukcji z pamięci operacyjnej i z jej wykonaniem tworzą tzw. cykl instrukcji danej instrukcji. Cykl wykonania każdej instrukcji można umownie podzielić na dwa główne funkcjonalne etapy:

1) pobranie instrukcji (według stanu LI) z pamięci operacyjnej oraz zmodyfikowanie części adresowych instrukcji,
2) wykonanie głównej funkcji instrukcji oraz powiększenie stanu licznika instrukcji.

Etap 1 przebiega w sposób jednakowy dla wszystkich instrukcji, natomiast etap 2 ma dla każdej instrukcji odmienny przebieg, gdyż instrukcje różnią się między sobą przypisaną im główną funkcją (np. jedna dodaje, druga mnoży, trzecia porównuje itp.).

Dla lepszego zrozumienia działania naszej prostej maszyny podamy dla przykładu ciąg operacji elementarnych składających się na wykonanie pełnego cyklu instrukcji dla operacji typu DODAJ STAŁOPRZECINKOWO DO AKUMULATORA R ZAWARTOŚĆ KOMÓRKI O ADRESIE N, tzn. wykonaj operację $n+r \rightarrow R$.

A. Instrukcja DODAJ STAŁOPRZECINKOWO, w której nie występuje modyfikacja adresu (część MOD=0):

1. LI → RAP
   Komentarz: blok sterowania maszyny generuje odpowiednie sygnały sterujące, które powodują przesłanie stanu LI do rejestru adresowego pamięci operacyjnej (RAP)
2. PAO → RIP oraz RIP → RI
   Komentarz: zawartość komórki, której adres określa RAP, zostaje odczytana z PAO do rejestru informacyjnego pamięci (RIP) w pierwszym kroku, a następnie zostaje przesłana do rejestru instrukcji (RI) i dekodowana.
3. Jeśli część MOD=0, to przechodzi się do punktu 4 przypadku A; inaczej przechodzi się do punktu 3a przypadku B.
4. RI (część ADRES POCZĄTKOWY) → RAP
   Komentarz: z rozszyfrowania (w RI) pobranej instrukcji wiemy, że jest to instrukcja dodawania stałoprzecinkowego typu $n+r \rightarrow R$. Zatem należy adres komórki N zapisać do RAP i zapoczątkować odczyt z PAO.
5. PAO (N) → RIP oraz RIP → RP
   Komentarz: zawartość komórki N zostaje przesłana, za pośrednictwem rejestru informacyjnego pamięci (RIP), do rejestru pomocniczego RP.
6. RI (część ADRES AKUMULATORA) → dekoder PA oraz wybór akumulatora R.
   Komentarz: ponieważ drugim argumentem operacji DODAJ jest zawartość akumulatora R, to należy adres akumulatora (zapisany w części ADRES AKUMULATORA instrukcji) skierować do dekodera, który wybierze jeden z 16 akumulatorów; po wykonaniu tej operacji na wyjściu pamięci akumulatorów będzie eksponowany stan wybranego akumulatora.
7. RP → sumator, PA → sumator oraz $n+r \rightarrow R$ i $n+r \rightarrow RW$.
   Komentarz: zawartość rejestru RP (zawierająca n) oraz zawartość wybranego akumulatora X zostają jednocześnie podawana na oba wejścia sumatora. Sumator, pod działanie odpowiedniego sygnału sterującego, ustawia się na dodawanie. Wynik sumowania zostaje zapisany w tym samym akumulatorze R, zaś pewne dodatkowe informacje dotyczące wyniku (np. wynik ujemny, zero, wynik z nadmiarem itp.) zostają jednocześnie odnotowane w rejestrze warunków RW.
8. LI+4 → LI. Zwiększenie stanu licznika instrukcji o 4, gdyż przyjmuje się, że długość wykonanej instrukcji typu DODAJ wynosi 4 jednostki pamięciowe (np. 4 bajty). W ten sposób w LI zapisaliśmy adres następnej instrukcji do wykonania, tzn. instrukcji zapisanej w PAO bezpośrednio za rozpatrywaną instrukcją DODAJ. Na tym kończy się cykl instrukcji typu DODAJ STAŁOPRZECINKOWO.

B. Instrukcja DODAJ STAŁOPRZECINKOWO, w której występuje modyfikacja adresu (MOD≠0).

Przebieg operacji elementarnych 1 i 2 jest identyczny z przebiegiem tych operacji w przypadku A.

Operacja 3 spowoduje w naszym przypadku przejście do punktu 3a i 3b, gdzie zostanie zrealizowana modyfikacja adresu pierwotnego N omawianej instrukcji.

3a. RI (część ADRES POCZĄTKOWY) → RP i RI (część ADRES MOD) → dekoder PA
   Komentarz: adres początkowy N (np. 12-bitowy) zostaje skierowany do rejestru pomocniczego RP w celu przygotowania modyfikacji; jednocześnie do dekodera PA podaje się adres modyfikatora (tj. jeden z rejestrów R1, R2, . . . , R14 lub R15).

3b. RP → sumator, PA → sumator oraz $N+m \rightarrow RIP$ i RIP → RI.
   Komentarz: w sumatorze następuje dodanie do adresu początkowego N zawartości wybranego modyfikatora R (którego zawartość oznaczono przez m) oraz przesłanie wyniku sumy do części adresowej rejestru instrukcji RI (tzn. operacja ta nie zmienia stanu pozostałych części rejestru instrukcji).

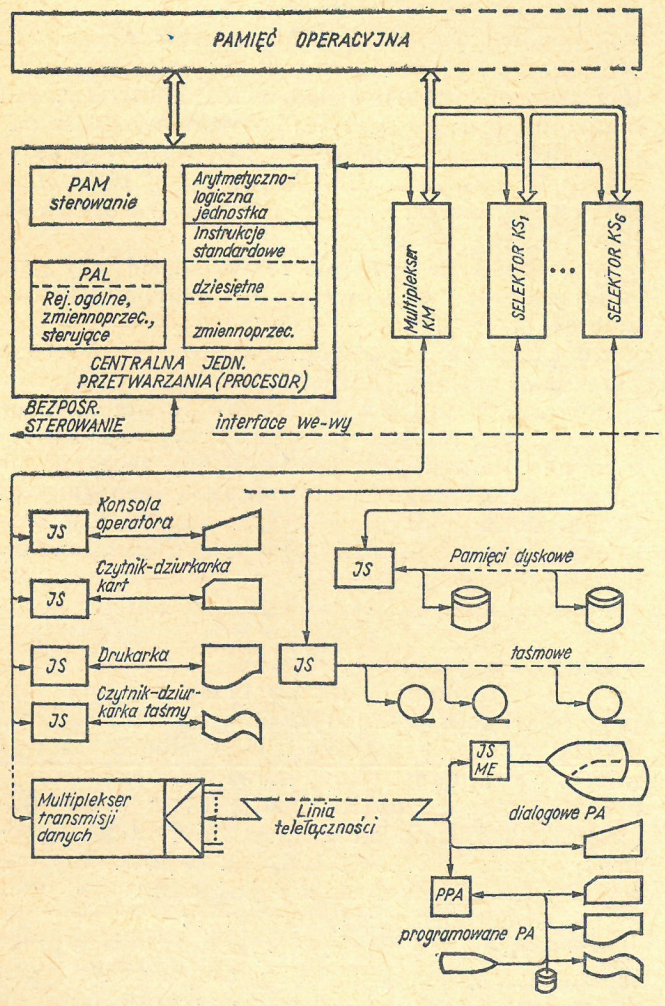
Dalszy przebieg wykonania instrukcji w przypadku B jest jednakowy z przebiegiem przypadku A (punkty 4, 5, 6, 7 i 8), z tą różnicą, że w przypadku B argument pamięciowy operacji DODAJ STAŁOPRZECINKOWO znajduje się nie w komórce o adresie N, lecz w komórce o adresie $N+m$ (gdzie m jest zawartością modyfikatora, którego adres określa część MOD instrukcji).

3. Podstawowe cechy architektury logicznej

3.1. Podstawowa struktura systemu

Jednolity System EMC ma budowę modułową, co pozwala na elastyczne zestawienie różnorodnych konfiguracji w zależności od potrzeb i przeznaczenia. Wydajność poszczególnych konfiguracji jest różna i zależna od szybkości centralnej jednostki przetwarzania (procesora), na której jest oparta dana konfiguracja, od szybkości i pojemności pamięci operacyjnej, a także od szybkości i ilości kanałów urządzeń zewnętrznych.

Elementy funkcjonalne Jednolitego Systemu powiązane są ze sobą w sposób pokazany na rys. 3.1. Podstawowymi elementami JS EMC są: pamięć operacyjna, centralna jednostka przetwarzania, dwa rodzaje kanałów — multiplekser i selektor oraz urządzenie wejścia-wyjścia podłączone do kanałów za pośrednictwem własnych jednostek sterujących (JS).



Rys. 3.1. Ogólna struktura JS EMC

Maszyny Jednolitego Systemu mogą także współpracować ze sobą poprzez:
- — połączenia dwóch jednostek centralnych wykorzystujących wspólną pamięć operacyjną,
- — połączenia dwóch maszyn za pomocą adaptera kanał-kanał,
- — połączenia dwóch maszyn za pomocą bloku bezpośredniego sterowania,
- — połączenia kilku maszyn mających dostęp do wspólnej zewnętrznej pamięci.

Przedstawiona na rys. 3.1 ogólna struktura logiczna nie odzwierciedla realizacji konkretnej maszyny, lecz podaje jedynie ogólną koncepcję struktury JS EMC oraz powiązań funkcjonalnych pomiędzy elementami systemu. Struktury poszczególnych modeli JS EMC mogą być różne. Określone środki, np. kanały lub wyspecjalizowane arytmometry, mogą stanowić samodzielne i autonomiczne urządzenia, mogą być w pełni zintegrowane z innymi środkami albo też wykorzystywać dany sprzęt wspólnie z innymi środkami. W każdej maszynie występują określone środki techniczne, które są traktowane jako standardowe (minimalne lub obowiązkowe) dla wszystkich jej konfiguracji. Pozostałe natomiast środki są traktowane jako dodatkowe i na ogół rozszerzają możliwości lub wydajność maszyny. Minimalny zbiór środków technicznych pozwala w praktyce na wykonanie programów użytkowych pod kontrolą systemu operacyjnego.

W większości maszyn Jednolitego Systemu przyjmuje się następujące środki standardowe:
- — środki wykonania instrukcji standardowych (tj. operacji arytmetyki stałoprzecinkowej, logicznych, przesuwań, sterujących, pakowania i rozpakowywania liczb dziesiętnych, translacji i operacji wejścia-wyjścia),
- — środki sterowania procesem obliczeniowym,
- — rejestry (lub akumulatory) ogólne,
- — minimalna wielkość pamięci operacyjnej (różna dla różnych modeli),
- — jeden kanał bajtowego multipleksera,
- — jeden kanał selektorowy,
- — interface wejścia-wyjścia,
- — pulpit sterowania maszyną.

Natomiast jako środki dodatkowe przyjmuje się:
- — środki wykonywania instrukcji arytmetyki dziesiętnej,
- — rejestry zmiennoprzecinkowe oraz środki wykonywania instrukcji arytmetyki zmiennoprzecinkowej,

— ochrona pamięci operacyjnej,
— zegar czasu realnego i czasomierz interwałowy,
— środki bezpośredniego sterowania (do przekazywania lub odbierania danych bezpośrednio do urządzenia zewnętrznego lub do drugiego procesora),
— adapter kanał-kanał (do organizacji wymiany danych pomiędzy kanałami),
— środki tworzenia systemów wielomaszynowych.

Brak określonych środków dodatkowych w danej maszynie nie oznacza braku zgodności programowej z innymi maszynami, aczkolwiek program wykorzystujący określone środki dodatkowe może nie zostać zaakceptowany przez maszynę, w której brak jest tych środków.

3.1.1. Pamięć operacyjna (PAO)

Pamięć operacyjna służy do przechowywania zarówno przetwarzanych danych jak i instrukcji. Najmniejszą jednostką informacji adresowaną w pamięci jest bajt (8 bitów + bit kontrolny).

Cykl pracy, pojemność i długość czytanych lub pisanych informacji pamięci jest różna dla różnych typów maszyn. Maksymalna pojemność pamięci bezpośrednio adresowanej wynosi 16 777 216 bajtów.

Miejsca pamięci są adresowane 24-bitową liczbą poczynając od 0.

3.1.2. Centralna jednostka przetwarzania (CJP) – procesor

Centralna jednostka przetwarzania wykonuje następujące funkcje:
— steruje wykonywaniem instrukcji przechowywanych w pamięci maszyny,
— przeprowadza automatyczną kontrolę poprawności działań,
— kieruje działaniem urządzeń wejścia-wyjścia.

CJP wyposażona jest w specjalne bloki, takie jak:
— arytmetyczno-logiczna jednostka, w której wykonuje się różne działania określone przez instrukcje maszyny,
— pamięć mikroprogramów (PAM), zawierającą logikę wszystkich operacji wykonywanych przez CJP w postaci operacji elementarnych (mikrooperacji),
— pamięć lokalna (PAL), jest szybką pamięcią roboczą zawierającą przeważnie 64 słowa 4-bajtowe, w których przechowuje się rejestry ogólne, zmiennoprzecinkowe, sterujące i robocze maszyny.

3.1.3. Kanały

Kanały służą do wprowadzania i wyprowadzania danych z lub do współpracujących urządzeń zewnętrznych. Kanały maszyny mogą przeważnie pracować równolegle z obliczeniami realizowanymi w CJP. Kanały współpracują z urządzeniami wejścia-wyjścia w oparciu o standardowe zasady współpracy.

Maszyna może być wyposażona w dwa typy kanałów:
— multiplekser (KM) współpracujący jednocześnie z 256 wolnymi urządzeniami wejścia-wyjścia w oparciu o zasadę podziału czasu; każda CJP może współpracować tylko z jednym KM;
— selektor (KS) umożliwiający szybkie, blokowe przesyłanie danych tylko z jednym z dołączonych do KS szybkich urządzeń wejścia-wyjścia w określonym przedziale czasu.

Do KS można dołączyć maksymalnie 256 urządzeń. Każda CJP może współpracować maksymalnie z 6 kanałami selektorowymi. Ilość kanałów selektorowych współpracujących z CJP zależy od typu maszyny.

3.1.4. Urządzenia wejścia-wyjścia

Każda maszyna może być wyposażona w dowolny zestaw urządzeń zewnętrznych, w skład którego mogą wchodzić: czytniki i dziurkarki kart, czytniki i dziurkarki taśmy papierowej, drukarki wierszowe, elektryczne maszyny piszące, pamięci dyskowe i jednostki sterujące (JS) pamięciami dyskowymi, pamięci taśmowe i JS pamięciami taśmowymi, pamięci bębnowe i JS pamięciami bębnowymi, urządzenia teleprzetwarzania (punkty abonenckie), monitory ekranowe i inne urządzenia.

3.1.5. Postać danych

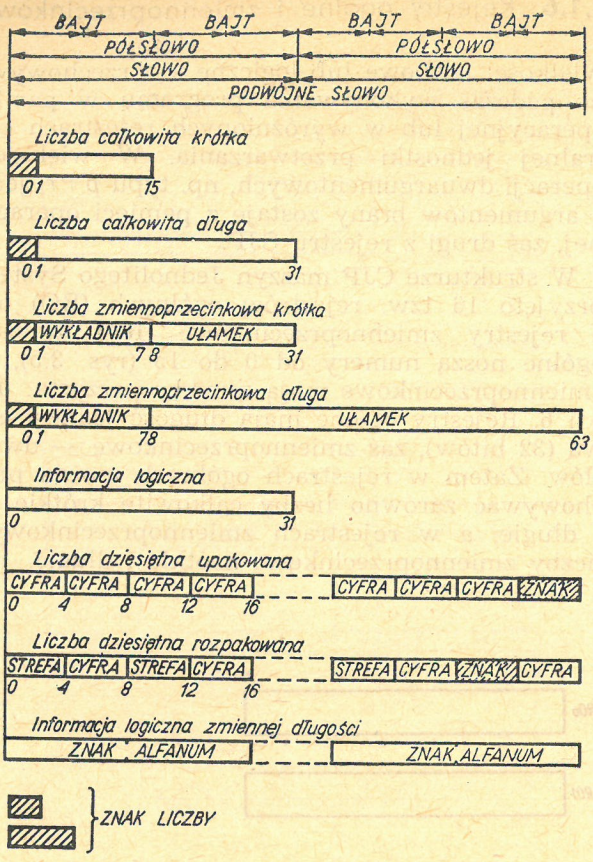
Podstawową jednostką informacji w maszynach Jednolitego Systemu jest bajt. Bajt zawiera 8 bitów informacyjnych i jeden bit kontrolny. Bit kontrolny jest automatycznie tworzony przez odpowiednie układy dla wszystkich bajtów przesyłanych w maszynie. Bit kontrolny jest bitem nieparzystości. Grupa n kolejnych bajtów nazywa się polem o długości n.

W Jednolitym Systemie rozróżnia się trzy pola stałej długości: półsłowo — tworzone przez dwa kolejne bajty, słowo — tworzone przez dwa kolejne półsłowa i podwójne słowo — tworzone przez dwa kolejne słowa.

Pola stałej długości nie mogą być umieszczone w dowolnym miejscu pamięci. Przyjmuje się regułę, że pola stałej długości powinny zaczynać się od adresu będącego wielokrotnością jego długości (dzięki temu założeniu przyśpieszone zostaje wykonanie instrukcji, w której występują argumenty o stałej długości).

Formaty danych jednolitego systemu przedstawia rys. 3.2.

Długość pól, do których odwołuje się instrukcja, są albo określane pośrednio przez kod operacji, jaką instrukcja wykonuje, albo są bezpośrednio podane w samej instrukcji. Gdy długość ta wynika z rodzaju instrukcji, mówi się, że dane wykorzystywane przez instrukcję mają stałą długość. Tego rodzaju dane mogą mieć długość jednego, dwóch, czterech lub ośmiu bajtów. Gdy natomiast długość danych nie jest określona przez kod operacji instrukcji, lecz przez parametr instrukcji, mówi się, że dane wykorzystywane przez tę instrukcję mają zmienną długość. W takim przypadku długość danej może wynosić od 1 do 256 bajtów.



Rys. 3.2. Formaty danych

Kodem wewnętrznym systemu jest *Extended Binary-Coded Decimal Interchange Code* (EBCDIC) lub kod ISO dla 7-ścieżkowej taśmy papierowej (odpowiadający *USA Standard Code for Information Interchange USA SCII)* w zależności od definicji podanej przez program.

Instrukcje maszynowe powodują wykonanie różnych operacji arytmetycznych i logicznych. W tych operacjach wyróżnia się następujące wielkości: liczby całkowite krótkie, liczby całkowite długie, liczby zmiennoprzecinkowe krótkie, liczby zmiennoprzecinkowe długie, liczby dziesiętne i informacje logiczne (rys. 3.2).

Liczba całkowita krótka zajmuje jedno półsłowo, gdzie 15 bitów określa część znaczącą liczby, zaś jeden bit określa znak liczby.

Liczba całkowita długa zajmuje jedno słowo i przedstawiona jest w postaci jednego bitu znaku oraz 31-bitowej części znaczącej.

Liczby stałoprzecinkowe dodatnie przedstawione są w normalnym kodzie binarnym z bitem znaku równym 0. Natomiast liczby stałoprzecinkowe ujemne są przedstawione jako uzupełnienia do liczby 2 z bitem znaku równym 1.

Wielkości całkowite (stałoprzecinkowe) mogą być używane zarówno do działań na danych liczbowych jak i na adresach.

Liczba zmiennoprzecinkowa krótka zajmuje jedno słowo. Jej część ułamkowa (mantysa) składa się z bitu znaku i 24-bitowej wartości bezwzględnej. Wykładnik składa się z 7 bitów.

Liczba zmiennoprzecinkowa długa zajmuje jedno podwójne słowo. Jej część ułamkowa składa się z jednego bitu znaku i 56-bitowej wartości bezwzględnej. Wykładnik składa się z 7 bitów.

Ułamek liczby zmiennoprzecinkowej jest wyrażony w cyfrach szesnastkowych (heksadecymalnych), z których każda składa się z czterech bitów i reprezentuje wartość od 0 do 15. Przecinek ułamka znajduje się bezpośrednio po lewej stronie najbardziej znaczącej cyfry ułamka. Ułamek liczby zmiennoprzecinkowej krótkiej składa się z 6 cyfr heksadecymalnych, zaś liczby długiej z 14 takich cyfr.

Wykładnik jest przedstawiony w normalnym kodzie binarnym przesuniętym o 64 i ma zakres od -64 do $+63$ odpowiadający wartościom binarnym od 0 do 127.

Liczba dodatnia i ujemna ma jednakową wartość bezwzględną. Liczba jest dodatnia, jeśli ma bit znaku równy 0, a ujemna, gdy bit znaku jest równy 1.

Aby otrzymać właściwą wartość liczby Q odpowiadającą liczbie zmiennoprzecinkowej, należy 16 podnieść do potęgi równej wykładnikowi (w) i pomnożyć przez ułamek (U):

$$Q = 16^w \cdot U$$

Liczba zmiennoprzecinkowa uważana jest za znormalizowaną, jeśli najbardziej znacząca cyfra szesnastkowa ułamka jest różna od zera, czyli:

$$U \geqslant \frac{1}{16}$$

Normalizacja liczby zmiennoprzecinkowej polega na przesuwaniu w lewo cyfr szesnastkowych ułamka, aż do różnej od zera najbardziej znaczącej cyfry, oraz zmniejszeniu wykładnika o liczbę przesunięć cyfr szesnastkowych. Jeśli ułamek liczby jest równy zero, to liczba nie zostaje znormalizowana.

Zmiennoprzecinkowe zero jest przedstawione w postaci wykładnika równego zero oraz ułamka równego plus zero.

Wartość bezwzględna liczby zmiennoprzecinkowej znormalizowanej znajduje się w przedziale

$$16^{-64} \leqslant |Q| \leqslant (1-16^{-6}) \cdot 16^{63}$$

dla liczb zmiennoprzecinkowych krótkich oraz w przedziale

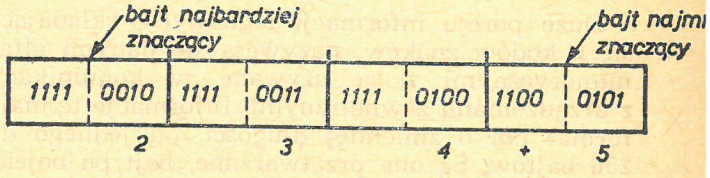
$$16^{-64} \leqslant |Q| \leqslant (1-16^{-14}) \cdot 16^{63}$$

dla liczb zmiennoprzecinkowych długich, co w przybliżeniu odpowiada

$$10^{-78} < |Q| < 10^{+75}$$

Liczba dziesiętna jest traktowana jako liczba całkowita składająca się z wartości bezwzględnej i znaku liczby. Zajmuje ona pole składające się z 1 do 16 bajtów. Cyfry oraz znak liczby są kodowane binarnie za pomocą 4 bitów. Liczby dziesiętne mogą być przedstawione w postaci:

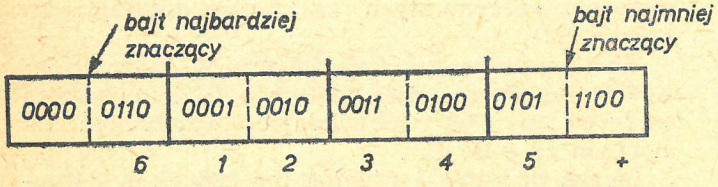
— liczb dziesiętnych upakowanych (rys. 3.3), w których jeden bajt zawiera dwie



Rys. 3.3. Przykład rozpakowanej liczby dziesiętnej (+2345)

cyfry dziesiętne, z wyjątkiem prawego skrajnego bajtu pola, gdzie na prawo od cyfry umieszczony jest znak liczby.

— liczb dziesiętnych rozpakowanych (rys. 3.4), w których jeden bajt przedstawia jedną cyfrę alfanumeryczną. Cztery prawe pozycje bajtu zawierają cyfrę, cztery lewe tak zwaną strefę.



Rys. 3.4. Przykład upakowanej liczby dziesiętnej (+612345)

Cyfry dziesiętne 0—9 są kodowane dwójkowymi kombinacjami od 0000 do 1001 (tabela 3.1). Znak plus w kodzie EBCDIC generowany przez maszynę ma postać 1100, zaś znak minus ma postać 1101. W trakcie wykonywania działań na liczbach dziesiętnych maszyna identyfikuje kody 1010 i 1111 jako znak plus, a kod 1011 jako znak minus. Wynik działania posiada znak w kodzie EBCDIC. W czasie wykonywania rozpakowywania w miejsce strefy wstawia się kod 1111.

Tabela 3.1

Kombinacje przyporządkowane cyfrom dziesiętnym i znakom liczby

Cyfra dziesiętna	Kombinacja binarna	Znak liczby	Kombinacja binarna
0	0000	+	1010 (ASCII)
1	0001	—	1011 (ASCII)
2	0010	+	1100 (EBCDIC)
3	0011	—	1101 (EBCDIC)
		+	1110
4	0100	+	1111
5	0101		
6	0110		
7	0111		
8	1000		
9	1001		

Informacje logiczne. Informacje logiczne mogą być traktowane jako dane o stałej lub zmiennej długości. Występują one w takich operacjach jak: porównywanie, tłumaczenie, redagowanie oraz w operacjach działających na poszczególnych bitach (w tym również na bitach znakowych).

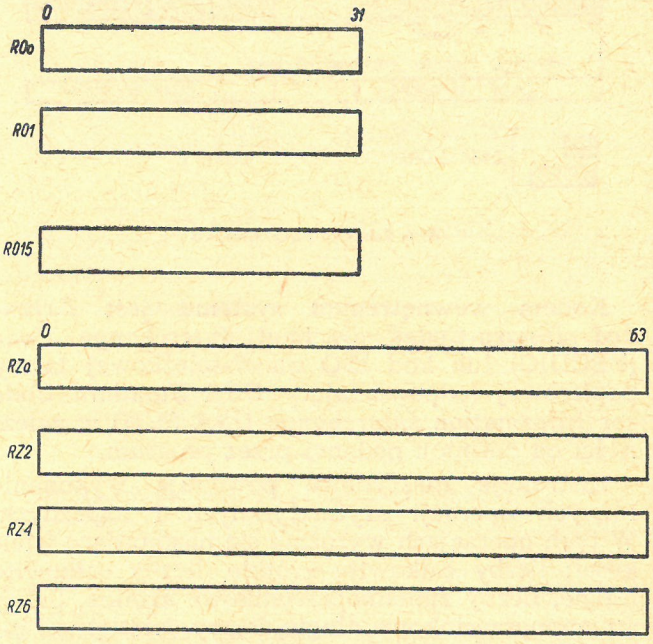
Informacje logiczne użyte jako operandy o stałej długości mogą składać się z jednego, dwóch, czterech bądź ośmiu bajtów.

Duże porcje informacji logicznych, składające się z kodów znaków, nazywają się danymi alfanumerycznymi i są używane w komunikacji z urządzeniami zewnętrznymi. Informacje te mają format pól o zmiennej długości, od jednego do 256 bajtów. Są one przetwarzane, bajt po bajcie, od lewej do prawej, bezpośrednio w pamięci.

3.1.6. Rejestry ogólne i zmiennoprzecinkowe

Wielkości liczbowe lub logiczne są przechowywane podczas wykonywania programu w pamięci operacyjnej lub w wyróżnionych rejestrach centralnej jednostki przetwarzania. W większości operacji dwuargumentowych, np. typu $p+r$, jeden z argumentów brany zostaje z pamięci operacyjnej, zaś drugi z rejestru CJP.

W strukturze CJP maszyn Jednolitego Systemu przyjęto 16 tzw. rejestrów ogólnych (RO) oraz 4 rejestry zmiennoprzecinkowe (RZ). Rejestry ogólne noszą numery od 0 do 15 (rys. 3.5), zaś zmiennoprzecinkowe mają numery parzyste: 0, 2, 4 i 6. Rejestry ogólne mają długość jednego słowa (32 bitów), zaś zmiennoprzecinkowe — dwóch słów. Zatem w rejestrach ogólnych można przechowywać zarówno liczby całkowite krótkie jak i długie, a w rejestrach zmiennoprzecinkowych liczby zmiennoprzecinkowe krótkie i długie.



Rys. 3.5. Rejestry ogólne i zmiennoprzecinkowe

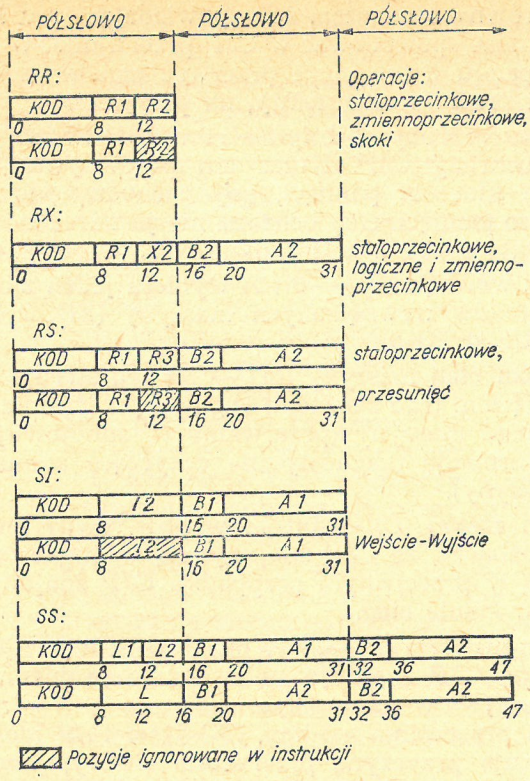
Rejestry ogólne spełniają nie tylko rolę akumulatorów w operacjach arytmetycznych i logicznych, lecz także rolę rejestrów indeksowych. Rejestry biorące udział w modyfikacjach indeksowych lub w operacjach są określane przez odpowiednie (4-bitowe) części rozkazu.

3.1.7. Formaty instrukcji (rozkazów)

W maszynach JS EMC instrukcje składają się z jednego, dwóch lub trzech półsłów, w zależności od ilości używanych instrukcji adresów:

instrukcja bezadresowa — jedno półsłowo,
instrukcja jednoadresowa — dwa półsłowa,
instrukcja dwuadresowa — trzy półsłowa.

Instrukcje JS mają pięć podstawowych postaci (rys. 3.6):



Rys. 3.6. Formaty instrukcji

RR — instrukcja postaci rejestr-rejestr,
RX — instrukcja postaci rejestr-pamięć indeksowana,
RS — instrukcja postaci rejestr-pamięć,
SI — instrukcja postaci pamięć-stały parametr,
SS — instrukcja postaci pamięć-pamięć.

Postać instrukcji związana jest z postacią danych, na których wykonywana jest operacja określona przez część KOD.

Znaczenie poszczególnych pól instrukcji jest następujące:

KOD — 8-bitowe pole pozwalające na rozróżnienie 256 instrukcji. Część KOD niezależnie od postaci instrukcji zajmuje zawsze pierwszy bajt. Dwa najbardziej znaczące bity kodu określają postać instrukcji, a tym samym definiują jego długość. Przyjęto poniższe przyporządkowanie:

Pozycje instrukcji		Postać instrukcji
0	1	
0	0	RR
0	1	RX
1	0	RS, SI
1	1	SS

R1, R2, R3 — 4-bitowe pole określające adres (0—15) rejestru ogólnego lub rejestru zmiennoprzecinkowego,
X2 — 4-bitowe pole określające adres (0—15) rejestru ogólnego zawierającego 24-bitowy indeks (tzw. modyfikacja indeksowa),
B1, B2 — 4-bitowe pole określające adres (0—15) rejestru ogólnego zawierającego adres bazowy (tzw. modyfikacja bazowa),
A1, A2 — 12-bitowy argument lub adres pierwotny instrukcji (0—4095), tzw. przesunięcie (oznaczone też przez D1, D2),
I2 — 8-bitowy stały parametr,
L1, L2 — 4-bitowa długość argumentu operacji liczona w bajtach (0—15),
L — 8-bitowa długość obu argumentów operacji liczona w bajtach (0—255).

Postać RR. W tych instrukcjach zawartość rejestru ogólnego określonego przez adres R1 stanowi pierwszy argument operacji. Zawartość natomiast rejestru ogólnego określonego przez adres R2 stanowi drugi argument operacji. Przykładem instrukcji postaci RR może być dodawanie stałoprzecinkowe typu: $|R1|+|R2|\rightarrow R1$, gdzie symbol $|R1|$ oznacza zawartość rejestru R1. Zatem w instrukcjach typu RR nie występują argumenty przechowywane w pamięci operacyjnej.

Postać RX. Zawartość rejestru ogólnego określonego przez adres R1 stanowi pierwszy argument operacji. Drugim argumentem jest wielkość przechowywana w pamięci operacyjnej. Dla uzyskania adresu efektywnego drugiego argumentu (E2), zawartość rejestrów ogólnych określonych adresami X2 i B2 jest dodawana do adresu pierwotnego instrukcji A2. Czyli $E2=|X2|+|B2|+A2$.
Grupa instrukcji postaci RX jest bogata i składa się z instrukcji działań stałoprzecinkowych, logicznych i zmiennoprzecinkowych.

U w a g a. Zawartość rejestru określonego przez pole X (odpowiednio B) zostaje dodana do A tylko wówczas, gdy $X\neq 0$ (odpowiednio $B\neq 0$). Zatem rejestrami modyfikującymi (indeksowymi) mogą być tylko rejestry ogólne od R01 do R015.

Postać RS. W tych instrukcjach adres efektywny drugiego argumentu określany jest przez $E2=|B2|+A2$. Postać RS używana przez instrukcje przesunięć, skoków oraz instrukcje typu: umieść wielokrotnie, pamiętaj wielokrotnie.

Postać SI. W tych instrukcjach zawartość rejestru ogólnego określonego adresem B1 jest dodawana do argumentu pierwotnego rozkazu A1 w celu uzyskania efektywnego adresu pierwszego argumentu operacji. Drugim argumentem jest stały parametr I2 (8-bitów), umieszczony bezpośrednio w instrukcji.

Postać SS. W tych instrukcjach oba argumenty są brane z pamięci operacyjnej. Zawartość rejestru ogólnego określonego przez B1 jest dodawana do argumentu pierwotnego A1 w celu uzyskania adresu skrajnego lewego bajtu pierwszego argumentu operacji.

Pole L1 określa liczbę dodatkowych bajtów położonych na prawo od bajtu określonego adresem pierwszego argumentu. W celu uzyskania adresu drugiego argumentu operacji, zawartość rejestru ogólnego określonego przez B2 jest dodawana do argumentu pierwotnego *A2*.

Pole L2 określa ilość dodatkowych bajtów, położonych na prawo od bajtu określonego adresem drugiego argumentu operacji. Jeśli oba argumenty mają taką samą długość, to długość ta jest określona przez 8-bitowy parametr L (dokładnie przez L+1).

3.2. Sterowanie wykonaniem programu

Przyjęty system sterowania wykonaniem programu zapewnia automatyczną reakcję maszyny w wyniku zaistnienia najrozmaitszych wewnętrznych zdarzeń. System sterowania pozwala na zorganizowanie jednoczesnego wykonania kilku programów (na zasadzie podziału czasu). W ten sposób zarówno procesor jak i pamięć operacyjna zostają odpowiednio dzielone pomiędzy przetwarzanymi programami. Operacje wejścia-wyjścia przebiegają na ogół równolegle z procesem obliczeń. Procesor uruchamia tylko operacje we-wy oraz reaguje na sygnały zakończenia tych operacji, natomiast przebieg przesyłania informacji odbywa się w pełni autonomicznie.

Poniżej opiszemy podstawowe zasady sterowania wykonaniem programu.

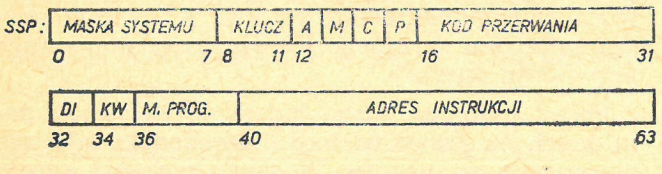
3.2.1. Słowo stanu programu (SSP)

Centralna jednostka przetwarzania wykonuje instrukcje programu sekwencyjnie, tzn. w tej kolejności, w jakiej występują w pamięci operacyjnej.

Adres miejsca, z którego jest pobrana do wykonania dana instrukcja, określony jest przez konwencjonalny licznik instrukcji. Licznik ten jest częścią podwójnego rejestru, zwanego słowem stanu programu (SSP). Po pobraniu instrukcji, adres zapisany w tym liczniku zwiększany jest o liczbę bajtów składających się na pobraną instrukcję, wskazując tym samym następną instrukcję do wykonania.

Zmiana kolejności wykonywania instrukcji może być spowodowana przez skok, zmianę stanu jednostki centralnej, przerwanie lub przez ręczną interwencję.

Słowo stanu programu zawiera różne informacje, które są potrzebne do właściwego wykonywania programu. Słowo SSP zawiera adres kolejnej instrukcji do wykonania, kod warunków oraz inne informacje sterujące. Zatem SSP steruje kolejnością wykonywania instrukcji oraz określa stan CJP, w którym znajduje się ona wykonując dany program. Aktualnie aktywne SSP nazywa się bieżącym SSP. W trakcie przerwania bieżące SSP jest zapisywane w określonym miejscu pamięci, co umożliwia jego późniejsze badanie. Wprowadzenie nowego SSP umożliwia zainicjowanie pracy nowego programu lub zmianę stanu CJP.



Rys. 3.7. Struktura podstawowego SSP

Struktura podstawowego SSP jest przedstawiona na rys. 3.7.

Poszczególne części (pola) dwusłowowego rejestru SSP zostały wykorzystane w sposób następujący:

Maska systemu — bity 0—7 w SSP przyporządkowane są sześciu kanałom we-wy (bity 0—6) i sygnałom zewnętrznym (bit 7). Bit maski równy 1 oznacza, że odpowiadające mu źródło może spowodować w CJP przerwanie. W przypadku 0 nie może przerwać działalności CJP i przerwania oczekują na przyjęcie.

Klucz ochrony — bity 8—11 w SSP tworzą klucz ochrony CJP. Klucz ten porównywany jest z kluczem pamięci, gdy zachodzą odwołania do pamięci. Gdy ochrona nie jest zrealizowana, bity 8—11 w SSP muszą być wyzerowane.

Kod wewnętrzny — bit A wybiera jeden z dwóch kodów wewnętrznych maszyny (EBCDIC lub USASCII-8).

Maska błąd maszyny (M) — bit ten steruje przyjęciem (lub zakazem) sygnału przerwania typu błąd maszyny (przerwanie to odnotowuje informacje diagnostyczne dotyczące stanu maszyny).

Stan **czekaj** (C) — bit 14 w SSP równy jedynce oznacza, że CJP znajduje się w stanie czekaj, a w przeciwnym przypadku CJP znajduje się w stanie **bieg.**

Stan **problem** (P) — gdy bit 15 w SSP jest jedynką, CJP znajduje się w stanie **problem;** inaczej — w stanie **supervisor.**

Kod przerwania — bity 16—31 w SSP identyfikują przyczynę przerwania we-wy, przerwania programowego, wywołaj supervisor lub przerwania zewnętrznego.

Kod długości instrukcji (DI) — bity 32 i 33 w SSP wskazują długość (w półsłowach) ostatnio interpretowanej instrukcji w momencie, gdy występuje przerwanie programowe lub typu wywołaj supervisor.

Kod warunków (KW) — dwa bity sterujące wykonaniem warunkowych instrukcji skokowych.

Maska programu — cztery bity sterujące zezwoleniem przerwań w przypadku zajścia jednej z czterech nieprawidłowości programu:
- — nadmiar stałoprzecinkowy,
- — nadmiar dziesiętny,
- — niedomiar wykładnika,
- — nieprawidłowa wartość.

Adres instrukcji — bity 40—63 w SSP tworzą adres lewego skrajnego bajtu następnej instrukcji do wykonania.

3.2.2. Stany centralnej jednostki przetwarzania

Działanie CJP zależy od stanu, w którym znajduje się ona wykonując dany program. Stan CJP określają cztery alternatywne stany programu: **problem-supervisor, czekaj-bieg, przerwanie zabronione-przerwanie dozwolone, stop-praca.** Stany te różnią się sposobem, w jaki oddziaływują na pracę CJP, w jaki są przełączane oraz sposobem ich sygnalizowania. Stan **przerwanie zabronione** posiada kilka alternatyw, pozostałe stany po jednej.

Wszystkie stany programu są od siebie niezależne, biorąc pod uwagę zarówno ich funkcję jak i sposób, w jaki są wskazywane i przełączane. Zmiana stanu nie wpływa na zawartość rejestrów arytmetycznych i na wykonanie operacji wejścia-wyjścia, natomiast może oddziaływać na pracę zegara.

3.2.2.1. Stan **problem — supervisor**

Wybór pomiędzy stanami **problem** i **supervisor** określa, czy pełny zbiór instrukcji jest dozwolony. W stanie **problem** realizuje się programy użytkowe (problemowe), zaś w stanie **supervisor** programy sterujące.

W stanie **problem** wszystkie instrukcje we-wy, ochrony i instrukcje bezpośredniego sterowania, jak również ŁADUJ SSP, USTAW MASKĘ SYSTEMU i DIAGNOZA są niedozwolone. Są to tzw. instrukcje uprzewilejowane lub nielegalne. Instrukcja uprzewilejowana występująca w stanie **problem** stanowi niedozwoloną operację i powoduje przerwanie programu. W stanie **supervisor** wszystkie rozkazy są dozwolone.

Przełączanie CJP pomiędzy stanami **problem** i **supervisor** odbywa się przez zmianę odpowiedniego bitu w SSP. Bit ten może ulec zmianie jedynie przez wprowadzenie nowego SSP (tj. przez wykonanie instrukcji ŁADUJ SSP lub w wyniku przerwania bieżącego programu). Również początkowe ładowanie programu wprowadza nowe SSP, a tym samym wywołuje nowy stan CJP. Nowe SSP może wprowadzić stan **problem** lub **supervisor** niezależnie od stanu poprzedniego.

3.2.2.2. Stan **czekaj — bieg**

W stanie **czekaj** instrukcje nie są wykonywane i wobec tego nie zachodzi odwołanie się do pamięci z inicjatywy CJP, natomiast w stanie **bieg** odbywa się normalne pobieranie i wykonanie instrukcji programu.

Jeżeli bit 14 w SSP jest jedynką, to CJP czeka; jeżeli jest zerem, to CJP znajduje się w stanie **bieg.** Stan **czekaj** jest sygnalizowany lampką na pulpicie sterowania systemem.

Przełączanie CJP pomiędzy stanami **czekaj** i **bieg** jest wywoływane przez wprowadzenie pełnego nowego SSP, podobnie jak w przypadku bitu stanu **program** (tj. przez wykonanie instrukcji ŁADUJ SSP, przez przerwanie programu lub przez początkowe ładowanie programu). Przełączenie ze stanu **czekaj** można uzyskać przez przerwania typu we-wy, przerwania zewnętrzne lub też przez początkowe ładowanie programu.

3.2.2.3. Stan **przerwanie zabronione — przerwanie dozwolone**

CJP może zabronić lub zezwolić na wszystkie przerwania typu we-wy, zewnętrzne i typu błąd maszyny oraz na pewne przerwania programowe. Gdy CJP zezwala na pewną grupę przerwań, przerwania te są przyjmowane. Gdy natomiast CJP znajduje się w stanie przerwanie zabronione, przerwania systemowe czekają na przejście CJP do stanu przerwanie dozwolone, podczas gdy przerwania programowe oraz przerwania typu błąd maszyny są ignorowane.

Bity maski systemu (bity 0—7 w SSP), bity maski programu (bity 36—39 w SSP) i bit maski błąd maszyny (bit 13 w SSP) stanowią grupę opisującą stan CJP — **przerwanie zabronione.** Gdy bit maski jest jedynką, odpowiadające mu przerwania są przyjmowane przez CJP. Gdy bit maski jest zerem, przerwania te są zabronione. Bity maski systemu opisują stan **przerwanie zabronione — przerwanie dozwolone** dla kanałów multipleksera i selektorów oraz dla sygnałów zewnętrznych. Natomiast bity maski programu opisują stan **przerwanie zabronione — przerwanie dozwolone** dla czterech z 15 typów nieprawidłowości programu. Bit maski błąd maszyny odpowiada wszystkim przerwaniom typu błąd maszyny (powodowane pojawieniem się błędu lub uszkodzenia w maszynie). Niezabronione przerwania programu, jak również przerwania WYWOŁAJ SUPERVISOR są zawsze przyjmowane.

3.2.2.4. Stan **stop — praca**

W stanie **stop** CJP nie wykonuje instrukcji ani też nie przyjmuje przerwań. Natomiast w stanie **praca** CJP wykonuje instrukcje (jeżeli stan **praca** nie jest stanem **czekaj**) i przyjmuje niezabronione przerwania.

Stan **stop** jest sygnalizowany lampką na pulpicie sterowania systemem. Żaden z bitów SSP nie jest przyporządkowany stanowi **stop — praca,** lecz jedynie ręczna interwencja lub błędne zadziałanie maszyny może spowodować zmianę stanów **stop** lub **praca.** Ani instrukcje, ani przerwania nie mogą zatrzymać ani też uruchomić CJP. CJP przechodzi do stanu **stop,** gdy spełniony jest jeden z następujących warunków: wciśnięty jest klucz **stop** na pulpicie operatora, porównywane adresy (bieżący i nastawiony na pulpicie) są jednakowe lub przełącznik rodzaju pracy ustawiony jest w położeniu CYKL INSTRUKCJI (powodujący pracę krokową CJP).

Przejście od stanu **praca** do stanu **stop** zachodzi pod koniec wykonywania danej instrukcji i przed przystąpieniem do wykonywania następnej instrukcji. Wszelkie przerwania oczekujące i niezabronione są przyjmowane w czasie, gdy CJP jest jeszcze w stanie **praca.** Powodują one zapamiętanie starego SSP i pobranie nowego SSP przed wejściem do stanu **stop.** CJP w stanie **stop** nie przyjmuje przerwań. Przerwania te oczekują na przyjęcie.

3.2.3. System przerwań programu

System przerwań programu pozwala CJP reagować na różne wewnętrzne i zewnętrzne zdarzenia, jakie mogą zaistnieć podczas pracy maszyny. Zdarzenia te mogą być związane zarówno z pracą urządzeń wejścia-wyjścia, z pracą samej CJP bądź z pracą innych zewnętrznych obiektów.

W wyniku pojawienia się i przyjęcia zgłoszenia przerwania następuje zmiana stanu CJP, co normalnie oznacza przerwanie wykonywania bieżącego programu i rozpoczęcie wykonywania nowego programu.

Rozróżnia się pięć rodzajów przerwań. Są to przerwania: we-wy, programowe, wywołujące supervisor, zewnętrzne i przerwania błędu (kontrola) maszyny.

Każdej klasie przerwań przyporządkowane jest na stałe dwusłowowe miejsce pamięci operacyjnej, w którym zapisuje się „stare SSP", oraz miejsce, z którego pobiera się „nowe SSP". W czasie przerwania bieżące SSP zapamiętywane jest w miejscu starego SSP. Następnie bieżącym SSP staje się SSP pobrane z miejsca nowego SSP. Po

wykonaniu tych działań stare SSP zawiera wszystkie informacje opisujące stan maszyny aktualny w momencie przerwania. Jeżeli w wyniku działania podprogramu obsługi przyjętego przerwania stare SSP stanie się na powrót bieżącym SSP, to maszyna będzie w takim samym stanie, w jakim była przed przerwaniem, a więc będzie kontynuowany przerwany program.

Tabela 3.2

Miejsca starych i nowych SSP w pamięci

Rodzaj przerwania	Adres początkowy starego SSP	Adres początkowy nowego SSP
Zewnętrzne	24	88
Wywołanie supervisora	32	96
Programowe	40	104
Błąd maszyny	48	112
Wejście-wyjście	56	120

Tabela 3.3

Przyczyny typu przerwanie wejścia-wyjścia

Kod przerwania (bity 16÷23 i 24÷31 starego SSP	Przyczyna przerwania	Bit maski systemu
00000000 uuuuuuuu	kanał 0	0
00000001 uuuuuuuu	kanał 1	1
00000010 uuuuuuuu	kanał 2	2
00000011 uuuuuuuu	kanał 3	3
00000100 uuuuuuuu	kanał 4	4
00000101 uuuuuuuu	kanał 5	5
00000110 uuuuuuuu	kanał 6	6

Litery u oznaczają 8-bitowy adres urządzenia, w którym powstał sygnał przerwania

Tabela 3.4

Przyczyny typu przerwanie programu

Binarny kod przyczyny (bity 24÷31 SSP)	Bit maski program	Przyczyna przerwania
00000001	—	nielegalny kod operacji (niewykorzystany)
00000010	—	nielegalna instrukcja (w stanie „problem")
00000011	—	ciąg instrukcji WYKONAJ INSTRUKCJĘ
00000100	—	próba naruszenia ochrony pamięci
00000101	—	adresacja (błędny adres)
00000110	—	specyfikacja (niespełnione jest określone ograniczenie)
00000111	—	błędne dane
00001000	36	nadmiar stałoprzecinkowy
00001001	—	nieprawidłowe dzielenie stałoprzecinkowe
00001010	37	nadmiar dziesiętny
00001011	—	nieprawidłowe dzielenie dziesiętne
00001100	—	nadmiar wykładnika ($\geqslant 128$)
00001101	38	niedomiar wykładnika (< 0)
00001110	39	zmiennoprzecinkowa różnica lub suma równa się zero
00001111	—	nieprawidłowe dzielenie zmiennoprzecinkowe

Tabela 3.5

Przyczyny typu przerwanie zewnętrzne

Ustawiony bit kodu przerwania starego SSP	Przyczyna przerwania	Bit maski
24	czasomierz interwałowy	7
25	klucz PRZERWANIE	7
26	sygnał zewnętrzny 2	7
27	sygnał zewnętrzny 3	7
28	sygnał zewnętrzny 4	7
29	sygnał zewnętrzny 5	7
30	sygnał zewnętrzny 6	7
31	sygnał zewnętrzny 7	7

Tabela 3.2 pokazuje zarezerwowane dwusłowowe miejsca w pamięci operacyjnej dla starych i nowych SSP wszystkich pięciu rodzajów przerwań.

Kod przerwania, tj. bity 16—31 starego SSP, określa przyczynę, która spowodowała dane przerwanie. Każdy rodzaj przerwań ma własną listę przyczyn przerwania, które wymienione są w tabelach 3.3, 3.4 i 3.5.

Przerwania wejścia-wyjścia. Przerwania te sygnalizują centralnej jednostce, że zadana operacja przesyłania danych do lub z określonego urządzenia we-wy została zakończona lub że w czasie przesyłania powstała sytuacja alarmowa (np. błąd w urządzeniu). Numer kanału (0÷6) i numer urządzenia we-wy, które spowodowało dane przerwanie, pamięta się w bitach 16—31 starego SSP. Natomiast dalsze informacje, bliżej określające przyczynę przerwania, znajdują się w tzw. słowie stanu kanału (SSK), które zostaje zapisane w pamięci operacyjnej w trakcie przerwania.

Zgłoszenie przerwania we-wy może wystąpić w dowolnym momencie wykonywania programu; możliwe jest pojawienie się więcej niż jednego zgłoszenia jednocześnie. Przerwania są przechowywane w systemie we-wy aż do przyjęcia ich przez CJP. Pomiędzy zgłoszeniami ustalany jest priorytet tak, że tylko jedno zgłoszenie przerwania jest przetwarzane w określonym czasie.

Przerwanie we-wy może nastąpić tylko po całkowitym zakończeniu wykonania bieżącej instrukcji i w chwili, gdy CJP jest w stanie Przerwanie Dozwolone, odpowiadającym kanałowi zgłaszającemu przerwanie. Kanały są maskowane bitami 0—6 maski systemu. Zabronione przerwania czekają.

Przerwania we-wy powodują zapamiętanie starego SSP w miejscu 56 oraz zapamiętanie SSK związanego z przerwaniem w miejscu 64. Następnie nowe SSP jest ładowane z miejsca 120.

Przerwania programowe. Przerwania te są powodowane pojawieniem się w wykonywanym programie określonej nieprawidłowości, jak np. nieprawidłowe dane lub nielegalna instrukcja. Pełna lista nieprawidłowości jest podana w tabeli 3.4.

Cztery z 16 nieprawidłowości można zignorować przez wpisanie 0 w odpowiednie bity maski programu (bity 36—39 SSP). Wpisanie natomiast 1 w odpowiedni bit maski zezwala na przyjęcie przerwania spowodowanego daną nieprawidłowością.

Przerwania zewnętrzne. Przerwania zewnętrzne umożliwiają CJP reagowanie na sygnały z czaso-

mierza, z klucza PRZERWANIE znajdującego się na pulpicie operatora, lub na sygnały z obiektów zewnętrznych (np. z drugiej EMC).

Zgłoszenia przerwania zewnętrznego są przechowywane aż do przyjęcia ich przez CJP. Jeśli nastąpi przerwanie zewnętrzne, to następuje jednocześnie przyjęcie wszystkich oczekujących przerwań zewnętrznych (ale każde z nich zgłasza się tylko raz).

Przerwanie zewnętrzne może nastąpić po wykonaniu bieżącej instrukcji i tylko wówczas, gdy 7 bit maski systemu jest jedynką. Tabela 3.5 podaje przyczyny powodujące przerwanie zewnętrzne oraz sposób ich identyfikacji.

Przerwania błąd maszyny. Przerwania te są powodowane pojawieniem się błędów w maszynie. Błędy te mogą być trwałe lub chwilowe. Przerwania błąd maszyny są maskowane przez 13 bit bieżącego SSP. Jeśli bit ten jest jedynką, to sumaryczny sygnał błąd maszyny powoduje wstrzymanie wykonania bieżącej instrukcji i zapoczątkowanie określonej procedury diagnostycznej, po wykonaniu której następuje normalna procedura przerwania. W wyniku przerwania błąd maszyny zapisuje się w pamięci operacyjnej, począwszy od miejsca 128, pełny stan CJP (tzw. erratografię), w którym znajdowała się ona w momencie pojawienia się błędu. Wielkość pola dla erratografii jest różna w różnych modelach. Programowa analiza erratografii pomaga w zlokalizowaniu źródła błędu maszyny.

Przerwanie wywołanie supervisora. Przerwanie to pojawia się w wyniku wykonania instrukcji o nazwie WYWOŁAJ SUPERVISOR. Instrukcja ta powoduje wpisanie jej 8-bitowego parametru I w bitach 24—31 starego SSP. Instrukcji tej używa się przede wszystkim do zmiany stanu CJP ze stanu **problem** na **supervisor**, czyli jest to sposób przejścia z programu użytkowego na program sterujący. Parametr I określa powód zastosowania tej instrukcji w programie.

Zarezerwowane pole pamięci. Z systemem przerwań związane jest początkowe pole pamięci operacyjnej. Komórki tego pola są na stałe zarezerwowane dla przechowywania różnych informacji, które przygotowuje system operacyjny maszyny, a wykorzystuje lub aktualizuje CJP w czasie pracy. Pole początkowe o adresach od 0 do 127 zostało jednakowo wykorzystane we wszystkich maszynach Jednolitego Systemu (tabela 3.6), natomiast dalszy obszar pamięci operacyjnej został różnie wykorzystany przez różne maszyny.

Tabela 3.6

Zarezerwowane pole pamięci pole przerwań

Adres początkowy i końcowy miejsca	Informacja pamiętana
0— 7	SSP procedury początkowego ładowania
8— 15	SKK1 procedury początkowego ładowania
16— 23	SKK2 procedury początkowego ładowania
24— 31	stare SSP przerwań zewnętrznych
32— 39	stare SSP przerwania wywołującego Supervisor
40— 47	stare SSP przerwań programowych
48— 55	stare SSP przerwań błędów maszynowych
56— 63	stare SSP przerwań wejścia-wyjścia
64— 71	słowo stanu kanału (SSK)
72— 75	słowo adresowe kanału (SAK)
80— 83	czasomierz interwałowy
88— 95	nowe SSP przerwań zewnętrznych
96—103	nowe SSP przerwania wywołującego Supervisor
104—111	nowe SSP przerwań programowych
112—119	nowe SSP przerwań błędów maszynowych
120—127	nowe SSP przerwań wejścia-wyjścia
128—	pole dla erratografii i innych informacji

SKK oznacza słowo komendy kanału, w którym przechowuje się różne informacje sterujące przesyłaniem danych (np. adres i ilość przesyłanych bajtów, typ operacji kanałowej itp.)

3.2.4. Priorytet przerwań

W trakcie wykonywania programu może wystąpić jednocześnie kilka przyczyn powodujących przerwania. Jednoczesne żądania przerwań uwzględniane są w następującym porządku:

— przerwanie błędu maszyny,
— przerwanie programowe lub wywołujące supervisor, które wzajemnie się wykluczają,
— przerwanie zewnętrzne,
— przerwanie we-wy.

Gdy zachodzi potrzeba obsłużenia więcej niż jednego przerwania, wtedy przerwanie o najwyższym priorytecie powoduje zapamiętanie starego SSP i pobranie, związanego z tym przerwaniem, nowego SSP. To SSP jest następnie zapamiętywane bez wykonania jakiejkolwiek instrukcji i pobierane jest nowe SSP, wyznaczone przez następne przerwanie.

Proces zapamiętywania i pobierania trwa aż do uwzględnienia ostatniego przerwania, z tym, że przerwania zewnętrzne i we-wy powodują zmianę SSP tylko wtedy, gdy ostatnio pobrane SSP nie maskuje ich. Gdy nie trzeba już pobierać następnego SSP, rozpoczyna się wykonywanie instrukcji. Wynika stąd, że kolejność wykonywania programów obsługi jest odwrotna niż kolejność w jakiej pobierane są słowa stanu programu. Zatem najpilniejsze przerwania (we-wy, zewnętrzne, programowe lub wywołujące supervisor) są obsługiwane jako pierwsze. Jeśli nastąpi błąd maszyny, to przez wpisanie zer w bitach maski zostają udaremnione inne przerwania.

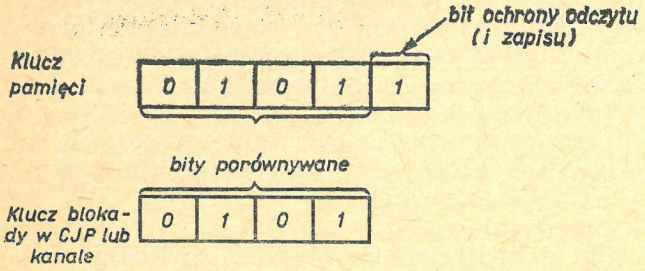
3.2.5. Ochrona pamięci

Ochrona służy do zabezpieczenia zawartości pewnych pól pamięci operacyjnej przed zniszczeniem (lub niewłaściwym wykorzystaniem) spowodowanym błędnym zapisem (lub zapisem i odczytem) informacji podczas wykonywania programu. Ochrona pamięci działa przy zapisie lub zapisie i odczycie. Ochrona ta realizowana jest przez przyporządkowanie danemu blokowi pamięci klucza, a następnie porównanie tego klucza z kluczem ochrony (blokady), dostarczonym razem z danymi (od procesora lub kanału) do zapamiętania. Wykrycie niezgodności powoduje skasowanie dostępu i rozpoznanie błędu ochrony.

Organizacja blokady polega na podzieleniu całej pamięci na bloki liczące po 2048 bajtów i związaniu z każdym blokiem pięciobitowego klucza. Zmianę i badanie tych kluczy umożliwiają instrukcje USTAW KLUCZ PAMIĘCI i CZYTAJ KLUCZ PAMIĘCI. Różne klucze mogą zawierać jednakowe kody.

Dostęp do pamięci określa czterobitowy rejestr, zwany kluczem blokady. Gdy do pamięci odwołuje się jednostka centralna, rolę klucza blokady spełnia klucz bieżącego SSP. Gdy natomiast dostępu do pamięci wymaga kanał, wtedy klucz blokady określony jest przez kanał.

W przypadku wbudowania jedynie blokady zapisu, odbywa się porównywanie klucza w pamięci z kluczem blokady, określonym czy to przez SPP, czy to przez kanał. Dostęp do pamięci możliwy jest tylko wtedy, gdy klucze te są zgodne lub gdy klucz blokady jest równy zeru. Natomiast w przypadku wbudowania blokady zapisu i odczytu dodatkowy bit klucza pamięci określa, czy odczyt jest kontrolowany, czy też nie.



Rys. 3.8. Klucz pamięci i klucz blokady

Próba naruszenia blokady powoduje przerwanie programu. Operacje we-wy naruszające blokadę są wstrzymywane, a fakt naruszenia blokady zaznaczony jest w słowie stanu kanału, zapisywanym w pamięci po każdej operacji we-wy.

3.2.6. Czasomierz interwałowy

Maszyny JS EMC są wyposażone w specjalny rejestr zwany czasomierzem interwałowym, który liczy kolejne jednostki czasu. Zmiana wartości czasomierza z dodatniej na ujemną powoduje przerwanie zewnętrzne, któremu odpowiada jedynka na pozycji 24 SSP.

Czasomierz zajmuje 32-bitowe słowo w miejscu 80—83 pamięci operacyjnej, od której odejmuje się określoną wartość ze stałą częstotliwością, tj. jedynkę na pozycję 23 słowa co 1/300 s. Wyższe dokładności osiąga się przez odjęcie jedynek w pozycjach 24—31, z odpowiednio wyższą częstotliwością, np. co 13 μs na pozycję 31.

Pełny cykl pracy czasomierza wynosi 15,5 godziny. Zmianę stanu czasomierza uzyskuje się przez zmianę zawartości słowa o adresie 80—83. Czasomierz ten stosowany jest przeważnie do mierzenia małych przedziałów czasu. Programowa i ciągła kumulacja małych przedziałów czasu pozwala również na mierzenie czasu realnego.

Dokładność czasomierza interwałowego nie jest wysoka. Dlatego w szybkich maszynach instaluje się dodatkowe dokładniejsze zegary, np. zegar czasu dobowego, czasomierz CJP lub urządzenia dynamicznie sprawdzające, czy zegar dobowy osiągnął określony moment czasu.

3.3. Charakterystyka repertuaru instrukcji maszynowych

Repertuar instrukcji maszynowych Jednolitego Systemu EMC jest bardzo obszerny i ukierunkowany na wykonanie działań na różnych postaciach informacji. Podobnie jak i w innych europejskich rodzinach komputerów, przyjęty w JS EMC repertuar instrukcji oraz związane z nim formaty informacji i ogólne zasady działania, wzorowane są na repertuarze instrukcji maszyn systemów IBM 360 [4]. Dzięki temu, że repertuar instrukcji maszynowych jest obszerny i obejmuje działania na różnych formatach informacji, można tworzyć zwarte i efektywne programy.

W repertuarze instrukcji przewidziano działania oparte o różne systemy liczenia i sposoby przedstawienia danych oraz przyjęto różne sposoby adresacji.

3.3.1. Sposoby adresowania

W maszynach Jednolitego Systemu przyjęto binarny system adresacji zarówno dla danych dwójkowych, jak i dziesiętnych. System adresacji jest jednolity dla danych i różnych formatów instrukcji. Najmniejszą adresowalną jednostką informacji w pamięci jest tzw. 8-bitowy bajt. Można zatem bezpośrednio zaadresować pojedynczy 8-bitowy znak alfanumeryczny, tj. jedną nieupakowaną cyfrę dziesiętną, dwie upakowane (4-bitowe) cyfry dziesiętne lub pojedynczy nienumeryczny znak pisarski.

Możliwość adresowania z dokładnością do bajtu jest bardzo istotna w przetwarzaniu danych, gdzie często spotyka się zbiory danych (zapisy) będące ciągami znaków alfanumerycznych o dowolnej długości. Długość tych ciągów określana jest również z dokładnością do bajtu. Zatem przy organizowaniu struktury zapisów nie zachodzi konieczność zarezerwowania dodatkowych pól związanych z ograniczeniami systemu adresacji, co ma miejsce w systemach o adresacji słowowej.

Przyjęcie 8-bitowego bajtu, który reprezentuje pojedynczy nienumeryczny znak lub parę cyfr dziesiętnych, jest kompromisem pomiędzy żądaniem wprowadzenia bogatego repertuaru znaków alfanumerycznych a zwartością zakodowanych informacji. Zwartość informacji zbiorów danych została osiągnięta dzięki przechowywaniu dwóch cyfr dziesiętnych w jednym bajcie. Taki sposób przechowywania danych dziesiętnych jest bardzo istotny, gdyż w wielu zastosowaniach dziesiętne dane stanowią powyżej 60% znaków zbiorów danych. Pakowanie dwóch cyfr dziesiętnych w jednym bajcie przyczynia się również do zwiększenia efektywności przesyłania danych pomiędy jednostką centralną i pamięciami zewnętrznymi.

Możliwość adresowania informacji z dokładnością do bajtu jest także bardzo przydatna przy teleprzetwarzaniu, gdzie przekazuje się i odbiera dowolne ciągi symboli.

Adres efektywny instrukcji jest przeważnie określany przez trzy pola instrukcji: pole rejestru bazy (B), pole rejestru indeksowego (X) i parametr A. Przy czym rejestrem bazy lub rejestrem indeksowym może być dowolny rejestr ogólny o adresie od 1 do 15. Parametr A zajmuje zawsze

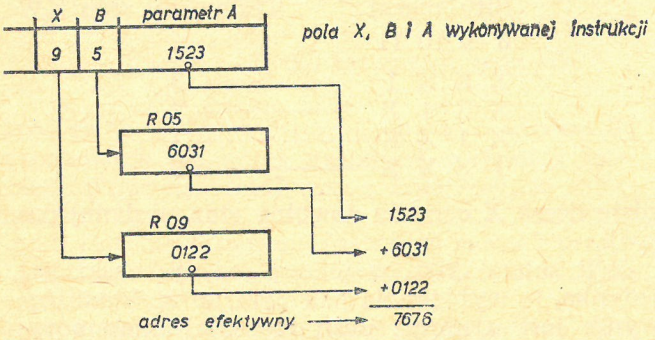
12 bitów instrukcji i przyjmuje wartości od 0 do 4095.

Adres efektywny E ustala się następująco (rys. 3.9.):

E:=A, dla B=X=0;
E:=(B)+A, dla B≠0 i X=0;
E:=(X)+A, dla B=0 i X≠0;
E:=(B)+(X)+A, dla B≠0 i X≠0;

gdzie symbol (B) oznacza 24-bitową zawartość rejestru ogólnego o adresie B (odpowiednio X).

W powyższej modyfikacji biorą udział tylko bity 8—31 rejestru ogólnego, zaś bity 0—7 są ignorowane. Zatem adres efektywny E pozwala na bezpośrednie adresowanie pamięci o pojemności do 16 Mbajtów.



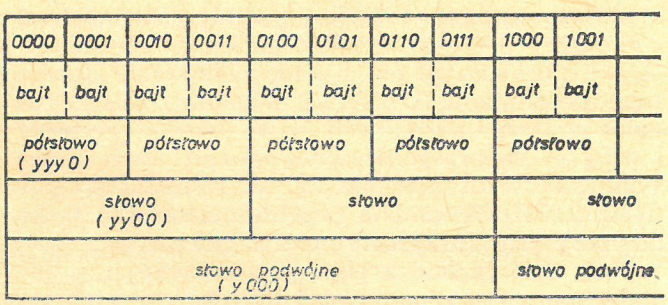
Rys. 3.9. Schemat modyfikacji adresu instrukcji

Wszystkie instrukcje komunikujące się z pamięcią zawierają modyfikację bazową, zaś instrukcje typu RX posiadają dodatkowo modyfikację X.

W rejestrze bazowym przechowuje się na ogół początkowy adres programu, czyli tzw. bazę programu. Natomiast rejestry indeksowe zawierają różne bieżące wskaźniki występujące w programach i zmieniające się dynamicznie w czasie wykonywania danego programu.

Modyfikacja indeksowa stosowana jest w instrukcjach arytmetycznych, logicznych i skokowych (typu RX).

Adresy początkowe pól o stałej długości. W pamięci operacyjnej wyróżnia się nie tylko pola o zmiennej długości (liczonej w bajtach), lecz także pola o stałej długości (półsłowa, słowa, słowa podwójne i słowa poczwórne). Zastosowanie wielkości liczbowych o stałej długości przyśpiesza wykonywanie zadań, jeśli wielkości te ulokowane są w całkowitych granicach. Oznacza to, że adresy początkowe tych wielkości powinny być wielokrotnością ich długości (rys. 3.10). Jeśli np. 4-bajtowa informacja (słowo) rozmieszczona jest w jednym słowie pamięciowym, to można ją przeczytać w jednym cyklu pamięci, gdzie również szyna informacyjna ma szerokość jednego słowa. Jeśli natomiast rozmieszczona jest w dwóch kolejnych słowach pamięci, to można ją przeczytać w całości organizując dwa oddzielne odczyty.



Rys. 3.10. Granice całkowite półsłów, słów i słów podwójnych (górny wiersz podaje cztery mniej znaczące bity adresu pamięciowego); litera *y* oznacza, że wartość bitu w tej pozycji adresu może przyjąć 0 lub 1

W celu przyspieszenia wykonania instrukcji, w których występują argumenty o stałej długości, nakłada się ograniczenia na rozmieszczenie tych argumentów w pamięci operacyjnej. Żąda się mianowicie, aby argumenty te rozmieszczone były w całkowitych granicach. Spełnienie tych ograniczeń w maszynach sprawdzają specjalne układy, które powodują przerwanie bieżącego programu, jeśli to ograniczenie nie jest spełnione.

Żądanie rozmieszczenia informacji liczbowych w całkowitych granicach adresowych jest niewygodne dla programisty i dlatego w nowszych maszynach rezygnuje się z tego żądania; chociaż może to powodować pewne zmniejszenie szybkości wykonania programu.

3.3.2. Zestawy instrukcji maszynowych

Zbiór instrukcji maszynowych Jednolitego Systemu EMC jest obszerny i składa się z kilku funkcjonalnych grup instrukcji. Głównymi grupami instrukcji są:

— instrukcje arytmetyki stałoprzecinkowej i instrukcje logiczne,
— instrukcje zmiennoprzecinkowe,
— instrukcje dziesiętne.

Tabela 3.7

Podział instrukcji EMC JS na zestawy

Grupa instrukcji	Podgrupy instrukcji	Zestaw instrukcji: standardowy	do obliczeń naukowo-technicznych	do zastosowań EPD	do zastosowań uniwersalnych
Instrukcje zmienno-przecinkowe	instrukcje działań na wielkościach zmienno-przecinkowych (krótkich i długich)				
Instrukcje standardowe	instrukcje arytmetyki stałoprzecinkowej (w tym operacje przesunięć i konwersji) instrukcje logiczne (w tym operacje przenoszenia, porównania, tłumaczenia i przesunięć logicznych) instrukcje pakowania i rozpakowywania instrukcje skoku instrukcje zmiany stanu instrukcje we-wy instrukcje ochrony pamięci				
Instrukcje dziesiętne	instrukcje działań dziesiętnych instrukcje redagowania				
Instrukcje bezpośredniego działania	instrukcja czytania bezpośredniego instrukcja pisania bezpośredniego				

Tabela 3.7. przedstawia grupy i podgrupy instrukcji oraz podział instrukcji maszynowych na zestawy. Przyjęty i wbudowany w danej maszynie zestaw instrukcji silnie charakteryzuje moc obliczeniową maszyny w różnych zastosowaniach.

Instrukcje maszynowe zostały podzielone na następujące zestawy:

a) zestaw instrukcji standardowych: jest to minimalny zestaw, bez którego maszyna (łącznie z przyjętymi systemami programowania) nie może funkcjonować; zestaw ten zawiera wszystkie instrukcje stałoprzecinkowe, instrukcje logiczne, skoku i zmiany stanu wewnętrznego jednostki centralnej oraz instrukcje we-wy i inne;
b) zestaw instrukcji do obliczeń naukowo-technicznych, który powstaje przez dodanie do zestawu standardowego licznej grupy instrukcji działań na wielkościach zmiennoprzecinkowych;
c) zestaw instrukcji do zastosowań EPD, gdzie do zestawu standardowego dodaje się dużą grupę instrukcji działań na liczbach dziesiętnych i działań na polach o zmiennej długości;
d) zestaw uniwersalny, który zawiera wszystkie wyżej opisane zestawy oraz instrukcje różnych opcji (np. instrukcje bezpośredniego czytania i pisania).

3.3.3. Sterowanie kolejnością wykonywania instrukcji

Przyjęty sposób sterowania kolejnością wykonywania instrukcji programu zapewnia:

— zmianę kolejności wykonania instrukcji w zależności od relacji wyniku operacji arytmetycznych lub logicznych (skoki warunkowe),
— bezwarunkową zmianę kolejności wykonywania instrukcji programu (skoki bezwarunkowe),
— bezwarunkowe przejście do wykonania nowej sekwencji instrukcji (np. podprogramu) z zapamiętaniem miejsca wyjścia (śladu) ze starej sekwencji instrukcji,
— zapamiętanie wybranego miejsca (śladu) sekwencji instrukcji programu, bez przejścia do nowej sekwencji,
— wymuszoną zmianę kolejności wykonywania instrukcji w wyniku zajścia dowolnego zdarzenia wewnątrz lub na zewnątrz jednostki centralnej (czyli w wyniku przerwania),
— wykonanie dowolnie wybranej instrukcji programu, nie będącej kolejną w wykonywanej sekwencji i powrót do poprzedniej sekwencji instrukcji.

Instrukcje skoku warunkowego są wykonywane w zależności od stanu specjalnego dwubitowego rejestru, zwanego rejestrem kodu warunków (KW). W czasie wykonywania operacji arytmetycznej lub logicznej rejestr KW jest nastawiony w jednym z czterech stanów: 0, 1, 2 lub 3. Stany te mogą po operacji logicznej oznaczać np., że wynik jest równy zeru (KW=0), bądź różny od zera (KW=1). Po operacji arytmetycznej mogą natomiast oznaczać, że wynik operacji jest równy zeru (KW=0), jest mniejszy od zera (KW=1), większy od zera (KW=2), bądź jest z nadmiarem, tzn. nie mieści się w przewidzianym dla wyniku polu (KW=3).

Po operacjach porównania dwóch wielkości stan rejestru może oznaczać, że wielkość pierwsza jest równa wielkości drugiej (KW=0), wielkość pierwsza jest mniejsza od wielkości drugiej (KW=1), bądź wielkość pierwsza jest większa od wielkości drugiej (KW=2).

Typową instrukcją skoku warunkowego jest instrukcja SKOCZ WEDŁUG WARUNKU (patrz również Dodatek A), która określa warunek, tj. stan rejestru KW, przy którym ma zajść skok.

Można wybrać jeden lub kilka warunków skoku, stosownie do wartości tzw. maski, czyli wartości pola R1 tej instrukcji.

Maska, tj. bity 8, 9, 10 i 11 instrukcji, została wykorzystana w sposób następujący:

Bity maski 8 9 10 11		Testowany stan KW
X X X 1	⟶	3
X X 1 X	⟶	2
X 1 X X	⟶	1
1 X X X	⟶	0

gdzie litera X oznacza dowolny stan w danej pozycji.

Jeśli zatem tylko w jednej pozycji maski jest jedynka, to wybiera się jeden warunek skoku (np. wynik jest ujemny). Jeśli we wszystkich pozycjach maski są zera, to skok nie zachodzi. Jeśli natomiast we wszystkich pozycjach są jedynki (czyli pole R1=15), to wówczas skok wykonuje się bezwarunkowo.

Instrukcja SKOCZ WEDŁUG WARUNKU może być typu RR lub RX. Adresem skoku (nowy stan licznika instrukcji w słowie SSP) jest zawartość rejestru ogólnego, którego adres określa pole R2 instrukcji typu RR, lub adres efektywny E2 instrukcji typu RX, gdzie E2=(X2)+(B2)+A2.

Dla organizowania optymalnej struktury procedur programowych z pętlami powtórzeń przyjęto instrukcje o nazwach: SKOCZ WEDŁUG LICZNIKA, SKOCZ PRZY INDEKSIE WIĘKSZYM, SKOCZ PRZY INDEKSIE MNIEJSZYM LUB RÓWNYM.

Pierwsza z tych instrukcji pomniejsza stan licznika powtórzeń o jeden (technicznie jest to stan rejestru ogólnego o adresie R1) oraz powoduje skok do określonej przez R2 lub E2 instrukcji programu, gdy licznik jest jeszcze różny od zera. W ten sposób jedna instrukcja zastępuje dwa „konwencjonalne" rozkazy spotykane w innych maszynach.

Dwie dalsze instrukcje, tj. SKOCZ PRZY INDEKSIE (RS), są bardziej złożone i bardziej efektywne w zastosowaniach. Instrukcje te mogą powodować zmianę licznika powtórzeń (indeksu) o dowolny przyrost, a także podejmować decyzje skoku, porównując ten licznik z dowolnie dobranym programem. Wszystkie wielkości biorące udział w tych instrukcjach — tj. licznik, przyrost i próg — są przechowywane w rejestrach o natychmiastowym dostępie, tj. w rejestrach ogólnych. Jeśli programista pragnie zastosować tylko dowolny krok (przyrost) zmiany indeksu (np. 2, 4, lub 8), zaś wyjście z pętli programowej ma nastąpić po „wyczerpaniu się" licznika, to wielkość progu może być utożsamiona (przez wskazanie nieparzystego rejestru) z wielkością kroku zmiany indeksu i tym samym zmniejszyć ilość różnych parametrów początkowych.

Instrukcja SKOCZ PRZY INDEKSIE MNIEJSZYM LUB RÓWNYM jest przeznaczona do zastosowania jej na końcu pętli programowej, zaś instrukcja SKOCZ PRZY INDEKSIE WIĘKSZYM występuje na ogół na początku pętli.

Sposób wywoływania podprogramów. Wywoływanie podprogramu, tj. wyjście z sekwencji instrukcji programu głównego i wejście do sekwencji instrukcji podprogramu, jest dokonywane w sposób znormalizowany poprzez zastosowanie instrukcji o nazwie SKOCZ ZE ŚLADEM. Funkcję tej instrukcji można w skrócony sposób przedstawić następująco:

1. $SSP \xrightarrow{32-63} R1$,
2a. $(R2) \longrightarrow LI$, dla instrukcji typu RR, lub
2b. $E2 \longrightarrow LI$, dla instrukcji typu RX.

Oznacza to, że w rejestrze ogólnym o adresie R1 przechowuje się adres następnej instrukcji programu głównego (ślad) wraz z dodatkowymi informacjami, jak kod długości instrukcji, kod warunku i maska programu (patrz też rys. 3.7 przedstawiający strukturę podstawowego SSP), po czym wpisuje się do licznika instrukcji adres pierwszej do wykonania instrukcji podprogramu.

Przechowywanie w śladzie również kodu warunku KW i maski programu głównego jest sprawą bardzo istotną, gdyż podprogram stosuje na ogół inną maskę programu oraz wytwarza własne lokalne kody warunków. Natomiast po powrocie z podprogramu do programu głównego stosuje się początkowy kod warunku i początkową maskę programu.

Odnowienie kodu warunku i maski programu dokonuje się jednocześnie za pomocą jednej instrukcji o nazwie USTAW MASKĘ PROGRAMU, która ładuje rejestr kodu warunków i rejestr maski programu zawartością odpowiednich bitów pierwszego bajtu wskazanego rejestru ogólnego.

Pośrednie wykonywanie instrukcji. Często można uprościć strukturę programu — lub uczynić ją bardziej elastyczną — jeśli jego „podprogramy" składają się z jednej tylko instrukcji. W celu wywołania i wykonania takiego „podprogramu" oraz zorganizowania powrotu do programu głównego, wprowadzono do repertuaru instrukcji maszynowych instrukcję WYKONAJ INSTRUKCJĘ (EX). Instrukcja ta pozwala wykonać dowolną instrukcję wyznaczoną przez E2 — nie umieszczoną w wykonywanej sekwencji — bez wyjścia z bieżącej sekwencji instrukcji. Wykonywana instrukcja (tj. część jej bitów 8—15) zostaje zmodyfikowana przez zawartość (bitów 24—31) rejestru ogólnego, określonego przez pole R1 instrukcji WYKONAJ INSTRUKCJĘ. W ten sposób można pośrednio określić bity 8—15 wykonywanej instrukcji, czyli określić: długość argumentów operacji (w instrukcjach typu SS), indeks, maskę, adresy rejestrów ogólnych i bezpośrednie parametry biorące udział w wykonywanej instrukcji. Można na przykład wykonywać instrukcje dziesiętne, gdzie długość ich argumentów operacji nie jest sztywno zadana przez L1 i L2 wykonywanej instrukcji, lecz określana jest pośrednio przez rejestr ogólny i ustalana zostaje dopiero przed samym wykonaniem danej instrukcji dziesiętnej.

3.3.4. Instrukcje przełączania stanu

Instrukcje przełączania stanu (zwane też sterującymi) sterują wykonaniem programu poprzez zmianę części lub całego słowa SSP, zmianę stanu kluczy ochrony pamięci lub też poprzez „dobrowolne" przerwanie programu użytkowego i wejście do programu systemu operacyjnego (do supervisora). Większość instrukcji przełączania stanu to instrukcje uprzywilejowane (nielegalne). Oznacza to, że są one wykonywane przez sprzęt tylko wówczas, gdy pojawią się w stanie supervisor. Pojawienie się natomiast w instrukcji uprzywilejowanej w stanie problem, tj. w stanie programu użytkowego, powoduje przerwanie programowe.

Wszystkie instrukcje przełączania stanu, za wyjątkiem dwóch instrukcji (RDD i WRD), należą do standardowego zestawu instrukcji Jednolitego Systemu EMC. Instrukcje przełączania stanu dokładnie charakteryzują działanie i możliwości maszyny i dlatego poniżej zostanie podany ich skrócony opis. Poniższa tabela podaje nazwy i kody literowe instrukcji przełączania stanu, wchodzących do podstawowego repertuaru instrukcji maszynowych:

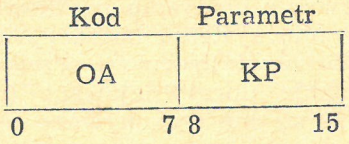
Nazwa instrukcji	Kod	Typ
ŁADUJ SSP	LPSW	SI
USTAW MASKĘ PROGRAMU	SPM	RR
USTAW MASKĘ SYSTEMU	SSM	SI
WYWOŁAJ SUPERVISOR	SVC	RR
USTAW KLUCZ PAMIĘCI	SSK	RR
CZYTAJ KLUCZ PAMIĘCI	ISK	RR
CZYTAJ BEZPOŚREDNIO	RDD	SI
PISZ BEZPOŚREDNIO	WRD	SI
DIAGNOZA	—	SI
TESTUJ I USTAW	TS	SI

Instrukcja ŁADUJ SSP powoduje pobranie z miejsca pamięci, określonego przez adres efektywny E, słowa podwójnego i załadowanie go do rejestru procesora SSP. W ten sposób pobrane tą instrukcją podwójne słowo staje się nowym SSP. Zatem wprowadza się nowy stan programu **(problem — supervisor, czekaj — bieg, stop — praca, przerwania zabronione — przerwania dozwolone)**, nową maskę systemu i program oraz nowy licznik instrukcji. Instrukcja ŁADUJ SSP jest przeważnie stosowana przez system operacyjny w celu przejścia do stanu **problem** lub do stanu **czekaj.**

W celu dokonania zmiany tylko w masce systemu, tj. w bitach 0—7 słowa SSP, stosuje się instrukcję USTAW MASKĘ SYSTEMU, która ładuje bajt pamięci — określony przez adres efektywny E — do rejestru SSP. Wprowadzona nowa maska systemu ustala nowe warunki dla przyjęcia przerwań z urządzeń wejścia-wyjścia, z czasomierza, z klucza operatora lub z sygnałów zewnętrznych. Można na przykład zablokować wszelkie przerwania tego typu na krótki czas

w celu aktualizacji określonych tablic systemu operacyjnego.

Instrukcja WYWOŁAJ SUPERVISOR normalizuje sposób wejścia do systemu operacyjnego w celu przekazania określonego zlecenia. Instrukcja ta ma specjalny format:



Powyższa instrukcja (nieuprzywilejowana) powoduje wywołanie systemu operacyjnego z pozostawieniem parametru KP w części kod przerwania starego SSP (tj. w bitach 24—31). Czyli pamięta się stare SSP w polu 32 i pobiera się nowe SSP z pola 96. Parameter *KP* określa 256 różnych powodów (zleceń) dobrowolnego przerwania programu użytkowego.

Podprogram supervisora, który obsługuje to programowe przerwanie, identyfikuje przyczynę przerwania dobrowolnego analizując parametr KP. Przy pomocy tej instrukcji można sygnalizować: koniec obliczeń programu, potrzebę wprowadzenia nowych danych z określonego urządzenia, wydrukowanie gotowych wyników, itp.

Instrukcje USTAW KLUCZ PAMIĘCI i CZYTAJ KLUCZ PAMIĘCI są związane z systemem protekcji pamięci operacyjnej, który to system chroni informacje programów przed mylnymi lub nieupoważnionymi odwołaniami do tych informacji ze strony bieżącego programu. Jak wiadomo (patrz punkt 3.2.5), każdy blok pamięci operacyjnej o pojemności 2048 bajtów ma własny 5-bitowy klucz pamięci. Powyższe instrukcje są formatu RR, czyli występuje w nich pole R1 i pole R2. Rejestr ogólny, w którym znajduje się adres bloku pamięci (czyli adres klucza pamięci), określany jest przez pole R2. Natomiast rejestr ogólny o adresie R1 zawiera 5-bitowy klucz, (tj. liczbę od 0 do 15 i bit ochrony odczytu), który to klucz trzeba wpisać do rejestru klucza danego bloku pamięci (w przypadku instrukcji PISZ). W przypadku instrukcji CZYTAJ przepisuje się zawartość określonego rejestru klucza ochrony do rejestru ogólnego o adresie R1. Instrukcje pisania i czytania klucza pamięci są stosowane przez system operacyjny przy przydzielaniu obszarów PA0 poszczególnym programom użytkowym.

Instrukcje PISZ BEZPOŚREDNIO i CZYTAJ BEZPOŚREDNIO, wbudowywane są jako opcja — wraz z blokiem bezpośredniego sterowania — i umożliwiają współpracę dwóch różnych jednostek centralnych. Przy pomocy tych instrukcji przekazuje się (lub odbiera się) z jednej jednostki centralnej do drugiej 8-bitową informację. Informacja ta jest traktowana przez odbierającą jednostkę centralną jako tzw. sygnały zewnętrzne, powodujące w niej przerwania. W ten sposób jedna maszyna może spowodować przerwanie w innej maszynie. Sygnały zewnętrzne mogą na przykład świadczyć o uszkodzeniu maszyny wysyłającej te sygnały.

Instrukcja TESTUJ I USTAW powoduje przeczytanie stanu wskazanego (przez adres E1) bajtu pamięci i ustawienie wartości kodu warunku zgodnie z zerowym (lewym) bitem tego bajtu. Następnie do wszystkich bitów bajtu w pamięci wpisuje się jedynki. Instrukcja ta normalizuje sposób korzystania ze wspólnego obszaru przez kilka programów. W tych przypadkach lewy bit określonego bajtu traktuje się jako bit sterujący.

We wszystkich programach mających korzystać ze wspólnego obszaru pamięci ustala się, że dostęp do tego obszaru jest dopuszczalny, o ile przykładowo w bicie sterującym jest zero. Gdy natomiast w bicie tym jest jedynka, obszar używany jest aktualnie przez inny program. Wartość bitu sterującego powinna być sprawdzana przy pomocy instrukcji TESTUJ I USTAW. Gdy w wyniku wykonania tej instrukcji kod warunku ma wartość zero, to obszar jest dla danego programu dostępny. Ponieważ ta instrukcja uniemożliwia dostęp do testowanego bajtu przed wpisaniem do niego jedynek, więc niemożliwe jest, aby inny program uzyskał dostęp do wspólnego obszaru, zanim program aktualnie z niego korzystający nie wpisze zera do bitu sterującego.

Instrukcja DIAGNOZA powoduje na ogół wykonanie określonych testów diagnostycznych przez centralną jednostkę przetwarzania i zapisanie wyników tych testów w zarezerwowanym początkowym (począwszy od miejsca 128) polu pamięci operacyjnej. Zapisany stan CJP może składać się z 256 bajtów informacji. Analiza stanu CJP, jaki osiągnęła ona po wykonaniu określonego testu, służy do zlokalizowania uszkodzonych zespołów maszyny.

3.3.5. Operacje arytmetyczne

W maszynach Jednolitego Systemu przyjęto obszerną grupę działań stałoprzecinkowych (patrz Dodatek A). Działania te wykonywane są na argumentach krótkich (16-bitowych) lub długich (32-bitowych). Dzięki przyjęciu binarnego systemu adresacji, można przy pomocy działań stałoprzecinkowych wykonywać także różne manipulacje na wielkościach adresowych. Zarówno dane stałoprzecinkowe, jak i adresy, są przedstawiane w dwójkowej arytmetyce uzupełnieniowej, gdzie pozycją znakową jest tylko lewy skrajny bit danej wielkości. Przyjęcie takiej arytmetyki uzupełnieniowej ma dwie ważne zalety:

— upraszcza programowanie działań stałoprzecinkowych o wielokrotnej precyzji obliczeń,
— zapewnia jednolitość działań na danych i adresach.

Większość wbudowanych instrukcji stałoprzecinkowych dotyczy argumentów długości jednego słowa. Może to być np. zawartość rejestru ogólnego lub zawartość 4 kolejnych bajtów pamięci operacyjnej (rozmieszczonych w całkowitych granicach adresowych). Jednosłowowa wielkość argumentów jest na ogół wystarczająca dla większości obliczeń na liczbach stałoprzecinkowych. Często również wystarcza mniejsza dokładność i zakres obliczeń. W związku z tym wbudowano również instrukcje działające na 16-bitowych argumentach. Argumenty te pozwalają osiągnąć większą szybkość obliczeń, w porównaniu z argumentami 32-bitowymi oraz lepszą pakowność danych w pamięci. Argumenty krótkie, czyli 16-bitowe, zajmują w czasie działania zawsze lewą połowę słowa (tj. bity 0—15). Oznacza to, że 16-bitowy wynik operacji wprowadzony zostaje w bitach 0—15

określonego rejestru ogólnego, z jednoczesnym wyzerowaniem prawej połowy (bity 16—31) tego rejestru. Dzięki temu można również wykonywać działania na mieszanych wielkościach, tzn. na 16- i 32-bitowych argumentach, traktowanych oczywiście w tej samej skali binarnej.

Programowane działania stałoprzecinkowe o wielokrotnej precyzji są wspomagane przez instrukcje dodające lub odejmujące (arytmetycznie!) 32-bitowe wielkości logiczne. Nazwa „wielkości logiczne" uzasadniona jest tym, że w tych argumentach zerowa pozycja słowa nie jest traktowana jako znak arytmetyczny, lecz jak najstarszy znaczący bit argumentu (lub wyniku operacji).

Instrukcje dodające lub odejmujące 32-bitowe wielkości logiczne sygnalizują ewentualne przeniesienia (z pozycji 0 na −1) poprzez wpisanie odpowiedniego kodu warunku w rejestrze KW. Rolę wspomagającą w podwójnej precyzji obliczeń odgrywają również takie operacje, jak dzielenie dwusłowowej liczby stałoprzecinkowej przez liczbę jednosłowową, tworzenie iloczynu dwusłowowego oraz operacje arytmetycznego i logicznego przesuwania dwóch kolejnych rejestrów ogólnych.

W czasie wykonywania działań stałoprzecinkowych mogą wystąpić dwie nieprawidłowości: nadmiar stałoprzecinkowy i dzielenie nieprawidłowe (tzn. iloraz dzielenia nie mieści się w rejestrze, co ma miejsce w przypadku dzielenia przez zero, lub gdy dzielnik jest mniejszy od dzielnej). Programista ma możliwość zignorowania nadmiaru stałoprzecinkowego przez wpisanie zera do odpowiedniego bitu maski programu (patrz tab. 3.4) lub zezwolenia na przerwanie w przypadku pojawienia się tego nadmiaru. Pojawienie się natomiast sygnału „nieprawidłowe dzielenie stałoprzecinkowe" zawsze powoduje przerwanie programowe, niezależnie od stanu maski programu.

Działania zmiennoprzecinkowe. Wybór długości 32 i 64 bitów dla liczb zmiennoprzecinkowych jest oczywiście podyktowany długością słowa maszynowego, przy czym pierwszy bajt obu liczb zmiennoprzecinkowych został wykorzystany dla przechowywania wykładnika i znaku części ułamkowej (mantysy) liczby zmiennoprzecinkowej (patrz rys. 3.2).

Liczby zmiennoprzecinkowe z mantysą o długości 56 bitów stosowane są w obliczeniach, gdzie wymagana jest duża dokładność obliczeń; natomiast liczby z mantysą 24 bitów stosowane są w obliczeniach o mniejszych wymaganiach co do dokładności obliczeń, przy czym, jak wykazuje praktyka, w ponad 90% zastosowań wystarcza dokładność 6 cyfr szesnastkowych.

W praktyce spotyka się również zastosowania obliczeniowe, gdzie nawet podwójna precyzja, w której mantysa ma 14 cyfr szesnastkowych, nie wystarcza i dlatego w dużych maszynach Jednolitego Systemu przyjęto i wbudowano instrukcje poczwórnej precyzji, gdzie mantysa ma 28 cyfr szesnastkowych.

Wyrażenie wykładnika liczby zmiennoprzecinkowej w formie potęgi o podstawie 16 (patrz punkt 3.1.5) pozwala na czterokrotne zwiększenie zakresu obliczeń w stosunku do podstawy 2.

W operacjach zmiennoprzecinkowych jeden z argumentów przechowywany jest w jednym z czterech rejestrów zmiennoprzecinkowych, zaś drugi w 8-bajtowym polu pamięci operacyjnej.

W skład instrukcji zmiennoprzecinkowych wchodzą instrukcje ładowania rejestrów, dodawania, odejmowania, porównania, mnożenia, dzielenia i pamiętania stanu rejestrów zmiennoprzecinkowych, przy czym istnieją osobne instrukcje działające na liczbach krótkich i osobne na liczbach długich. W celu uzyskania maksymalnej dokładności obliczeń w instrukcjach dodawania, odejmowania, mnożenia i dzielenia, można spowodować wykonanie normalizacji wyników. Dwie operacje, tj. dodawanie i odejmowanie, mogą być również wykonywane bez normalizacji wyników. Zwykle normalizacja ma miejsce w odniesieniu do końcowego wyniku obliczeń. Natomiast w operacjach mnożenia i dzielenia zachodzi normalizacja argumentów bezpośrednio przed wykonaniem operacji. W związku z tym, że normalizację wykonuje się dla cyfr szesnastkowych (z dokładnością do 4 bitów), trzy najbardziej znaczące bity normalizowanej liczby mogą być zerami.

Poczwórna dokładność zmiennoprzecinkowa, zwana też rozszerzoną precyzją, stosuje 4-słowowe argumenty operacji. Argumenty te zajmują dwa rejestry zmiennoprzecinkowe (o adresach 0,2 lub 4,6) lub dwa kolejne 8-bajtowe pola pamięci operacyjnej, zatem składają się z dwóch części. Wykładnik i znak bardziej znaczącej części są traktowane jako wykładnik i znak liczby o rozszerzonej precyzji. Pole ułamka mniej znaczącej części zawiera 14 mniej znaczących cyfr szesnastkowych z 28-cyfrowego ułamka liczby o rozszerzonej precyzji. Znak i wykładnik mniej znaczącej części są ignorowane. Jeżeli jednak wynik operacji jest liczbą o rozszerzonej precyzji, to wówczas znak mniej znaczącej części jest równy znakowi bardziej znaczącej części, a wykładnik mniej znaczącej części jest mniejszy o 14 od wykładnika bardziej znaczącej części.

W czasie wykonywania działań zmiennoprzecinkowych może powstać nadmiar lub niedomiar zmiennoprzecinkowy. Nadmiar zmiennoprzecinkowy oznacza, że wykładnik wyniku jest zbyt duży i nie mieści się w 7-bitach, czyli jest równy lub większy od 128. Natomiast niedomiar zmiennoprzecinkowy oznacza, że wykładnik wyniku jest mniejszy od zera. Pojawienie się nadmiaru zmiennoprzecinkowego świadczy na ogół o błędzie w obliczeniach lub o przyjęciu niewłaściwej procedury obliczeń; dlatego nadmiar ten jest zawsze wykrywany w maszynie i powoduje przerwanie programowe. Natomiast niedomiar zmiennoprzecinkowy świadczy tylko o tym, że wynik jest zbyt mały i tym samym może być przyrównany do zera. Programista może zignorować fakty powstania niedomiaru — przez wpisanie zera w bicie 38 słowa SSP — lub zezwolić na zasygnalizowanie tych faktów przez przerwanie programu.

W niektórych obliczeniach może być również przydatna informacja o tym, że różnica lub suma mantys liczb zmiennoprzecinkowych jest równa zeru. Dlatego wprowadzono również sprzęt do wykrywania i sygnalizowania takiej zerowej różnicy lub sumy. O tym, czy fakt ten ma spowodować przerwanie programowe, czy też nie, decyduje programista przez wpisanie 1 lub 0 w bicie 39 słowa SSP.

Działania dziesiętne. Głównym celem wbudowywania arytmetycznych działań dziesiętnych jest

usunięcie konieczności konwersji wprowadzanych danych z układu dziesiętnego na układ binarny oraz konwersji wyników obliczeń z układu binarnego na układ dziesiętny. Jest to szczególnie ważne, gdy na wprowadzonych danych wykonuje się zaledwie kilka prostych działań arytmetycznych. W takich przypadkach nie warto wykonywać czasochłonnych operacji konwersji z jednego układu na drugi, lecz efektywniej jest wykonać te działania arytmetyczne od razu na danych w postaci dziesiętnej. Dzięki temu tradycyjny schemat przygotowania danych do obliczeń i wykonanie obliczeń w układzie dwójkowym (pakowanie — konwersja dziesiętne na binarne — binarne działania arytmetyczne — konwersja binarne na dziesiętne — rozpakowanie do wydruku), zastępuje się prostszym schematem (pakowanie — dziesiętne działania arytmetyczne — rozpakowanie do wydruku).

Jak już wiemy liczby dziesiętne występują w postaci rozpakowanej lub w postaci upakowanej, gdzie dwie cyfry dziesiętne przechowywane są w jednym bajcie. Instrukcje arytmetyki dziesiętnej umożliwiają dodawanie, odejmowanie, porównywanie, mnożenie i dzielenie liczb dziesiętnych oraz przekształcanie formatów. We wszystkich działaniach dziesiętnych, za wyjątkiem instrukcji PAKUJ I ROZPAKUJ, argumenty i wynik operacji występują w postaci upakowanej. Natomiast początkowy argument instrukcji PAKUJ oraz wynik instrukcji ROZPAKUJ występują oczywiście w postaci rozpakowanej (gdzie jedna cyfra zajmuje jeden bajt).

Liczby dziesiętne mają zmienną długość, liczoną w bajtach od 1 do 16, i składać się mogą maksymalnie z 31 cyfr dziesiętnych oraz ze znaku liczby.

Format wszystkich instrukcji dziesiętnych jest typu SS. Format ten określa w pamięci dwa dowolne ciągi (pola) cyfr dziesiętnych, przez podanie adresów A1 i A2 lewych skrajnych bajtów tych pól oraz parametrów L1 i L2 określających ilość dodatkowych bajtów położonych na prawo od bajtu określonego adresem. Określone pola zawierają odpowiednio pierwszy i drugi argument operacji dziesiętnej; zaś po operacji zapisuje się wynik działania w miejsce pierwszego argumentu. Funkcję instrukcji DODAJ DZIESIĘTNIE, którą podaje się dla przykładu, można przedstawić następująco:

a) (E1, L1)+(E2, L2) → E1, L1; czyli dodaje się dwa argumenty, zaś sumę umieszcza się w miejsce pierwszego;
b) wpisuje się 0, 1, 2 lub 3 do rejestru KW, jeśli odpowiednio suma dziesiętna jest równa zeru, jest ujemna, dodatnia, lub nie mieści się w polu pierwszego argumentu.

Jeśli pole pierwszego argumentu jest zbyt krótkie, aby pomieścić wszystkie znaczące cyfry wyniku operacji dziesiętnej, to występuje tzw. nadmiar dziesiętny. Nadmiar ten może spowodować przerwanie programowe, jeśli jego maska (tj. bit 37 słowa SSP) jest w stanie 1. Przerwanie programowe zostanie również spowodowane w czasie wykonywania działań dziesiętnych, jeśli będzie miało miejsce którekolwiek z poniższych zdarzeń:

— niedozowolony kod operacji, tzn. pojawiła się w maszynie instrukcja dziesiętna, pomimo że w danej maszynie nie wbudowano sprzętowej przystawki arytmetyki dziesiętnej;
— próba naruszenia blokady (ochrony) pamięci, tzn. określony przez adres argumentu pamięciowego klucz ochrony bloku nie jest zgodny z kluczem przechowywanym w słowie SSP;
— błędny adres, tzn. adres argumentu wykracza poza wielkość zainstalowanej pamięci operacyjnej w danej maszynie;
— błędna specyfikacja, tzn. długość dzielnika lub mnożnika przekracza 16 cyfr łącznie ze znakiem, względnie przekracza długość dzielnej lub mnożnej;
— błędne dane, tzn. w argumencie określonego działania dziesiętnego znajduje się nieprawidłowy kod cyfry (tj. kod większy od 9) bądź nieprawidłowy kod znaku, bądź pola argumentów tych działań nakładają się w niewłaściwy sposób, bądź wreszcie mnożna ma niewystarczającą ilość początkowych zer, zapewniających zmieszczenie się znaczącej części iloczynu w pierwszym polu.

3.3.6. Działania niearytmetyczne

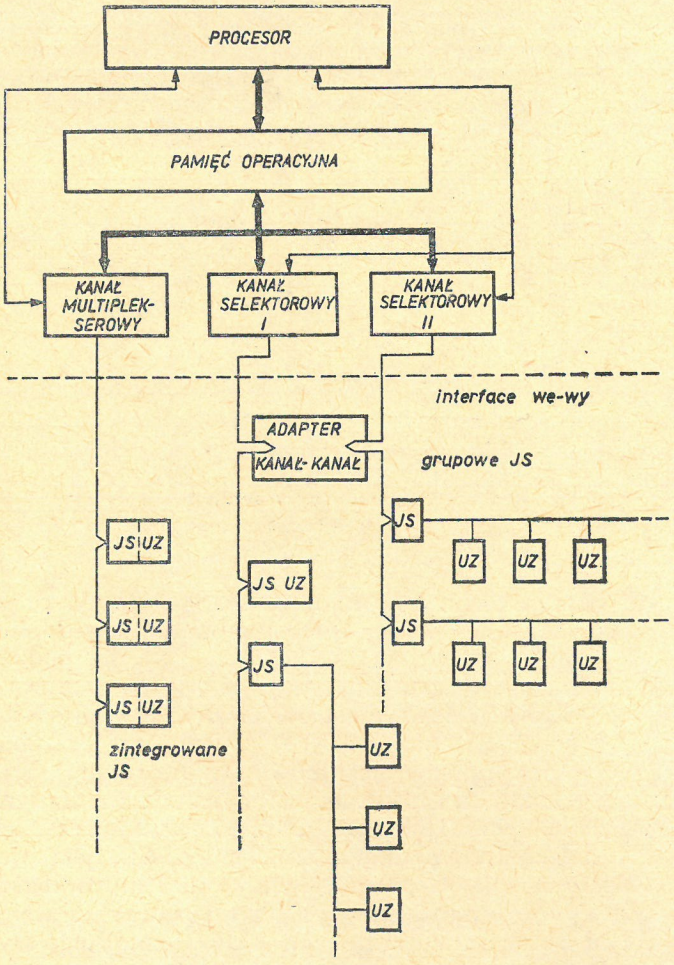
Na efektywność i zawartość programów mają wpływ nie tylko dotychczas opisane instrukcje arytmetyczne i instrukcje skoku, lecz także różne inne grupy instrukcji powodujące wykonanie takich działań, jak operacje logiczne, przesuwania, przenoszenia danych, porównania, redagowania, testowania, tłumaczenia i innych. Operacje niearytmetyczne są szczególnie przydatne w programach translacji z języków wysokiego poziomu.

Argumenty operacji niearytmetycznych traktowane są przeważnie jako 8-bitowe bajty lub jako ciągi takich bajtów (o długości do 256 bajtów). W niektórych przypadkach poszczególne połowy bajtów traktowane są jako oddzielne całości, a w jeszcze innych przypadkach wykonuje się działania na poszczególnych bitach.

4. Organizacja kanałów wejścia-wyjścia

4.1. Wprowadzenie

Kanały wejścia-wyjścia zajmują się przesyłaniem informacji pomiędzy pamięcią operacyjną a urządzeniami zewnętrznymi maszyny. Sposób połączenia kanałów JS EMC z urządzeniami zewnętrznymi (UZ) jest zilustrowany na rys. 4.1.



Rys. 4.1. Ogólna struktura systemu we-wy
UZ — urządzenie zewnętrzne, JS — jednostka sterująca UZ

W tym rozdziale zostanie podany krótki opis cech i zasad działania systemu wejścia-wyjścia. Opis natomiast konkretnych implementacji kanałów we-wy oraz ich parametry techniczne zostaną podane w części IV.

Przyjęta koncepcja organizacji systemu wejścia-wyjścia pozwala zachować zgodność urządzeń zewnętrznych, tzn. dowolne urządzenie zewnętrzne JS EMC może być podłączone do każdej jednostki centralnej JS EMC.

Przyjęta koncepcja organizacji systemu wejścia-wyjścia wyróżnia dwa poziomy środków technicznych. W skład pierwszego (tj. niższego) poziomu wchodzą konkretne urządzenia zewnętrzne oraz ich jednostki sterujące. Natomiast poziom wyższy zawiera środki organizujące pracę wszystkich urządzeń zewnętrznych wchodzących w dany zestaw komputerowy oraz organizuje przebieg przesyłań danych na drodze PAO — jednostka sterująca UZ.

Jednostki sterujące urządzeniami zewnętrznymi dopasowują ogólne zasady logicznego i technicznego działania interface'u we-wy do specyfiki danego urządzenia zewnętrznego. Dzięki temu można było przyjąć, niezależnie od charakterystyki technicznej danego urządzenia, ujednolicone zasady sterowania na poziomie wyższym. Główną funkcją jednostki sterującej urządzenia zewnętrznego jest przyjęcie i zinterpretowanie poleceń otrzymanych ze strony kanału oraz generowanie odpowiednich ciągów sygnałów sterujących potrzebnych do wykonania danego polecenia.

Jednostka sterująca może stanowić samodzielną konstrukcję (jak np. w przypadku jednostki sterującej pamięciami dyskowymi lub taśmowymi) lub może być zintegrowana konstrukcyjnie ze swoim urządzeniem zewnętrznym (jak w przypadku prostych urządzeń zewnętrznych). Ponadto niektóre jednostki sterujące są związane wyłącznie z jednym urządzeniem zewnętrznym, zaś inne sterują pracą grupy urządzeń zewnętrznych (przeważnie tego samego typu). W Jednolitym Systemie opracowane dotychczas pamięci dyskowe, taśmowe i bębnowe mają grupowe jednostki sterujące. Większość grupowych jednostek sterujących funkcjonuje w ten sposób, że w danym przedziale czasowym organizują przesyłanie bloku danych tylko do jednego (lub z jednego) spośród urządzeń zewnętrznych podłączonych do tej jednostki sterującej. Przełączenie współpracy jednostki sterującej z jednego urządzenia na drugie odbywa się poprzez nowe polecenie, w którym określa się również numer urządzenia przełączanego.

W Jednolitym Systemie występują również jednostki sterujące, które organizują jednoczesną pracę wielu urządzeń zewnętrznych. Są to tzw. multipleksery lokalne lub zdalne, o których będzie mowa później.

Środki techniczne wyższego poziomu systemu wejścia-wyjścia, tj. kanały — które traktuje się funkcjonalnie jako samodzielne zespoły — realizują przesyłania danych do lub z PAO po zapoczątkowaniu ich działania przez procesor (np. in-

strukcją START WE-WY). Przebieg przesyłania danych może się odbywać w pełni autonomicznie (tzn. bez wstrzymania pracy procesora) lub nieautonomicznie. Autonomiczne przesyłanie danych wymaga rozbudowanych kanałów, w których znajdują się wszystkie niezbędne rejestry do przechowywania informacji sterujących przesyłaniem danych oraz niezależne układy do aktualizacji informacji. W związku z tym kanały autonomiczne są instalowane tylko w dużych maszynach Jednolitego Systemu. Natomiast w małych i średnich maszynach wbudowywane są tzw. kanały zintegrowane, które mają w dużej części wspólny z procesorem (tzw. dzielony) sprzęt: np. wspólny arytmometr, wspólna pamięć mikroprogramów, wspólne szyny do PAO itp.

Współpraca kanałów z jednostkami sterującymi odbywa się za pomocą zestandaryzowanej szyny, zwanej interface'em wejścia-wyjścia. Szyna interface'u pozwala podłączyć do 8 jednostek sterujących do każdego kanału, przy czym łączna ilość urządzeń zewnętrznych nie może przekroczyć liczby 256.

Tryby wykonywania operacji w kanale. Przesyłanie danych pomiędzy pamięcią a urządzeniami we-wy może odbywać się dwoma sposobami, inaczej — trybami. Rozróżnia się tryby: selektorowy i multiplekserowy.

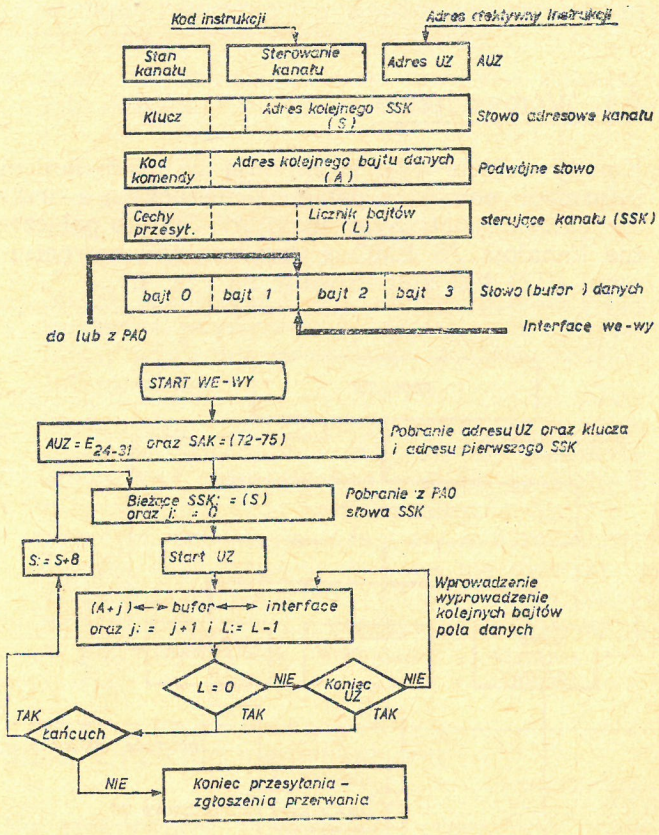
W trybie selektorowym urządzenie we-wy monopolizuje szynę interfejsu we-wy i pozostaje logicznie związane z kanałem na czas przesyłania jednej sekwencji danych. Sekwencja taka może składać się z kilku bajtów, może być całym blokiem danych, a także ciągiem bloków uzupełnionym informacjami sterującymi.

W trybie multiplekserowym sprzęt kanału może być wykorzystywany jednocześnie przez wiele urządzeń we-wy. W tym przypadku wszystkie operacje we-wy dzielone są na krótkie odcinki czasowe, w których przez szynę we-wy przechodzi jedynie część (porcja) informacji i w których tylko jedno urządzenie jest logicznie połączone z kanałem. Kanał łączy się z urządzeniem w odpowiedzi na jego żądanie i jest zajęty daną operacją jedynie w czasie potrzebnym na przesłanie jednej porcji informacji. Porcja taka może składać się z jednego lub kilku bajtów danych. Jeśli porcja ta składa się z jednego bajtu, dany kanał nazywa się bajtowo-multiplekserowy, jeśli zaś z wielu bajtów — blokowo-multiplekserowy.

Mała sekwencja informacji może być przesyłana zarówno w trybie selektorowym jak i multiplekserowym. Przy wyborze trybu istotna jest długość sekwencji. Przyjmuje się np., że jeśli sekwencja danych spowoduje podłączenie urządzenia do kanału na dłużej niż 100 mikrosekund, wtedy kanał sam przechodzi na tryb selektorowy (czas ten jest różny w różnych modelach JS EMC, np. w R-32 wynosi około 80 μs). Zatem tryb pracy kanału jest określany charakterem urządzenia, a nie programem.

Opisane dwa tryby pracy powodują odmienną reakcję kanału. Kanał inaczej reaguje na instrukcje we-wy pracując w trybie selektorowym, a inaczej w trybie multiplekserowym. W pierwszym przypadku jest on zajęty dla nowych instrukcji we-wy, w drugim, mogąc współpracować z wieloma urządzeniami, akceptuje instrukcje nakazujące zainicjowanie operacji na nowym urządzeniu.

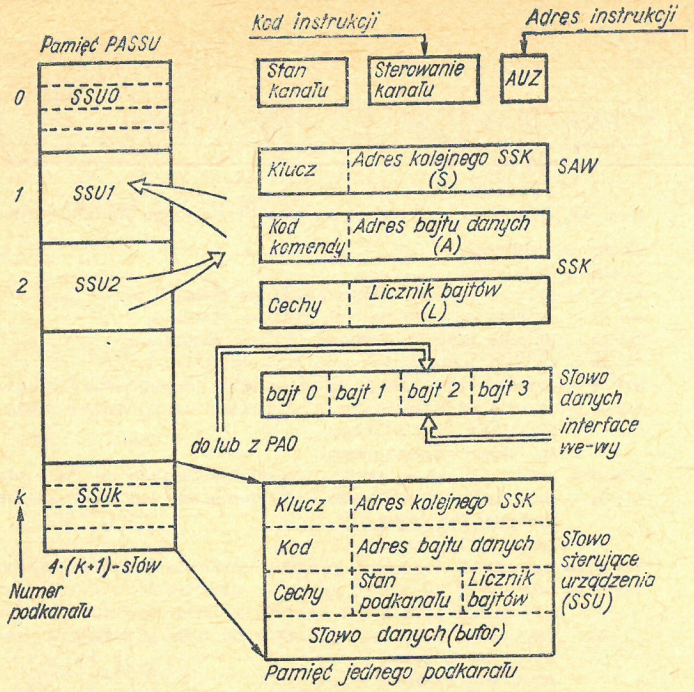
Typy kanałów. W maszynach Jednolitego Systemu występują dwa typy kanałów: kanał selektorowy i kanał multiplekserowy (patrz rys. 4.2a i 4.2b). Kanał multiplekserowy może pracować zarówno w trybie selektorowym, jak i w trybie multiplekserowym, w zależności od tego, jakie urządzenie obsługuje. Natomiast kanały typu selektor mogą pracować tylko w trybie selektorowym.



Rys. 4.2a. Rejestry i uproszczony schemat działania kanału selektorowego

Środki techniczne kanału, które służą do przeprowadzania operacji przesyłania danych do lub z jednego urządzenia we-wy, nazywa się podkanałami. Każdy podkanał zawiera własną pamięć — tj. własny zestaw rejestrów zwany słowem sterującym urządzenia (SSU) — służącą do przechowywania adresów, liczników oraz informacji o stanie podkanału i informacji sterujących związanych z daną operacją we-wy. Pamięć słów sterujących urządzeń (lub podkanałów), tj. PASSU, jest niedostępna dla programu. Tylko kanał multiplekserowy może pisać lub czytać z tej pamięci. Pamięć PASSU stanowi w dużych EMC Jednolitego Systemu samodzielny blok pamięci (np. o pojemności 4 Kbajtów), natomiast w średnich i małych maszynach jest częścią pamięci operacyjnej (dostępną tylko dla kanału).

Tryb pracy kanału zależy od tego, czy ma on jeden, czy więcej podkanałów. Kanał selektorowy ma tylko jeden podkanał i pracuje tylko w trybie selektorowym. Sekwencja przesyłanych danych obejmuje cały blok danych, a nawet — w przypadku tzw. łańcucha komend — ciąg bloków. Kanał selektorowy nie może pracować w trybie multiplekserowym i w związku z tym może brać



Rys. 4.2b. Rejestry kanału multiplekserowego
Schemat inicjacji i działania kanału multiplekserowego w trybie elektronowym jest taki sam jak schemat kanału selektorowego. Natomiast w trybie multiplekserowym — po pobraniu słowa adresowego i słowa sterującego kanału do odpowiednich rejestrów kanału oraz po zastartowaniu urządzenia — pamięta się stan rejestrów kanału w odpowiednim 4-słowowym polu pamięci słów sterujących urządzenia (PASSU). Po przechowaniu tych informacji kanał przechodzi w stan oczekiwania na zgłoszenia obsługi od urządzeń. Jeśli dane urządzenie (na rys. drugie) jest gotowe do przekazania lub odebrania bajtu, kanał pobiera pełną informację sterującą danego podkanału z PASSU i czyta z szyny interface'u lub pisze do szyny kolejny bajt danych. Następnie aktualizuje się wielkości *A* i *L*, a także zapisuje się lub odczytuje z PAO — w miarę potrzeby — słowo danych. Po tej aktualizacji zapisuje się stan rejestrów kanału do pola danego podkanału w PASSU i kanał znów przechodzi w stan oczekiwania

udział w jednej tylko operacji przesyłania danych na raz. Inne urządzenia podłączone do tego kanału mogą jednocześnie wykonywać uprzednio zainicjowane operacje, o ile nie wymagają one współpracy z kanałem (np. przewijanie taśmy do początku).

Kanał multiplekserowy zawiera wiele podkanałów i może pracować w obydwu trybach. W trybie multiplekserowym może on współpracować jednocześnie z wieloma urządzeniami, z których każde używa oddzielnego podkanału. W przypadku pracy w trybie selektorowym tylko jedno z urządzeń podłączonych do kanału może przesyłać dane. Tryb pracy zależy od urządzenia we-wy i może zmieniać się w dowolnej chwili. Przesyłanie danych związanych z jedną operacją może odbywać się częściowo w trybie selektorowym i częściowo w trybie multiplekserowym. Przeważnie urządzenia o dużej szybkości przesyłania współpracują z kanałem w trybie selektorowym, a urządzenia wolne — w trybie multiplekserowym.

Kanały multiplekserowe mogą mieć różne ilości podkanałów. Gdy kanał multiplekserowy pracuje w trybie multiplekserowym, na różnych podkanałach mogą dokonywać się jednocześnie różne operacje (po jednej operacji na podkanał). Ilość informacji przesyłanych przez kanał nie może jednak przekraczać jego maksymalnej wydajności. Z punktu widzenia programisty każdy z podkanałów multipleksera jest niezależnym kanałem typu selektorowego. Gdy kanał multiplekserowy przesyła dane w trybie selektorowym, wówczas podkanał związany z tym przesyłaniem wykorzystuje wszystkie możliwości transmisyjne kanału. Pozostałe podkanały nie mogą współpracować z urządzeniami aż do zakończenia przesyłania sekwencji danych.

Podkanały dzielone i niedzielone. Podkanały multipleksera mogą być dzielone bądź niedzielone. Przez kanał niedzielony rozumie się taki podkanał, który współpracuje z jednym tylko urządzeniem we-wy. Niedzielone podkanały stosuje się dla urządzeń, które nie mają żadnych ograniczeń co do równoczesności przesyłania danych (np. drukarka wierszowa, czytnik kart itp.).

Natomiast podkanał nazywa się dzielonym, jeżeli współpracuje z kilkoma urządzeniami we-wy. Niemniej w danej chwili może przesyłać informację tylko jedno z podłączonych urządzeń. Podkanały dzielone stosuje się dla takich urządzeń, które mają jedną wspólną jednostkę sterującą (np. pamięci taśmowe). Należy jednak zwrócić uwagę, że podkanał nie musi być koniecznie dzielony, nawet wtedy, gdy urządzenia we-wy dzielą między siebie jednostkę sterującą. Na przykład w multiplekserze przesyłania danych JS-8402, który jest wspólną jednostką sterującą dla 176 różnych punktów abonenckich, każda linia transmisji (lub każdy punkt abonencki) występuje jako podkanał niedzielony.

Adresem urządzenia może być dowolna liczba z przedziału, 0—255, co umożliwia adresowanie do 256 urządzeń na kanał. W przypadku kanału multiplekserowego 8-bitowy adres urządzenia identyfikuje podkanał, jednostkę sterującą oraz urządzenie we-wy. Adresy urządzeń 0—127 należą do podkanałów niedzielonych. Każdemu takiemu podkanałowi odpowiada dokładnie jeden adres urządzenia. Natomiast adresy 128—255 mogą odpowiadać niedzielonym lub dzielonym podkanałom. Dzielonym podkanałom przypisane są całe zbiory kolejnych adresów. Jeżeli multiplekser ma dzielone i niedzielone podkanały o adresach z przedziału 128—255, to podkanałom niedzielonym przypisywane są niższe adresy niż dzielonym. Ilość i typ podkanałów dostępnych w kanale multiplekserowym oraz przyporządkowanie adresów zależą od modelu Jednolitego Systemu i są różne dla różnych zestawów.

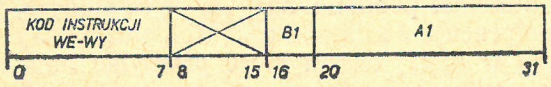
4.2. Operacje wprowadzania-wyprowadzania informacji

Operacje wejścia-wyjścia są zapoczątkowywane i sterowane za pomocą kilku typów operacji. Są to tzw. instrukcje, komendy i polecenia. Instrukcje wejścia-wyjścia (podobnie jak np. instrukcje arytmetyczne) występują na poziomie programu i interpretowane są przez procesor. Natomiast komendy wejścia-wyjścia są rozszyfrowywane i wykonywane przez kanały oraz przez urządzenia zewnętrzne. Komendy stosuje się do zapoczątkowywania takich operacji, jak czytanie i pisanie danych, badanie, sterowanie itp. Instrukcje programu i komendy — zawarte w tzw. słowie sterującym kanału — pobierane są z pamięci operacyjnej i są w zasadzie jednakowe dla wszystkich

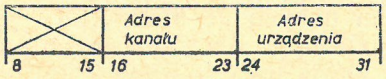
urządzeń zewnętrznych. Polecenia natomiast są rozszyfrowywane i wykonywane tylko przez jednostkę sterującą urządzenia zewnętrznego. Polecenia określają specyficzne operacje związane z typem urządzenia zewnętrznego (np. przewijanie taśm).

4.2.1. Instrukcje wejścia-wyjścia

Procesor zapoczątkowuje pracę kanałów za pomocą czterech uprzywilejowanych instrukcji o nazwach: START WE-WY, TESTUJ WE-WY, STOP WE-WY i TESTUJ KANAŁ. Instrukcje wejścia-wyjścia mają format SI, a ich struktura pokazana jest na rysunku 4.3. Adres urządzenia i adres kanału, którego dotyczy dana instrukcja, jest określany przez jej efektywny adres E1=(B1)+A1, pokazany na rysunku 4.4.



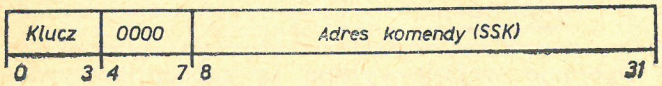
Rys. 4.3. Struktura instrukcji wejścia—wyjścia



Rys. 4.4. Adres kanału i urządzenia określony przez 24-bitowy adres efektywny instrukcji we-wy

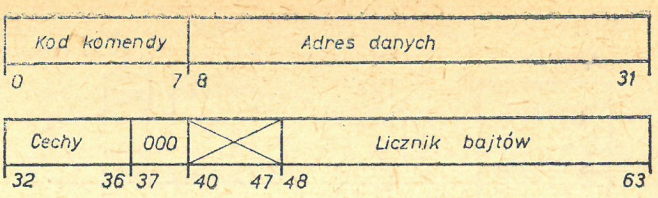
Podstawową instrukcją, za pomocą której rozpoczyna się jedną z 6 komend wejścia-wyjścia, jest instrukcja START WE-WY. Adres tej instrukcji określa tylko numer urządzenia i numer kanału, do którego podłączone jest dane urządzenie zewnętrzne. Niemniej dla zorganizowania przesyłania bloku danych — np. z PAO do określonego urządzenia zewnętrznego — potrzebne są również inne informacje, określające między innymi: początek pola bloku danych w PAO, wielkość pola danych, typ operacji we-wy (tj. typ komendy i polecenia), sposób pracy kanału w czasie przesyłania (cechy przesyłania), klucz ochrony pamięci operacyjnej itp.

Powyższe informacje przygotowywane są przez program użytkowy i sterujący — w PAO, przed zastosowaniem instrukcji START WE-WY. Natomiast instrukcja uruchamia kanał, który sam już pobiera z PAO kolejno:



Rys. 4.5. Struktura słowa adresowego kanału (SAK)

a) słowo adresowane kanału (rys. 4.5), które znajduje się w stałym miejscu pamięci operacyjnej (komórka 72—75); słowo to określa klucz — tj. klucz ochrony tych bloków PAO, z którymi kanał może się kontaktować w czasie przesyłania danych — oraz adres pierwszego tzw. słowa sterującego kanału (SSK); słowo SSK zajmuje zawsze 8 bajtów w dowolnym miejscu pamięci operacyjnej;



Rys. 4.6. Struktura podwójnego słowa sterującego kanału (SSK)

Cechy przesyłania:
Cechy przesyłania:
bit 32 — określa łańcuch danych (ŁD) w stanie 1,
bit 32 — określa łańcuch danych (ŁD) w stanie 1,
bit 33 — określa łańcuch komend (ŁK) w stanie 1,
bit 33 — określa łańcuch komend (ŁK) w stanie 1,
bit 34 — określa sygnalizację w przypadku niepoprawnej długo-
bit 34 — określa sygnalizację w przypadku niepoprawnej długości bloku przesyłanego,
ści bloku przesyłanego,
bit 35 — powoduje, że w czasie przesyłania danych do PAO nie
bit 35 — powoduje, że w czasie przesyłania danych do PAO nie zachodzi ich zapis,
zachodzi ich zapis,
bit 36 — powoduje, że kanał generuje przerwanie programowe,
bit 36 — powoduje, że kanał generuje przerwanie programowe, gdy kanał przechodzi na kolejne SSK w łańcuchu da-
gdy kanał przechodzi na kolejne SSK w łańcuchu danych lub komend,
nych lub komend,
bit 37 — określa, że adres danych w początkowym SSK jest wir-
bit 37 — określa, że adres danych w początkowym SSK jest wirtualny
tualny

b) słowo SSK, które zawiera (rys. 4.6.): początek pola danych w PAO oraz długość tego pola liczoną w bajtach (tzw. licznik bajtów), kod komendy oraz cechy przebiegu operacji wejścia-wyjścia.

Instrukcja START WE-WY przygotowuje również rejestr kodu warunku (KW), z którego program dowiaduje się (przez testowanie stanu KW) o tym, czy dana operacja została przyjęta — przez kanał i urządzenie — do wykonania. Rejestr kodu warunku przyjmuje na ogół wartości:

0 — dla zaznaczenia, że komenda określona przez SSK została przyjęta i kanał przystąpił do jej wykonania;
1 — dla zaznaczenia, że komenda nie została przyjęta z powodów opisanych przez tzw. słowo stanu kanału (SSTK); słowo to zapisuje kanał w polu 64—71 pamięci operacyjnej i określa między innymi stan kanału i stan urządzenia;
2 — dla zaznaczenia, że kanał lub podkanał jest zajęty;
3 — dla zaznaczenia, że urządzenie jest nieoperatywne (tzn. uszkodzone lub nieprzygotowane do pracy).

Kolejna instrukcja wejścia-wyjścia, która brzmi TESTUJ WE-WY, testuje stan zespołów we-wy i ustawia rejestr kodu warunków KW, stosownie do stanu tych zespołów. Można np. zbadać za pomocą tej instrukcji, czy dany kanał jest wolny, czy też zapamiętane zostało w stałym miejscu pamięci słowo stanu kanału (SSTK). Słowo SSTK szczegółowo opisuje stan urządzenia zewnętrznego i jego jednostki sterującej.

Instrukcja TESTUJ KANAŁ stosowana jest w przypadkach, gdy chcemy sprawdzić stan adresowanego kanału. Wynik testowania zostaje zanotowany w rejestrze kodu warunków i może świadczyć o tym, że dany kanał jest wolny (KW=0), kanał oczekuje na przerwanie (KW=1), kanał działa w trybie selektorowym (KW=3) lub dany

kanał jest nieoperatywny. W czasie wykonywania tej instrukcji kanał nie zmienia swojego stanu, czyli instrukcja ta nie zakłóca przebiegu pracy kanału.

Instrukcja STOP WE-WY powoduje wstrzymanie aktualnie wykonywanej operacji wejścia-wyjścia w określonym urządzeniu zewnętrznym, w podkanale lub kanale.

4.2.2. Komendy wejścia-wyjścia

Instrukcja START WE-WY może uruchomić jedną z 6 operacji we-wy, których kod określa tzw. kod komendy (rys. 4.6). Komendy te mają następujące kody i nazwy:

Kod komendy	Nazwa komendy
XXXX0000	kod nieprawidłowy
MMMM0100	BADAJ
XXXX1000	PRZEŁĄCZ W KANALE
MMMM1100	CZYTAJ WSTECZ
MMMMMM01	PISZ
MMMMMM10	CZYTAJ
MMMMMM11	STERUJ

gdzie litera X oznacza, że wartość tej pozycji jest ignorowana, zaś litera M oznacza tzw. bity modyfikujące, które bliżej określają daną operację z punktu widzenia urządzenia zewnętrznego.

W tym celu kod komendy nie tylko jest interpretowany w kanale, lecz także zostaje przesłany do jednostki sterującej urządzenia zewnętrznego (z wyjątkiem komendy PRZEŁĄCZ, która jest wykonywana tylko w kanale).

Poniżej opisano krótko funkcję komend kanałowych.

Komenda CZYTAJ powoduje wprowadzenie bloku danych z adresowanego urządzenia zewnętrznego do zadanego pola pamięci operacyjnej. Początek pola w PAO oraz ilość bajtów do wprowadzenia zadaje się przez słowo sterujące kanału (rys. 4.6). Proces wprowadzania bajtów danych do PAO odbywa się w kolejności rosnących adresów, począwszy od adresu określonego przez SSK. Oznacza to, że pierwszy bajt danych zapisywany zostaje w miejscu pamięci o adresie A, drugi w A+1, zaś ostatni w A+L−1 (gdzie A jest początkowym adresem zapisanym w bitach 8—31 słowa SSK, zaś L początkowym stanem licznika bajtów określonym przez bity 48—63 tego SSK).

Komenda PISZ powoduje wyprowadzenie bloku danych z zadanego pola pamięci operacyjnej do adresowanego urządzenia zewnętrznego. Proces wyprowadzania danych odbywa się w kolejności rosnących adresów miejsc pamięci.

Komenda CZYTAJ WSTECZ powoduje wprowadzenie bloku danych z adresowanego urządzenia zewnętrznego do zadanego pola pamięci operacyjnej. Proces wprowadzania bajtów danych do PAO odbywa się w kolejności malejących adresów miejsc pamięci, począwszy od adresu zapisanego w SSK. Oznacza to, że w czasie wprowadzania danych zarówno licznik bajtów, jak i adres danych zostają pomniejszone. Natomiast przy wykonywaniu pozostałych komend pomniejszony zostaje tylko licznik bajtów, zaś adres danych zostaje powiększony po przesłaniu kolejnego bajtu danych. Komenda CZYTAJ WSTECZ jest przeważnie stosowana w odniesieniu do pamięci taśmowej, gdzie pozwala efektywnie wykorzystać czas przesuwu taśmy do tyłu.

Komenda STERUJ jest stosowana w celu rozpoczęcia operacji we-wy w samym urządzeniu zewnętrznym (np. przewijanie taśm magnetycznych, nastawianie głowicy na dysku itp.). Operacja ta nie ma na celu wprowadzenia lub wyprowadzenia danych, chociaż w niektórych przypadkach wykorzystuje się „adres danych" i „licznik bajtów" ze słowa SSK. W tych przypadkach chodzi po prostu o przesłanie do urządzenia zewnętrznego z PAO więcej informacji sterujących opisujących jednoznacznie treść polecenia do wykonania w urządzeniu zewnętrznym lub w jego jednostce sterującej. W prostych przypadkach dane polecenie jest określane przez 6 bitów M komendy STERUJ, natomiast w przypadkach złożonych (gdy np. trzeba określić adres na dysku), podaje się dodatkowe informacje sterujące w polu pamięci operacyjnej określonym przez słowo SSK.

Komenda BADAJ powoduje przeczytanie z urządzenia zewnętrznego do PAO — według parametrów ze słowa SSK — różnych informacji szczegółowo opisujących stan urządzenia zewnętrznego. Za pomocą tej komendy czyta się informacje dotyczące nieprawidłowości wykrytych przy wykonywaniu ostatniej operacji wejścia-wyjścia oraz informacje o obecnym stanie urządzenia zewnętrznego. Informacje czytane komendą BADAJ są bardziej szczegółowe od informacji zawartych w tzw. słowie stanu kanału (gdzie przeznaczono tylko 8 bitów dla opisu stanu urządzenia). Komenda BADAJ powoduje przeważnie przeczytanie dwóch bajtów opisujących stan urządzenia oraz szereg dalszych bajtów informacji typu diagnostycznego (zależnie od rodzaju urządzenia zewnętrznego). Pierwszych 6 bitów pierwszego bajtu przeczytanego tą komendą opisuje różne nieprawidłowości; znaczenie tych bitów jest jednakowe dla wszystkich typów urządzeń zewnętrznych.

Bit	Opis znaczenia danego bitu
0	Odrzucenie poprzedniej komendy. Oznacza to, że miał miejsce błąd programowy (np. próba czytania wstecz z dysku, próba zapisu do zablokowanej taśmy magnetycznej, próba zapisu na dysku bez wcześniejszego określenia bloku danych na dysku itp.).
1	Wymagana interwencja operatora. Oznacza to, że ostatnia operacja nie została wykonana przez dane urządzenie, gdyż wymagana jest interwencja operatora (np. z powodu braku papieru w drukarce, braku kart w czytniku itp.).
2	Błąd parzystości. Oznacza to, że do urządzenia lub do jego jednostki sterującej przesłano kod komendy lub kod danych z błędem parzystości.
3	Błąd sprzętu. Oznacza to, że w czasie wykonywania poprzedniej operacji wykryto błąd, który spowodowało samo urządzenie zewnętrzne (np. wykryto błąd parzystości w buforze drukarki wierszowej).
4	Błąd danych. Oznacza to, że w urządzeniu lub w jego jednostce sterującej wykryto błąd danych różny od błędu parzystości. Bit ten sygnalizuje różne błędy związane z nośnikiem informacji danego urządzenia (np. niepoprawny kod na karcie lub na taśmie).
5	Brak obsługi kanału. Bit ten zapala się, gdy

kanał zbyt długo nie odpowiada na żądania przyjęcia lub przekazania kolejnej informacji z lub do urządzenia zewnętrznego. Powyższa sytuacja ma miejsce, gdy szybkość pracy kanału (np. multiplekserowego) jest mniejsza od sumarycznej szybkości aktywnych urządzeń zewnętrznych.

Komenda PRZEŁĄCZ W KANAŁ powoduje, że kanał pobiera nowe SSK z miejsca pamięci operacyjnej, określonego przez pole „adres danych" bieżącego SSK. W ten sposób za pomocą tej komendy można pobrać kolejne SSK nie sąsiadujące z bieżącym SSK.

4.2.3. Sposoby przesyłania danych

Operacje wejścia-wyjścia, inicjowane instrukcją START WE-WY, mogą przebiegać w różny sposób, zależnie od programowo ustawianych tzw. cech przesyłania (rys. 4.6). Cechy przesyłania są określane przez odpowiednie bity słowa SSK. Cechy te pozwalają między innymi przesłać całą sekwencję pól i danych za pomocą jednej instrukcji, przy czym pola danych mogą leżeć w niespójnych obszarach pamięci operacyjnej. Można również polecić kanałowi, aby przy przesyłaniu całego ciągu pól danych składał meldunki przy przejściu z jednego pola na drugie.

Poniżej opiszemy sposoby przesyłania danych zadawane za pomocą bitów części „cechy" słowa SSK.

Termin „pole danych" oznacza tutaj pole opisane przez jedno SSK, tzn. przez części „adres danych" i „licznik bajtów" danego SSK. Natomiast przez blok danych rozumie się porcję informacji zapisanych (lub przeczytanych) pomiędzy uruchomieniem a zatrzymaniem urządzenia zewnętrznego. Oczywiście długość bloku danych zależy od typu urządzenia i używanego przez to urządzenie nośnika informacji. Może to być np. informacja jednej karty dziurkowanej, jednego wiersza druku, lub zapis na taśmie magnetycznej ograniczony przerwami międzyblokowymi.

Instrukcja START WE-WY może zapoczątkować w urządzeniu operację określoną przez jedno SSK lub przez grupę SSK. Zatem przesyłane dane można traktować jako jedno pole danych lub jako ciąg pól danych. Jeśli w jednej operacji wejścia-wyjścia występuje grupa słów SSK, mówi się, że mamy do czynienia z tzw. łańcuchem SSK. Normalnie, kolejne SSK danego łańcucha zajmują kolejne podwójne słowa w pamięci operacyjnej, czyli początkowe adresy dwóch sąsiadujących ze sobą SSK różnią się o 8. Dzięki temu kanał może, po wyczerpaniu bieżącego SSK, pobrać następne SSK z pola o adresie zwiększonym o 8 w stosunku do pola bieżącego SSK. Niemniej dopuszcza się przypadek kiedy kolejne SSK leży w dowolnej odległości od bieżącego SSK. Zatem ciąg słów nie musi leżeć w spójnym obszarze pamięci. W takim przypadku przechodzi się z bieżącego SSK na następne SSK za pomocą komendy PRZEŁĄCZ W KANAŁ, która powoduje przełączenie się kanału do innego pola słów SSK.

W organizacji kanałów wyróżnia się dwa typy łańcucha: łańcuch danych i łańcuch komend, określonych odpowiednio za pomocą bitów 32 i 33 słowa SSK.

Łańcuch danych pozwala przesyłać dane do lub z nieciągłych obszarów pamięci operacyjnej, przy czym zmiana pola, tj. zmiana SSK, odbywa się wyłącznie w kanale, a urządzenie zewnętrzne nie jest powiadamiane o tej zmianie i kontynuuje wykonanie operacji przesyłania z pełną szybkością. Oczywiście w łańcuchu danych, gdzie przez cały czas zachodzi określony i ten sam typ operacji we-wy, nie może nastąpić zmiana kodu komendy i dlatego kod komendy nowego SSK jest ignorowany (z wyjątkiem kodu PRZEŁĄCZ W KANAŁ).

Łańcuch komend ma miejsce, gdy pobiera się nie tylko nowe SSK z pamięci, lecz także, gdy pobiera się nowy kod komendy (z nowego SSK). W ten sposób kanał nie tylko zmienia pole danych, lecz może również zmienić typ wykonywanej operacji we-wy. W łańcuchu komend pobiera się nowe SSK i inicjuje się nową operację dopiero po odebraniu sygnału „koniec urządzenia" dla bieżącej operacji. Sygnał ten świadczy, że przesłany został cały blok danych. Łańcuch komend pozwala zatem zorganizować — za pomocą jednej instrukcji — przesłanie ciągu bloków danych, bez konieczności informowania programu o zakończeniu poszczególnych operacji łańcucha. Łańcuch komend umożliwia także nastawianie określonego podkanału do wykonywania pewnych funkcji pomocniczych w urządzeniu zewnętrznym (np. wybór głowicy dysku), po zakończeniu których od razu przystępuje się do przesyłania danych. W ten sposób program nie musi ingerować po zakończeniu każdej operacji wejścia-wyjścia. Oczywiście kanał przechodzi automatycznie do nowej operacji łańcucha komend tylko w przypadku prawidłowego zakończenia bieżącej operacji wejścia-wyjścia.

Organizacja kanału dopuszcza również łączenie łańcucha danych i łańcucha komend. W takim przypadku pierwsze SSK związane z daną operacją wejścia-wyjścia rozpoczyna łańcuch danych, zaś ostatnie SSK związane z tą operacją rozpoczyna łańcuch komend dla następnych operacji wejścia-wyjścia.

Kolejną cechą przesyłania jest tzw. czytanie z pomijaniem zapisu. Cecha ta jest zadawana przez jedynkę w bicie 35 słowa SSK i oznacza, że w czasie czytania informacji z urządzenia zewnętrznego do PAO (tj. w czasie operacji CZYTAJ, CZYTAJ WSTECZ I BADAJ) pomija się proces zapisu do PAO. Przy czym wszystkie mechanizmy w kanale (z wyjątkiem zwrotów dla zapisu) i urządzeniu zewnętrznym działają tak, jak przy normalnym czytaniu danych. Zwłaszcza identycznie działają mechanizmy aktualizacji adresów i liczników oraz pobierania kolejnych słów sterujących kanału.

Łącznie z cechą czytania z pomijaniem zapisu można stosować łańcuch danych lub łańcuch komend. Jeśli zatem w łańcuchu danych pobrane kolejne SSK ma w bicie 35 zero, to przestaje działać zakaz zapisu do PAO i w tym samym rozpoczyna się normalny proces czytania danych z urządzenia i zapis tych danych do PAO. Można zatem — poprzez łączne zastosowanie łańcucha danych i czytania z pomijaniem zapisu — przeczytać i zapisać do PAO tylko wybrane fragmenty bloku danych.

Jedynka w bicie 36 słowa SSK zadaje cechę

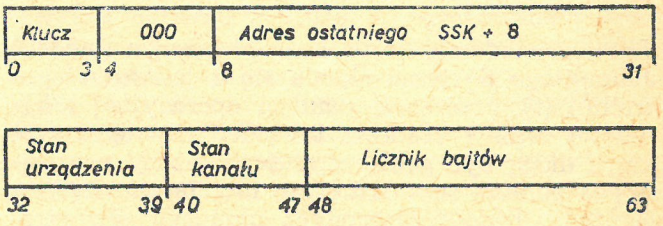
o nazwie „przerwanie sterowane programowo" (PSP). Cecha PSP powoduje wygenerowanie sygnału przerwania programu, gdy kanał pobiera kolejne słowo SSK w czasie wykonywania łańcucha danych lub łańcucha komend. W ten sposób program jest zawiadamiany o tym, że kolejne pole danych zostało zapisane (odczytane) w pamięci operacyjnej i może na przykład przystąpić do przetwarzania tych danych nie czekając na koniec przesyłania całej sekwencji pól danych.

4.3. Zakończenie przesyłania danych

Wszystkie operacje wejścia-wyjścia, po ich zapoczątkowaniu przez procesor, przebiegają wyłącznie pod kontrolą kanału i jednostki sterującej urządzenia. W ten sposób procesor jest wolny i może kontynuować wykonanie programu. Proces przesyłania danych zostaje normalnie zakończony po przesłaniu określonej sekwencji danych. Koniec przesyłania danych jest sygnalizowany technicznie przez wygenerowanie warunku, „kanał zakończył". Warunek ten powstaje zarówno przy prawidłowym zakończeniu operacji (np. wyzerowanie się licznika bajtów lub gdy urządzenie wykryło znak końca), jak i przy powstaniu błędu w trakcie wykonywania operacji (np. błąd ochrony lub błąd łańcucha). Jeśli kanał wygeneruje warunek „kanał zakończył", urządzenie zostaje o tym również powiadomione. Często zakończenie pracy kanału — który może być wolny dla innych urządzeń — nie musi być równoznaczne z zakończeniem pracy urządzenia (np. drukarki wierszowej mającej własny bufor). Dlatego wyróżnia się również warunek „urządzenie zakończyło pracę", a nawet odrębny warunek „jednostka sterująca zakończyła pracę".

Jeśli kanał lub urządzenie zakończy zadaną operację, to zgłasza przerwanie programu, które normalnie zostaje przyjęte, jeśli odpowiednia maska systemu zezwala na przerwania z danego kanału. Po przyjęciu tego zgłoszenia kanał czyta tzw. bajt stanu urządzenia i zapisuje go w stałym polu pamięci operacyjnej, łącznie ze stanem kanału, kluczem ochrony pamięci, adresem ostatniego użytego słowa SSK powiększonym o 8 oraz z pozostałością licznika bajtów z ostatnio użytego SSK. Wszystkie powyższe informacje zapisane w polu 64—71 tworzą tzw. słowo stanu kanału (SSTK), które to słowo przedstawia rysunek 4.7. Słowo SSTK zostaje stworzone i zapamiętane nie tylko w wyniku przerwania typu we-wy, lecz także może zostać zapamiętane w wyniku wykonania instrukcji TESTUJ WE-WY, STOP WE-WY i START WE-WY.

Program może — w wyniku analizy stanu pola pamięci 64—71 — zidentyfikować dokładną przy-



Rys. 4.7. Struktura słowa stanu kanału

czynę przerwań lub dowiedzieć się o stanie urządzenia zewnętrznego. Natomiast numer urządzenia i kanału, które spowodowały dane przerwanie typu we-wy, znajduje się w starym słowie stanu programu (SSP), tj. w polu 56—63, gdzie zapamiętuje go normalny proces przerwania programu.

W części „stan urządzenia" słowa SSTK rejestruje się między innymi takie informacje, jak: jednostka sterująca zakończyła operację (bit 34), urządzenie — lub jego jednostka sterująca — jest zajęte wykonywaniem poprzedniej operacji (bit 35), kanał zakończył operację i jest wolny dla innej operacji we-wy (bit 36), urządzenie zakończyło operację i jest wolne (bit 37), powstał błąd lub jakiś inny szczególny warunek, który należy przeczytać komendą BADAJ (bit 38). Natomiast bajt „stan kanału" może zawierać informacje o błędach wykrytych w kanale — np. niewłaściwy adres, licznik lub kod w SSK, błąd ochrony, błędne dane itp. — lub o tym, że zaszło programowo sterowane przerwanie, które jest zwykłym meldunkiem o przebiegu procesu przesyłania sekwencji pól danych.

5. Charakterystyka jednostek centralnych JS EMC

Jak już wiadomo, każda spośród maszyn (modeli) Jednolitego Systemu składa się z jednostki centralnej (tj. procesora, pamięci operacyjnej i kanałów) oraz z zestawu urządzeń zewnętrznych, które mogą być podłączane do jednostki centralnej za pomocą standardowego interface'u wejścia-wyjścia. Ilość podłączonych urządzeń zewnętrznych zależy od ilości i typów kanałów danej jednostki centralnej.

Wszystkie modele Jednolitego Systemu mają dla urządzeń zewnętrznych wspólną nomenklaturę, z której dobiera się potrzebną dla danego modelu konfigurację. Dzięki temu użytkownik może tworzyć dowolną konfigurację użytkową, dostosowaną do jego aktualnych potrzeb. Systemy operacyjne, które otrzymuje użytkownik są również „krojone" na miarę zainstalowanej konfiguracji sprzętu.

5.1. Model JS-1010

Maszyna cyfrowa JS-1010, opracowana i produkowana w Węgierskiej Republice Ludowej, stanowi najmniejszy model Jednolitego Systemu. Pod względem charakterystyki funkcjonalnej przypomina ona nowoczesne minikomputery i przeznaczona jest zarówno do wykorzystania jako samodzielny system, jak i do pracy w charakterze komputera — satelity (tj. jako punkt abonencki w systemach teleprzetwarzania).

Typowa konfiguracja tej maszyny zawiera następujący zestaw urządzeń: procesor, czytnik i dziurkarka taśmy papierowej, elektryczna maszyna pisząca oraz mała pamięć dyskowa o pojemności 864 Kbajtów. Maszyna JS-1010 posiada własny system operacyjny, tzw. OS-10, który oczywiście ma ograniczone funkcje i możliwości w porównaniu z systemem operacyjnym OS-JS (przeznaczonym dla maszyn R-20, R-30, R-32, R-40 i R-50). Maszyna JS-1010 ma również odmienny system instrukcji i formatów informacji od pozostałych EMC Jednolitego Systemu. Wymagana moc dla typowego zestawu maszyny wynosi 2 kV·A.

Jednostka Centralna JS-2010. Jednostka ta stanowi centralną część maszyny i zawiera pamięć, arytmometr, środki sterowania i wymiany informacji oraz środki współpracy z urządzeniami wejścia-wyjścia. Pamięć operacyjna, zbudowana na rdzeniach ferrytowych, ma pojemność od 8 do 64 Kbajtów, z możliwością rozszerzenia jej blokami po 8 Kbajtów; cykl pamięci wynosi 0,8÷1 μs, zaś czas dostępu wynosi 0,4÷0,5 μs. Szerokość czytanej lub pisanej informacji — 2 bajty.

Sterowanie procesorem jest mikroprogramowe, a pamięć mikroprogramów zbudowana jest na półprzewodnikowych układach scalonych. Pojemność pamięci mikroprogramów wynosi od 512 do 1536 słów o długości 16 bitów i czasie dostępu 60 μs. Poza tym istnieje druga pamięć, zwana pamięcią mikroinstrukcji, która składa się z 96 słów 48-bitowych, także o cyklu 60 μs. Ostatnia pamięć zawiera słowa sterujące (wykonawcze), które otwierają lub zamykają odpowiednie drogi przepływu informacji w arytmometrze.

Tabela 5.1

Czas wykonania operacji w procesorach JS EMC

Model JS — / Operacja	2010	2021	2020	2030	2032	2040	2050
Dodawanie-odejmowanie stałoprzecinkowe	2,1 — 3,1	22	30	7 — 11	2,2 — 2,8	1,4 — 2,0	0,65
Dodawanie-odejmowanie zmiennoprzecinkowe	—	—	70	10 — 15	5,6 — 5,8	2,5 — 3,6	1,4
Mnożenie stałoprzecinkowe	40	173	350	32 — 37	16 — 17	7,2 — 8,2	2,0
Mnożenie zmiennoprzecinkowe	—	—	480	27 — 33	14 — 15	6,5 — 7,1	2,0
Dzielenie stałoprzecinkowe	45	—	400	88 — 93	17 — 18	16,2-17,3	8,3
Dzielenie zmiennoprzecinkowe	—	—	400	47 — 51	15 — 16	10,4-20,3	7,2
Operacje krótkie	2,1 — 3,1	15	30	10	2,2	1,1 — 1,9	0,65

Uwaga. Czas wykonania operacji zostały podane w mikrosekundach (μs)

Arytmometr wykonuje operacje na półsłowach, tj. na bajtach. Czas wykonania poszczególnych operacji arytmetycznych i logicznych przedstawiony jest w tabeli 5.1 [9]. Podstawowy cykl procesora wynosi 300 ns. Rejestry robocze procesora stanowi szybkodostępna pamięć półprzewodnikowa o pojemności 64×2 bajtów i o cyklu 60 ns. Szyna wymiany informacji pomiędzy zespołami funkcjonalnymi zbudowana jest na tzw. zasadzie MINI-BUS. Do szyny tej podłącza się, między innymi, adaptery standardowego interface'u wejścia-wyjścia. Maksymalna szybkość przesyłania danych w trybie multiplekserowym (ilość podkanałów do 127) wynosi 30 Kbajtów na sekundę, zaś w trybie selektorowym — 140 Kbajtów na sekundę.

5.2. JS-1021

Maszyna cyfrowa JS-1021 jest jednym z małych modeli Jednolitego Systemu. Została ona opracowana w Czechosłowackiej Republice Ludowej; tam też jest produkowana. Maszyna JS-1021 przeznaczona jest do rozwiązywania zadań ekonomicznych i zarządzania (w niższych ogniwach Zintegrowanego Systemu Zarządzania). Ponadto może być zastosowana we wstępnej obróbce danych przeznaczonych dla bardziej wydajnych maszyn.

W skład typowego zestawu maszyny wchodzą:
— procesor wraz z pamięcią operacyjną i kanałami,
— dwie jednostki wymiennej pamięci dyskowej o pojemności po 7,25 Mb (szyfr JS-5053),
— czytnik kart dziurkowanych (JS-6016),
— drukarka wierszowa (JS-7034),
— elektryczna maszyna pisząca (JS-7071).

Lista instrukcji modelu JS-1021 nie jest w pełni zgodna z listą instrukcji JS EMC. Dlatego zgodność programowa tego modelu z pozostałymi modelami JS EMC została osiągnięta tylko na poziomie języków programowania, tj. języka ASSEMBLER, języka RPG i języków algorytmicznych. Lista instrukcji modelu JS-1021 zawiera 65 instrukcji, zatem jest ona podzbiorem pełnej listy JS EMC. Przyjęte formaty instrukcji i danych są w pełni zgodne z formatami EMC Jednolitego Systemu.

Maszyna JS-1021 ma również własny system operacyjny (MOS-JS), odmienny od standardowych systemów operacyjnych JS EMC (tj. DOS-JS i OS-JS).

Rozmieszczenie typowego zestawu maszyny wymaga 50 m² powierzchni; pobór mocy — 23 kV·A.

Jednostka Centralna JS-2021. Jest to jednostka stanowiąca część centralną maszyny, a składająca się z bloków wykonania operacji arytmetycznych i logicznych, bloku sterowania, pamięci operacyjnej oraz z multiplekserowego i selektorowych kanałów wejścia-wyjścia.

Pamięć operacyjna zbudowana jest na rdzeniach ferrytowych, a pojemność jej wynosi 16 Kbajtów, z możliwością rozszerzenia jej do 32 i 64 Kbajtów. Cykl pamięci wynosi 2 μs, zaś czas dostępu 1 μs. Szerokość dostępu — 1 bajt. Ochrona pamięci nie jest przewidziana w tej maszynie.

Pamięć operacyjna posiada środki, tzw. blok bezpośredniej łączności, zapewniające bezpośrednią wymianę danych pomiędzy pamięciami operacyjnymi dwóch połączonych maszyn cyfrowych JS-1021. Wymiana ta odbywa się za pomocą mikroprogramowego sterowania, z prędkością 500 Kb/s.

Arytmometr modelu JS-1021 ma strukturę jednobajtową, tzn. wykonuje działania bajt po bajcie, co powoduje określone wydłużenie czasu wykonania operacji na wielkościach jednosłowowych (patrz tabela 5.1). Podstawowy cykl procesora wynosi 300 ns. Rejestry robocze procesora i kanałów tworzą oddzielną pamięć buforową o pojemności 384 bajtów i cyklu 250 ns. Mikroprogramowane sterowanie maszyny jest oparte na pamięci stałej o pojemności 3072 słów i cyklu 250 ns. Długość słowa mikroprogramu wynosi 72 bity.

Kanały wejścia-wyjścia modelu JS-1021. Maszyna JS-1021 może mieć dwa kanały selektorowe; do każdego kanału podłącza się do 160 urządzeń zewnętrznych poprzez 10 jednostek sterujących (tymi urządzeniami). Szybkość przesyłania danych w kanale selektorowym wynosi 250 Kb/s. Natomiast do kanału multiplekserowego można podłączyć do 16 podkanałów oraz do 136 wolnych urządzeń zewnętrznych poprzez 10 jednostek sterujących. Szybkość przesyłania danych w kanale multiplekserowym wynosi 35 Kb/s dla trybu multiplekserowego i 220 Kb/s dla trybu selektorowego.

5.3. Model JS-1020

Maszyna cyfrowa JS-1020 — opracowana i produkowana w Związku Radzieckim oraz w Ludowej Republice Bułgarii — jest również małą maszyną Jednolitego Systemu, przeznaczoną głównie do zastosowań EPD. Może być również zastosowana w prostych obliczeniach naukowo-technicznych.

W odróżnieniu od wcześniej omówionych modeli, tj. JS-1010 i JS-1021, maszyna JS-1020 jest w pełni zgodna programowo z pozostałymi (starszymi) EMC Jednolitego Systemu na poziomie języka maszynowego, czyli na poziomie instrukcji maszynowych. W modelu JS-1020 wbudowano pełną listę instrukcji, tj. zestaw uniwersalny.

Podstawowym systemem operacyjnym modelu JS-1020 jest systemem DOS-JS. Wymaga on pamięci operacyjnej o minimalnej pojemności 32 Kbajtów i co najmniej jednej jednostki pamięci dyskowej (np. o pojemności 7,25 Mbajtów). System DOS-JS ma ograniczone możliwości funkcjonalne — np. organizuje pracę wieloprogramową tylko dla trzech programów umieszczonych w stałych obszarach — i przeznaczony jest dla małych EMC Jednolitego Systemu z pamięcią operacyjną do 256 Kbajtów.

W modelu JS-1020 może również funkcjonować bardziej rozbudowany w swoich funkcjach system operacyjny OS-JS. System ten potrzebuje pamięci operacyjnej o minimalnej pojemności 256 Kbajtów i przeznaczony jest głównie dla starszych modeli JS EMC.

Jednostka Centralna JS-2020. Jednostka ta zawiera — w formie odrębnych konstrukcyjnie urządzej — centralną jednostkę przetwarzania z dwoma kanałami selektorowymi i jednym kanałem multiplekserowym (JS-2420), pamięć operacyjną (JS-3220) oraz system zasilania (JS-0820). Wyżej wymienione urządzenia są lokalizowane w osobnych typowych szafach.

Struktura logiczna arytmometru modelu JS-1020 jest jednobajtowa, co pozwala uzyskać określoną prostotę rozwiązań. Niemniej struktura taka powoduje wydłużenie czasu wykonania operacji na argumentach jednosłowowych. Cykl wewnętrzny procesora wynosi 200 ns.

Sterowanie jest mikroprogramowe, oparte na pamięci stałej typu transformatorowego. Pamięć stała ma pojemność 8096 słów 64-bitowych i cykl odczytu 1 μs.

Pamięć operacyjna jest typu ferrytowego, system $2\frac{1}{2}$D, wykonana w trzech funkcjonalnych wariantach: pamięć lokalna, pamięć multipleksera i podstawowa pamięć operacyjna.

Pamięć lokalna składa się z 256 bajtów; przechowuje się w niej stany rejestrów ogólnych i roboczych procesora.

Pamięć multipleksera ma pojemność 768—1536 bajtów i przechowuje informacje sterujące przesyłaniem danych przez miltiplekser. Podstawowa pamięć operacyjna o pojemności od 64 do 256 Kbajtów instalowana jest w blokach po 64 Kbajtów. Szerokość szyny czytanych i pisanych informacji wynosi 2 bajty. W maszynie JS-1020 wbudowano pełną ochronę pamięci, tzn. zarówno przy zapisie jak i odczycie informacji z pamięci.

Kanały wejścia-wyjścia. Do modelu JS-1020 można podłączyć jeden lub dwa kanały selektorowe i jeden kanał multiplekserowy. Jeśli pracują oba kanały selektorowe, to pierwszy ma większą szybkość przesyłania i dlatego podłącza się do pierwszego kanału pamięci dyskowe. Natomiast pamięci taśmowe podłącza się do drugiego kanału selektorowego. Szybkość kanału selektorowego wynosi do 200 Kb/s.

Kanał multiplekserowy może sterować pracą od 48 do 112 podkanałów — w zależności od pojemności pamięci multipleksera — i przesyłać dane z szybkością do 16 Kb/s w trybie multiplekserowym, oraz z szybkością 100 Kb/s w trybie selektorowym. Sterowanie kanałów jest mikroprogramowe i znajduje się w tej samej pamięci stałej, co mikroprogramy procesora.

5.4. Model JS-1030

Maszyna JS-1030 została opracowana w ZSRR i produkowana jest w tym kraju. Jeden egzemplarz tej maszyny został również wykonany w WZE MERA-ELWRO w roku 1972. Początkowy zamiar produkcji maszyny JS-1030 w Zakładach MERA-ELWRO został zaniechany i przystąpiono do opracowania własnej maszyny o oznaczeniu R-32, w której zastosowano krajowe rozwiązania konstrukcyjno-technologiczne oraz układy scalone średniej skali integracji (TTL). Natomiast w maszynie JS-1030 — której projekt techniczny powstał w latach 1969—71 — zastosowano układy scalone małej skali integracji (również TTL).

Model JS-1030 jest maszyną średniej klasy wśród EMC Jednolitego Systemu, przeznaczoną zarówno do obliczeń naukowo-technicznych jak i do przetwarzania danych.

Maszyna JS-1030 jest w pełni zgodna programowo z pozostałymi maszynami Jednolitego Systemu, tj. z maszynami R-20, R-32, R-40 i R-50. W maszynie JS-1030 wbudowano uniwersalny zestaw listy instrukcji maszynowych.

Typowy zestaw maszyny JS-1030 składa się z procesora, pamięci operacyjnej, trzech kanałów, dwóch pamięci dyskowych po 7,25 Mb (JS-5052), czterech pamięci taśmowych (JS-5010), z czytnika i dziurkarki taśmy papierowej, z czytnika i dziurkarki kart, z drukarki wierszowej i elektrycznej maszyny piszącej.

Maszyna JS-1030 ma sterowanie mikroprogramowe przechowywane w pamięci stałej typu transformatorowego o cyklu pracy 600 ns i pojemności 4096 słów 72-bitowych.

W maszynie tej wprowadzono małą pamięć lokalną o pojemności 64 słów 4-bajtowych i o cyklu 600 ns — do przechowywania stanu rejestrów ogólnych i rejestrów zmiennoprzecinkowych. Pamięć lokalna zbudowana jest w oparciu o technikę pręcików magnetycznych.

Pamięć operacyjna maszyny JS-1030 wynosi w typowym zestawie 256 Kbajtów. Pamięć ta składa się z dwóch bloków po 128 Kbajtów rozmieszczonych w jednej typowej szafie. Natomiast zasilanie tej pamięci zajmuje osobną szafę. Pamięć operacyjną można maksymalnie rozbudować do 512 Kbajtów. Cykl pamięci operacyjnej wynosi 1,25 μs, zaś czas dostępu 0,75 μs. Szerokość czytanej lub pisanej do pamięci informacji wynosi 4 bajty.

Struktura procesora JS-2030 jest również 4-bajtowa.

Kanały wejścia-wyjścia (JS-4430). W maszynie JS-1030 przyjęto układową realizację kanałów wejścia-wyjścia. Oznacza to, że kanały zdolne są do przesyłania danych do lub z pamięci operacyjnej, bez pośrednictwa procesora. Taki system przesyłania danych pozwala na pełną niezależność pracy kanałów i procesora. Niemniej system ten jest kosztowny w realizacji (np. w maszynie JS-1030 sprzęt kanałów zajmuje jedną typową szafę). W modelu JS-1030 można wbudować jeden kanał multiplekserowy i do trzech kanałów selektorowych (pracujących z szybkością około 800 Kb/s). Szybkość kanału multiplekserowego wynosi 40 i 300 Kb/s — odpowiednio w trybie multiplekserowym i w trybie selektorowym. Słowa sterujące kanału multiplekserowego przechowuje się w odrębnej pamięci o pojemności 1024 słów 32-bitowych.

5.5. Model JS-1040

Maszyna JS-1040 została opracowana i produkowana w Niemieckiej Republice Demokratycznej. Jest to jedna z wysokowydajnych EMC Jednolitego Systemu, przeznaczona zarówno do złożonych zastosowań EPD jak i do obliczeń naukowo-technicznych, gdzie wymagana jest duża szybkość przetwarzania i duża pojemność pamięci.

Maszyna JS-1040 jest w pełni programowo zgodna z pozostałymi EMC Jednolitego Systemu (z wyjątkiem maszyn JS-1010 i JS-1021) na poziomie języka maszynowego. W modelu JS-1040 wbudowano pełny uniwersalny zestaw instrukcji.

Maszyna jest dostarczana użytkownikowi wraz z systemem operacyjnym DOS-JS lub OS-JS, zależnie od typu zastosowań i wymaganej efektywności procesu przetwarzania.

Typowy zestaw maszyny — w tym 6 jednostek pamięci dyskowej typu JS-5055 i 8 jednostek pamięci taśmowej typu JS-5016 — wymaga 150 m^2 powierzchni i pobiera moc 65 kV·A.

Procesor (JS-2040) maszyny składa się z dwóch arytmometrów, tj. z 8-bajtowego arytmometru przeznaczonego do wykonywania działań na argumentach określonych przez instrukcje oraz z 1-bajtowego arytmometru przeznaczonego głównie do pobierania i przygotowywania instrukcji do wykonania. Cykl pracy procesora wynosi 200 ns. Sterowanie wykonaniem podstawowej funkcji instrukcji jest mikroprogramowe, zaś przygotowanie instrukcji do wykonania — układowe. Wybór kolejnej instrukcji oraz wykonanie instrukcji poprzedniej zachodzi częściowo równolegle. Pamięć mikroprogramów ma 3072 słów 130-bitowych i cykl czytania 450 ns.

Pamięć operacyjna maszyny JS-1040 może mieć

pojemność 256, 512 lub 1024 Kbajtów. Pamięć ma cykl 1,35 μs i czas dostępu 0,45 μs, zaś szerokość jej szyny informacyjnej wynosi 8 bajtów. Każdy blok pamięci operacyjnej jest powiększony o dodatkową pamięć (oprócz podstawowej pamięci adresowanej przez program) do przechowywania słów sterujących kanału multiplekserowego i kluczy ochrony. Dodatkowa pamięć nie jest dostępna dla programisty.

Sterowanie pamięcią operacyjną jest tak zorganizowane, że pozwala na jednoczesną pracę bloków po 256 Kbajtów.

Kanały wejścia-wyjścia. Do maszyny JS-1040 można podłączyć jeden kanał multiplekserowy i do 6 kanałów selektorowych. Kanał multiplekserowy może mieć różną ilość podkanałów w zależności od pojemności pamięci operacyjnej, a mianowicie: 128 podkanałów dla pojemności 256 Kbajtów i 256 podkanałów dla pojemności 512 lub 1024 Kbajtów. Szybkość pracy multipleksera wynosi 110 i 670 Kb/s — odpowiednio dla trybu multiplekserowego i trybu selektorowego. Do kanału multiplekserowego można podłączyć do 10 jednostek sterujących.

Pierwszy kanał selektorowy ma największą szybkość przesyłania danych; wynosi ona 1300 Kb/s. Kanały selektorowe drugi i trzeci mają szybkość 500 Kb/s, zaś czwarty, piąty i szósty — 300 Kb/s.

5.6. Model JS-1050

Maszyna JS-1050 jest najszybszą maszyną Jednolitego Systemu RIAD-1. Przeznaczona jest do rozwiązywania złożonych problemów naukowo-technicznych, zarządzania i innych.

W maszynie JS-1050 wbudowano wszystkie funkcje i mechanizmy występujące w JS EMC RIAD-1, do uzyskania maksymalnej szybkości przetwarzania i przesyłania danych. W celu zwiększenia szybkości przesyłania danych kanały są w pełni autonomiczne, zarówno w sensie logicznym jak i pod względem konstrukcyjnym.

Typowy zestaw tej maszyny wymaga pomieszczenia o powierzchni 230 m² i pobiera moc 70 kV·A.

Jednostka centralna JS-2050. Jednostka centralna wykonuje wszystkie operacje artymetyczne i logiczne występujące w programach oraz organizuje współpracę z pamięcią operacyjną i kanałami. Podstawowy arytmometr tej maszyny ma strukturę 8-bajtową i cykl 160 ns. Wykonuje się w nim wszystkie operacje, z wyjątkiem operacji dziesiętnych i działań na polach o zmiennej długości. Te ostatnie operacje są wykonywane w oddzielnym 1-bajtowym arytmometrze.

Sterowanie centralnej jednostki przetwarzania jest w pełni zrealizowane środkami układowymi, co pozwala osiągać dużą szybkość wykonywania operacji. Na szybkość obliczeń wpływa również przyjęty system równoległego wykonywania bieżącej instrukcji i przygotowanie do wykonania kilku następnych instrukcji. W tym celu wprowadzono cztery bufory jednosłowowe do przechowania przygotowanych do wykonania instrukcji. W buforach tych można na przykład przygotować do 8 instrukcji typu RR lub do 3 instrukcji typu SS.

Rejestry ogólne i robocze maszyny są zrealizowane na układach scalonych w formie bloku o pojemności 128 bajtów, skąd jednocześnie można czytać jeden argument i pisać dwa argumenty.

W maszynie JS-1050 wbudowano system ochrony pamięci zarówno przy zapisie, jak i przy odczycie informacji. Sprawdzanie zgodności kluczy odbywa się równolegle z procesem odwoływań do pamięci operacyjnej. Klucze ochrony przechowuje się w odrębnej małej pamięci o pojemności 512 bajtów i o cyklu 320 ns. Taka ilość kluczy (tj. 512) jest wystarczająca dla pamięci operacyjnej o pojemności 256, 512 i 1024 Kbajtów.

Czasomierz interwałowy maszyny JS-1050 zmienia swój stan bardzo często, tj. co 13,3 μs. Zmiana ta, tzw. zmiana szybka, odbywa się środkami sprzętowymi. Natomiast zmiana wolna, która powoduje zmianę stanu odpowiedniej komórki pamięci operacyjnej, odbywa się raz na 20 ms.

Z powyższego opisu widać, że procesor JS-2050 jest najbardziej złożonym strukturalnie i efektywnym w działaniu urządzeniem JS EMC (procesor wraz ze swoim zasilaniem zajmuje 5 standardowych szaf).

Pamięć operacyjna JS-3205. Model JS-1050 może mieć pamięć operacyjną o pojemności od 256 do 1024 Kbajtów, instalowaną w osobnych blokach logicznych po 256 Kbajtów. Taki jeden blok (mający szyfr JS-3205) łącznie ze swoim systemem zasilania zajmuje dwie standardowe szafy. Zatem pojemność 1024 Kbajtów zajmuje 8 szaf. Szerokość szyny informacyjnej PAO — PROCESOR wynosi 8 bajtów. Cykl pamięci operacyjnej (ferrytowej) wynosi 1,25 μs, zaś czas dostępu — 0,8 μs.

W celu zwiększenia szybkości współpracy z pamięcią operacyjną przyjęto przeplot adresów drugiego stopnia, co pozwala na jednoczesną pracę wszystkich czterech bloków logicznych. Dzięki temu średni czas odwoływań do PAO skraca się w istotny sposób.

Kanały wejścia-wyjścia. Kanały maszyny JS-1050 mają strukturę logiczną w pełni samodzielną, tzn. organizują przesyłanie danych do lub z PAO bez pośrednictwa procesora. Również konstrukcyjnie kanały są niezależnie, tzn. znajdują się w osobnych szafach (dwie szafy standardowe zajmuje kanał multiplekserowy ze swoim zasilaniem, a 4 szafy zajmuje 6 kanałów selektorowych ze swoim zasilaniem).

Kanał multipleksera (JS-4012) może mieć do 196 podkanałów, z których 192 są normalnymi niedzielonymi podkanałami multipleksera, zaś 4 są dzielonymi podkanałami typu selektorowego.

W ten sposób kanał multipleksera ma 5 pełnych i niezależnych szyn interface'u wejścia-wyjścia. Kanał ten oczywiście pracuje zarówno we własnym trybie, jak i w trybie selektorowym i może osiągnąć łączną szybkość przesyłania 110 Kb/s dla wszystkich 192 podkanałów. Podkanały selektorowe pracują tylko w trybie selektorowym i każdy może osiągnąć szybkość do 180 Kb/s. Niemniej sumaryczna szybkość wszystkich podkanałów w trybie selektorowym nie może przekroczyć 670 Kb/s. W celu przechowywania słów sterujących podkanałów multipleksera, słów stanów i bieżących danych wprowadzono specjalną pamięć o pojemności 4 Kbajtów.

Kanał selektorowy (JS-4035) może osiągnąć szybkość przesyłania do 1300 Kb/s. Przesyłanie

danych pomiędzy pamięcią operacyjną a kanałem selektorowym odbywa się paczkami po 8 bajtów (w czasie jednego odwołania do PAO). Do każdego kanału można dołączyć do 8 jednostek sterujących (w których można wyróżnić do 256 urządzeń zewnętrznych).

Każdy kanał selektorowy może współpracować z adapterem kanał-kanał (JS-4060), za pośrednictwem którego kanał ten może komunikować się z innym kanałem selektorowym (drugiej lub tej samej EMC).

W minimalnym zestawie maszyny JS-1050 instaluje się 3 kanały selektorowe, które mieszczą się w jednej standardowej szafie. Natomiast zasilanie tych trzech kanałów mieści się w dodatkowej szafie.

5.7. Model JS-1032

5.7.1. Wstęp

W pierwszym okresie opracowywania i budowy środków technicznych JS EMC, tj. w latach 1969—72, Polska opracowała i wdrożyła do produkcji różne urządzenia zewnętrzne JS EMC (w tym drukarki wierszowe JS-7033, pamięci taśmowe JS-5019 i inne). Natomiast udział Polski w zakresie jednostek centralnych JS EMC miał charakter rozpoznawczy i szkoleniowy. Do opracowania własnej konstrukcji jednostki centralnej przystąpiono w II połowie 1972 r., a więc z pewnym opóźnieniem w stosunku do pozostałych krajów uczestniczących w JS EMC. W związku z tym, że w 1972 roku istniały lepsze możliwości w bazie elementowej oraz bogatsze doświadczenia w zakresie technologii wytwarzania zespołów komputerowych, można było przyjąć bardziej perspektywiczne postępowanie przy opracowywaniu i budowie polskiej maszyny cyfrowej JS EMC, tj. maszyny R-32. Przyjęto między innymi generalną zasadę stosowania bazy elementowej oraz rozwiązań konstrukcyjnych i technologicznych wyprzedzających możliwości przemysłu krajowego o 2—3 lata, dzięki czemu w momencie rozpoczęcia produkcji seryjnej maszyna ta będzie technicznie nowoczesna. W szczególności przyjęto następujące rozwiązania w konstrukcji maszyny R-32 [9]:

— strukturę modułową,
— szerokie zastosowanie mikroprogramowania,
— elementy scalone standardowej i średniej skali integracji,
— wielowarstwowe płytki pakietów o dużym formacie (do 12 warstw),
— wielowarstwowe platery,
— bardzo szybką pamięć stałą o dużej pojemności,
— szybką i pojemną ferrytową pamięć operacyjną o nowoczesnej konstrukcji (tzw. planarnej),
— płaskie kable dla połączeń między blokami maszyny,
— bezszafową, aluminiową konstrukcją mechaniczną.

W wyniku realizacji powyższych założeń maszynę R-32 charakteryzują następujące cechy:
— łatwość rozszerzania pojemności pamięci operacyjnej,
— bardzo wysoki współczynnik wydajności/koszt,
— wysoka niezawodność,
— małe rozmiary i ciężar,
— bardzo mały pobór mocy.

5.7.2. Jednostka centralna maszyny R-32

EMC R-32 składa się z trzech podstawowych bloków funkcjonalnych:
— procesora,
— kanałów,
— pamięci operacyjnej (PAO).

Procesor zawiera arytmometr, pamięć lokalną, pamięci mikroprogramów i układy sterowania. Procesor wykonuje wszelkie operacje arytmetyczno-logiczne, steruje systemem kontroli i diagnostyki, kontaktuje się z pamięcią operacyjną, a także wykonuje wiele funkcji dla koordynatora kanałów i dla poszczególnych kanałów.

Koordynator kanałów organizuje pracę czterech kanałów, tj. jednego kanału multiplekserowego i trzech selektorowych.

Kanały nie mają bezpośredniego dostępu do pamięci operacyjnej, lecz przesyłają dane do lub z PAO za pośrednictwem koordynatora kanałów i procesora. Rozwiązanie to pozwoliło uzyskać bardzo prostą strukturę kanałów.

Pamięć operacyjna składa się z niezależnych bloków o pojemności 128 Kbajtów, wyposażonych w adaptery i podłączonych do wspólnej szyny PROCESOR — PAO.

Adapter zawiera m.in. układy nadajników i odbiorników tej szyny oraz pamięć kluczy ochrony dla danego bloku PAO.

Pojemność pamięci operacyjnej EMC R-32 może wynosić od 128 do 1024 Kbajtów.

Arytmometr maszyny R-32 zawiera dwa typy sumatorów: słowowy i bajtowy, różne rejestry robocze i adresowe, układy kontroli i diagnostyki i inne. Sumator słowowy wykonuje operacje arytmetyczne (binarne i dziesiętne) na argumentach 32-bitowych. Natomiast sumator bajtowy służy do realizacji wszelkich działań logicznych na bajtach, w tym operacji na polach o zmiennej długości.

W arytmometrze wbudowano specjalny układ, zwany komutatorem diagnostycznym, który ma zdolność odczytywania stanu wszystkich rejestrów i wskaźników procesora. Dzięki temu możliwe jest zapamiętanie stanu procesora w pamięci operacyjnej, czyli wygenerowanie tzw. erratografii procesora. Analiza erratografii pozwala na szybkie zlokalizowanie uszkodzenia.

Wbudowane układy kontroli umożliwiają dynamiczną kontrolę poprawności pracy określonych zespołów procesora. Oprócz układowej kontroli wprowadzono również tzw. kontrolę okresową, która sprawdza poprawność pracy ponad 90% sprzętu.

Pamięć lokalna zbudowana jest na układach scalonych i składa się z 64 słów 32-bitowych. Zawiera ona 16 rejestrów ogólnych, 4 rejestry zmiennoprzecinkowe i 40 rejestrów roboczych procesora i kanałów.

Pamięć mikroprogramów jest pamięcią stałą typu transformatorowego i ma cykl równy 300 ns. Zawiera ona 2816 słów 86-bitowych, w których zakodowane jest sterowanie procesora i kanałów. Zastosowanie pamięci mikroprogramów o tak du-

żej pojemności pozwoliło na znaczne uproszczenie i zmniejszenie logiki układowej maszyny, a także na wyposażenie maszyny w pewne nowe — w stosunku do innych maszyn JS EMC — własności, jak:
— kontrola okresowa,
— automatyczne zapamiętanie stanu maszyny w PAO przy wyłączeniu zasilania i odtwarzanie tego stanu po włączeniu zasilania,
— specjalne testy techniczne procesora i pamięci.

Procesor pracuje w jednym z dwóch typów pracy: tryb CJP i tryb WE-WY. W trybie CJP procesor wykonuje mikroprogramy instrukcji, natomiast w trybie WE-WY realizowane są mikroprogramy obsługi kanałów. Przejście z trybu CJP do WE-WY odbywa się w sposób wymuszony, tj. na żądanie koordynatora kanałów, i może nastąpić w trakcie wykonywania dowolnej instrukcji. W takim przypadku mikroprogram bieżącej instrukcji zostaje zawieszony, a adres następnej mikroinstrukcji przepisuje się do specjalnego rejestru buforowego i procesor przechodzi do wykonania jednego z mikroprogramów obsługi kanałów. Po skończeniu operacji WE-WY i przy braku następnych zgłoszeń z koordynatora kanałów procesor wznawia zawieszony mikroprogram instrukcji. W trybie WE-WY określone części mikroinstrukcji są dekodowane odmiennie niż w trybie CJP. W ten sposób uzyskuje się dodatkowy zbiór mikrooperacji służących wyłącznie do sterowania pracą kanałów.

Sterowanie centralne rozwiązane zostało w taki sposób, aby cykl pracy procesora i kanałów był ograniczony tylko sumą propagacji sygnałów informacyjnych, natomiast samo sterowanie nie wnosiło dodatkowych opóźnień. Uzyskano to dzięki zastosowaniu m. in. rejestru buforowego dla mikroinstrukcji pobieranych z pamięci stałej. W ten sposób procesor i kanały o czasie cyklu 300 ns współpracują z pamięcią stałą, wykorzystując jej maksymalną szybkość działania (3,3 MHz).

5.7.3. Kanały wejścia-wyjścia

Przyjęty koordynator kanałów współpracuje z jednej strony z układami sterowania procesora i pamięcią mikroprogramów, z drugiej — steruje pracą trzech kanałów selektorowych i jednego multiplekserowego. Wszelkie zgłoszenia z kanałów przyjmuje tzw. układ priorytetu, który decyduje o kolejności przyłączenia kanałów do procesora w zależności od numeru kanału i rodzaju zgłoszenia. W danym momencie procesor może obsługiwać tylko jeden kanał.

W maszynie R-32 można wmontować jeden kanał multiplekserowy i do trzech kanałów selektorowych.

Szybkość pracy kanału multiplekserowego wynosi 110 i 250 Kb/s — odpowiednio dla trybu multiplekserowego i trybu selektorowego. Jako pamięć roboczą dla słów sterujących kanału multiplekserowego wykorzystuje się niedostępny programowo obszar PAO równy 2 lub 4 Kbajty, co pozwala mieć odpowiednio 128 lub 256 podkanałów. Obszar ten znajduje się zawsze na końcu zainstalowanej PAO. Takie rozwiązanie pamięci multipleksera jest bardzo proste i tanie w stosunku do rozwiązań opartych o niezależny blok pamięci.

Operacje multipleksera są wykonywane przez odpowiednie mikroprogramy, przechowywane w tej samej pamięci stałej co mikroprogramy procesora.

Kanały selektorowe pracują zawsze w trybie selektorowym i umożliwiają bardzo szybkie transmisje danych. Kanały te mają wbudowane układowe rejestry sterujące operacjami WE-WY, a także korzystają z mikroprogamów zawartych w pamięci stałej. Szerokie zastosowanie sterowania mikroprogramowego w kanałach zmniejszyło ich logikę układową, przy jednoczesnym zachowaniu wysokiej przepustowości. Szybkość przesyłania kanału selektorowego wynosi 1100 Kb/s. Natomiast sumaryczna przepustowość wszystkich kanałów selektorowych wynosi 2,6 Mb/s, podczas gdy teoretycznie maksymalna przepustowość — przy rozwiązaniu całkowicie układowym — mogłaby wynosić 3,3 Mb/s.

Kanały wyposażone są także w pewien dodatkowy sprzęt kontrolny i diagnostyczny, który wykrywa w sposób ciągły błędy pracy kanału oraz umożliwia zapisanie stanu kanału w PAO w celu lokalizacji uszkodzenia.

5.7.4. Pamięć operacyjna

Procesor może wykonywać 3 rodzaje działań na pamięci operacyjnej:
— odczyt słowa 4-bajtowego (32 bity informacyjne + 4 kontrolne),
— zapis słowa 4-bajtowego (32 bity informacyjne + 4 kontrolne),
— zapis dowolnych bajtów w słowie 4-bajtowym.

W przypadku niepełnego zapisu słowa następuje zapisanie bajtów podanych przez procesor do rejestru danych pamięci operacyjnej; następnie odczytuje się z pamięci pozostałe bajty i zapisuje nowo utworzone słowo pod ten sam adres. Przy zapisie procesor cechuje każdy bajt bitem parzystości, a przy odczycie sprawdza poprawność odczytanej informacji. Oprócz tego, przy każdym kontakcie z pamięcią operacyjną adapter może sprawdzać zgodność klucza podanego przez procesor z kluczem wyczytanym z pamięci kluczy ochrony; jest to kontrola systemowa poprawnej pracy programów. Procesor współpracuje także z adapterem przy odczycie i zapisie informacji z/do pamięci kluczy ochrony pamięci.

Każdy blok 128 Kb pamięci operacyjnej ma zbudowaną na układach scalonych pamięć kluczy ochrony (PAK) o objętości 64 bajtów. Tak więc objętość PAK zmienia się w systemie proporcjonalnie do objętości PAO. Jeden cykl pracy PAO (1200 ns) odpowiada 4 cyklom procesora (300 ns), czyli wykonaniu czterech mikroinstrukcji. Mikroprogramy procesora i kanałów są tak zaprojektowane, że procesor w większości przypadków uruchamia kontakt z PAO właśnie co 4 mikroinstrukcje. Dzięki takiemu zsynchronizowaniu procesor jest wstrzymywany przez PAO w bardzo małym stopniu, a PAO rzadko „czeka bezczynnie" na akcję procesora. Każdy blok PAO ma 3-bitowy adres ustawiany na przełącznikach; tym samym można ustalić jego położenie adresowe w obszarze

pamięci operacyjnej. W przypadku niesprawności któregokolwiek z bloków można szybko zrekonfigurować system zmieniając adresy bloków sprawnych i odłączając niesprawne.

5.7.5. Konstrukcja jednostki centralnej

Pakiety logiczne maszyny R-32 są montowane na wielowarstwowych płytkach drukowanych o rozmiarach 2 razy większych (295×150) w stosunku do płytek używanych w maszynach R-20 — R-50. W celu zwiększenia upakowania elementów logicznych na płytkach zastosowano dodatkowe połączenia pomiędzy sąsiednimi pakietami za pomocą łączówek umieszczonych w przedniej części płytek. W ten sposób pakiet logiczny może mieć 2, 3 lub 4 łączówki 84-stykowe; zatem ilość wyprowadzeń może dochodzić do 336. Dzięki zastosowaniu płytek wielowarstwowych i dodatkowych połączeń pomiędzy pakietami oraz starannemu zaprojektowaniu podziału struktury logicznej na pakiety uzyskane dobre wyniki upakowania układów scalonych i małą ilość sprzętu.

Poniższa tabela porównuje wskaźniki wykorzystania powierzchni i łączówek w pakietach maszyn R-20, R-30 i R-32:

Wskaźnik	R-20 (pakiet pojed.	R-30 (pakiet pojed.)	R-32 (pakiet podw.)
Średnie upakowanie elementów scalonych na pakiecie	14	18	81
Średnie wykorzystanie styków w łączówkach	88%	93%	96%
Ilość pakietów logicznych	460	1200	29

Przyjęta konstrukcja elektroniczna, zastosowanie układów scalonych średniej skali integracji oraz efektywne mikroprogramowanie i optymalna struktura logiczna sprawiły, że hardware procesora i kanałów EMC R-32 odznacza się stosunkowo bardzo niewielką objętością, dużą szybkością działania, wysoką niezawodnością i niewielkim kosztem wytwarzania.

Pamięć stała zbudowana jest w technice transformatorowej i rozmieszczona na płytkach 4-warstwowych o wielkości identycznej jak dla pakietów logicznych. Całość pamięci stałej składa się z 16 pakietów o tej samej konstrukcji, różniących się między sobą tylko zawartością informacji „zaszytych" w rdzeniach ferrytowych. Każdy z pakietów zawiera 352 słowa 43-bitowe. Pamięć stała EMC R-32 odznacza się dużą szybkością działania (jak na technikę transformatorową): czas dostępu wynosi 120 ns, a czas cyklu 300 ns. Dzięki zastosowaniu oszczędnych układów sterowania — jeden tranzystor steruje czterema słowami — uzyskano niewielkie rozmiary i niski koszt całej pamięci.

Pamięć operacyjna zbudowana jest na małych rdzeniach ferrytowych (0,54 mm) w tzw. technice planarnej. Polega to na umieszczaniu dużych ilości rdzeni na płytach drukowanych i montowaniu układów elektronicznych w pobliżu matrycy rdzeniowej. Zaletą tego systemu jest możliwość uzyskania: dużej pojemności pamięci w niewielkiej objętości, małego poziomu zakłóceń, dużych szybkości działania i znacznego zmniejszenia pracochłonności wykonania w stosunku do tradycyjnych pamięci opartych o tzw. blok ferrytowy.

Każdy moduł pamięci operacyjnej składa się z płyty z matrycą rdzeniową i płyty sterowania, a pojemność jego wynosi 16 Kbajtów (8 K×18 bitów). Blok PAO o pojemności 128 Kbajtów zawiera 8 modułów, 1 pakiet sterowania i 3 pakiety adaptera. Moduły PAO zakończone są łączówkami i można je wymieniać między sobą, co znacznie przyśpiesza wykrywanie uszkodzeń i naprawy.

System zasilania do standardowego wykonania EMC R-32 składa się z dwóch bloków prostowniczych (pierwszy dla logiki i pamięci stałej, drugi dla pamięci operacyjnej — 256 Kbajtów), układów automatyki i szeregu jednakowych — dla danego napięcia — stabilizatorów. Układy automatyki wykonują automatyczne włączanie i wyłączanie zasilania systemu, kontrolują pracę stabilizatorów, przeprowadzają automatyczne pomiary napięć i wysyłają do procesora i bloków pamięci operacyjnych sygnały ostrzegające o awarii zasilania. Ta ostatnia właściwość automatyki pozwala procesorowi zapamiętać w pamięci operacyjnej stan maszyny i odtworzyć go po włączeniu napięć. W ten sposób można kontynuować pracę programów przerwanych wskutek awarii zasilania lub zaniku napięć w sieci. Stabilizatory napięć zakończone są łączówkami i można je wymieniać między sobą, co znacznie ułatwia naprawy.

5.7.6. System kontroli i diagnostyki

Wymagania nałożone na maszynę R-32 zakładają nie tylko dużą jej wydajność i elastyczność konfiguracji, lecz także — co jest nie mniej ważne dla użytkownika — dużą niezawodność systemu i jak najkrótszy czas międzyawaryjny przestojów. W konstrukcji maszyny przyjęto między innymi dwa podstawowe wymagania [7]:

1) średni użyteczny czas pracy — nie mniejszy niż 95%,
2) średni czas wznowienia pracy maszyny — nie dłuższy niż 30 minut.

W celu osiągnięcia powyższych wymagań wbudowano różne układy kontroli oraz układy diagnostyczne.

Przyjęty system kontroli obejmuje wiele punktów przepływu informacji, co umożliwia efektywne i dynamiczne sprawdzanie poprawności pracy maszyny. Wykrycie błędu może spowodować wykonanie erratografii stanu maszyny i uaktywnić diagnostyczną część systemu operacyjnego. Jeżeli uszkodzenie ma charakter chwilowy, to fakt wystąpienia błędu i wykonanie erratografii maszyny podczas jego zaistnienia mogą być wykorzystane do tworzenia statystyki uszkodzeń maszyny. Natomiast w przypadku uszkodzenia stałego wykonanie szeregu procedur diagnostycznych pozwala na lokalizację uszkodzonego elementu. Dokładność tej lokalizacji zależy od położenia elementu w strukturze logicznej.

System kontroli jednostki centralnej. Określone węzły przepływu informacji w maszynie są kontrolowane w sposób ciągły. W przypadku wykry-

cia błędu i gdy bit 13 słowa SSP zezwala na przerwanie typu błąd maszyny, zachodzi natychmiast przerwanie po wykryciu błędu. Poza tym wprowadzono okresową kontrolę maszyny pozwalającą z określoną częstotliwością sprawdzić dokładnie arytmometr za pomocą testów mikroprogramowych. Częstotliwość i czas trwania tego testu są tak dobrane, aby nie zabierać więcej niż 1% czasu pracy maszyny.

W procesorze kontrolowane są w sposób ciągły następujące węzły przepływu informacji:
— układ interface'u PAO-procesor wraz z pamięcią operacyjną,
— informacja czytana z pamięci mikroprogramów (PAM),
— układy wybierania mikroprogramów kanałowych,
— układy interface'u kanał-jednostki sterujące urządzeniami wejścia-wyjścia.

Kontrolę prawidłowej pracy PAO oraz szyny PAO-procesor przeprowadza się badając na wejściu do procesora prawidłową parzystość każdego z czterech przeczytanych bajtów. W przypadku wykrycia błędu informacja odczytana z błędem nie zostanie wykorzystana przez maszynę.

Prawidłowość cechy parzystości informacji czytanej z PAM jest sprawdzana przy każdym odczycie. W przypadku wykrycia błędu pobrana mikroinstrukcja nie spowoduje zmiany zawartości rejestrów maszyny.

Kontrola poprawności systemu we-wy polega na sprawdzeniu prawidłowego wejścia do mikroprogramowanych funkcji kanału oraz poprawności pracy interface'u we-wy. Oznacza to, że w każdym przypadku wejścia do mikroprogramu kanałowego sprawdza się, czy sprzęt prawidłowo wybrał pierwszą mikroinstrukcję tego mikroprogramu. W przypadku błędu normalna akcja kanału zostaje zawieszona. W interface'ie wejścia-wyjścia istnieje bit parzystości do sprawdzania poprawności przyjętych informacji. Poza tym specjalne pętle mikroprogramowe odliczają w określonych przypadkach czasy maksymalne przewidziane na ustalone następstwo pytań i odpowiedzi w interface'ie we-wy. W przypadku wykrycia błędu przez jeden z tych układów reakcja systemu zależy od ustawienia bitu 13 słowa SSP. Gdy bit ten jest równy 1 — wykonuje się erratografię procesora, koordynatora kanałów oraz kanału, w którym nastąpił błąd, po czym następuje przerwanie programu typu błąd maszyny. Gdy natomiast bit 13 słowa SSP jest równy 0 — następuje wykonanie erratografii kanału, w którym wykryto błąd, a następnie przerwanie typu we-wy. W słowie stanu kanału odpowiednie bity wskazują na rodzaj błędu.

Kontrola okresowa procesora. Procesor jest poddawany co 20 ms okresowej kontroli, tzn. za pomocą tzw. trudnych szablonów sprawdza się poprawność działania układów procesora. Niezgodność wyników ze wzorem powoduje przerwanie typu błąd maszyny.

System diagnostyki jednostki centralnej.

A. Diagnostyka bloków pamięci operacyjnej

Pamięć operacyjna ma wbudowane układy, umożliwiające autonomiczne sprawdzanie fizycznego bloku pamięci za pomocą specjalnego testera i po uprzednim odłączeniu logicznym bloku od szyny PAO-procesor. Pozostałe bloki mogą pracować w systemie po dokonaniu rekonfiguracji za pomocą odpowiednich przełączników. Poza tym procesor posiada w PAM zespół testów sprawdzających wszystkie bloki PAO dowolną informacją, zadaną z klawiatury pulpitu technicznego, i tzw. trudnym szablonem pamięci. Istniejąca sygnalizacja na pulpicie technicznym pozwala na dokładną lokalizację błędu.

B. Erratografia stanu maszyny po błędzie

W przypadku wykrycia błędu przez układy ciągłej kontroli i przy zezwoleniu na przerwanie typu błąd maszyny (bit 13 SSP=1) maszyna rozpoczyna automatycznie akcję — realizowaną zarówno sprzętowo, jak i mikroprogramowo — zakończoną przerwaniem typu błąd maszyny. Pierwszym etapem tej akcji jest blokada impulsów sterujących pracą procesora i kanałów, co zapobiega wykorzystaniu błędnej informacji przez procesor. Następnie wymusza się stały adres wejścia do PAM, od którego rozpoczyna się mikroprogram erratografii. Erratografia zostaje zapisana w początkowym polu pamięci zarezerwowanym dla Supervisora.

Zapisany stan maszyny — w zarezerwowanym polu PAO — składa się z zawartości różnych rejestrów procesora i kanałów. Są to na przykład: rejestr adresowy PAO, rejestr adresowy PAM, rejestr błędów, wszystkie rejestry informacyjne i sterujące arytmometru, rejestry sterujące i buforowe kanału, który w momencie wystąpienia błędu współpracował z CJP itp. Sekwencja zapisu stanu maszyny po błędzie kończy się — podobnie jak i inne przerwania — zwykłą wymianą starego i nowego SSP, co umożliwia uaktywnienie odpowiedniego pakietu procedur diagnostycznych systemu operacyjnego.

Do rejestracji różnych typów błędów maszyny wprowadzono specjalny 8-bitowy rejestr, zwany rejestrem błędów (RB). Bity 0—3 tego rejestru są przeznaczone do rejestrowania błędów parzystości z czterech bajtów informacji odczytywanej z PAO. Sygnał błędu jest ładowany podczas każdego odczytu z PAO. W kolejnych bitach tego rejestru notuje się błąd w PAM, błąd w kanale, błąd wykryty przez okresowy test maszyny. Zawartość RB jest sprawdzana w każdym cyklu maszynowym. W przypadku niezerowej zawartości i zezwolenia na przerwanie typu błąd maszyny sterowanie automatycznie inicjuje wykonanie erratografii stanu maszyny po błędzie.

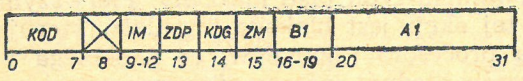
C. Instrukcja DIAGNOZA

W repertuarze instrukcji JS EMC wprowadzono uprzywilejowaną instrukcję, zwaną DIAGNOZA, która powoduje uruchomienie określonej sprzętowej procedury diagnostycznej. W maszynie R-30U procedurami diagnostycznymi są specjalne mikroprogramy (przechowywane w PAM), którym przyporządkowano określone funkcje do celów kontroli i diagnostyki. Są to np. mikroprogramy realizujące takie funkcje, jak:
— zapamiętanie (w PAO) stanu procesora,
— zapamiętanie stanu kanałów,
— zapamiętanie stanu wybranego kanału selektorowego,
— zapamiętanie zawartości pamięci lokalnej,
— zapamiętanie pamięci multipleksera (tj. pamięci słów sterujących podkanałów),

— ładowanie rejestrów procesora (informacją z PAO),
— ładowanie pamięci lokalnej itp.

Instrukcja DIAGNOZA jest wykonywana w dwóch etapach. W pierwszym etapie ustawia się różne liczniki i przerzutniki sterujące, tzw. przerzutniki diagnostyczne, które bliżej określają sposób wykonania tej instrukcji. Natomiast w drugim etapie następuje wejście do mikroprogramowej procedury diagnostycznej, według początkowego adresu tej procedury zapisanego w odpowiedniej części instrukcji DIAGNOZA.

Instrukcja DIAGNOZA, składająca się z dwóch półsłów, ma strukturę nietypową (rys. 5.1).



Rys. 5.1. Struktura instrukcji DIAGNOZA

Adres efektywny E1:=(B1)+A1 określa adres pierwszej mikroinstrukcji do wykonania, tj. pierwszej mikroinstrukcji uruchamianej procedury diagnostycznej. Natomiast pola tej instrukcji — oznaczone przez IM, ZDP, KDG i ZM — zawierają początkowe parametry, według których ustawia się stan przerzutników diagnostycznych (w tym i licznika diagnostycznego) mających takie same oznaczenia.

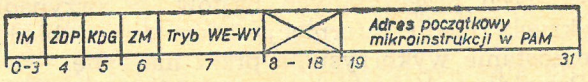
Poniżej opisano rolę przerzutników diagnostycznych.

Ilość mikroinstrukcji (IM). Jest to wielkość 4-bitowa przechowywana — w czasie wykonywania instrukcji DIAGNOZA — w układowym liczniku IM. Wielkość ta określa ilość mikroinstrukcji do wykonania przez instrukcję DIAGNOZA. Oznacza to, że po wykonaniu kolejnej mikroinstrukcji stan licznika IM zostaje pomniejszony o jeden. Po uzyskaniu stanu zerowego tego licznika przechodzi się automatycznie do wykonania następnej instrukcji diagnostycznej. W ten sposób można wykonać dowolną sekwencję mikroinstrukcji (nie tylko procedury diagnostycznej) nie przekraczającą 16 mikroinstrukcji. Dłuższe sekwencje mikroinstrukcji wykonuje się przez wielokrotne zastosowanie instrukcji DIAGNOZA.

Przerzutnik ZM. Przerzutnik ten, w stanie ZM=1, zezwala na odejmowanie jedynek od licznika IM w czasie wykonywania instrukcji DIAGNOZA. Przerzutnik zabrania również kontaktów z PAO, pomimo że takie kontakty występują w wykonywanym mikroprogramie.

Przerzutnik ZDP (zezwolenie na dostęp do PAO). Jeśli przerzutnik ten jest w stanie 1, zezwala się na kontakty z PAO niezależnie od stanu przerzutnika ZM.

Przerzutnik KDG (kontynuj diagnozę). Przerzutnik ten w stanie 1 powoduje wykonanie „łańcucha" procedur diagnostycznych uruchamianych jedną instrukcją DIAGNOZA, gdzie każda procedura tego łańcucha ma swój opis w postaci tzw. diagnostycznego słowa sterującego (DSS). Słowo DSS zawiera właściwie te same parametry co instrukcja DIAGNOZA i ma postać zilustrowaną na rys. 5.2.



Rys. 5.2. Struktura diagnostycznego słowa sterującego

Programowe środki diagnostyczne. Jak już było powiedziane, wraz z przebiegiem obliczeń zachodzi proces kontroli (układowej lub okresowo mikroprogramowej) poprawności pracy procesora lub kanałów. Jeśli kontrola ta wykryje błąd, zachodzi przerwanie programowe typu błąd maszyny (przy zezwoleniu na przerwanie) i tym samym następuje przejście do programu obsługi przerwań typu błąd maszyny. Program ten może różnie zadziałać, np. ograniczyć się tylko do rejestracji faktu błędu chwilowego i wykonanej erratografii stanu CJP w momencie jego wystąpienia lub też uruchomić złożone programowe procedury diagnostyczne w celu zlokalizowania uszkodzonego elementu. W tych programowych procedurach wykorzystuje się instrukcję DIAGNOZA oraz mikroprogramowe procedury diagnostyczne, za pomocą których generuje się różne robocze stany procesora i kanałów. Analiza takich stanów umożliwia zlokalizowanie źródła uszkodzenia.

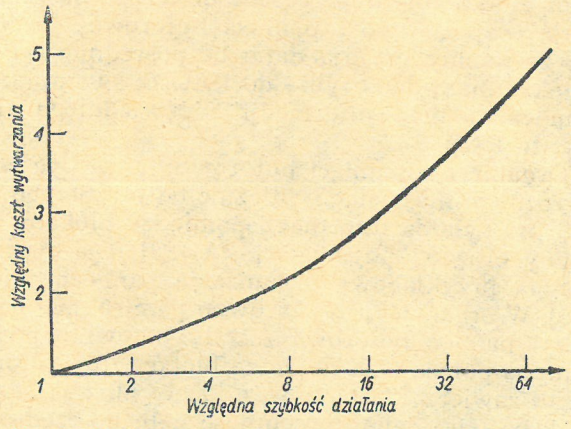
6. Pamięć buforowa

6.1. Wprowadzenie

Pamięć operacyjna maszyny cyfrowej jest to pamięć, której elementy (np. bajty) są wprost adresowane przez instrukcje programu. W pamięci operacyjnej, zwanej też pamięcią główną, przechowuje się programy i dane biorące udział w procesie przetwarzania. Szybkość działania, pojemność i cena pamięci operacyjnej wahają się na „średnim" poziomie w porównaniu z innymi pamięciami, na przykład z wolnymi pamięciami zewnętrznymi lub z tzw. szybko działającymi pamięciami buforowymi. Średnia realna (w gotowym urządzeniu) cena jednego bitu pamięci operacyjnej wynosiła w roku 1974 około 1 centa, podczas gdy cena jednego bitu pamięci dyskowej wynosiła tylko 0,003 centa, zaś cena szybko działających małych pamięci buforowych (cykl 70 ns) — około 8 centów.

Przeciętna użytkowa szybkość działania pamięci operacyjnej współczesnej maszyny cyfrowej waha się w przedziale 0,70÷1,5 mikrosekundy.

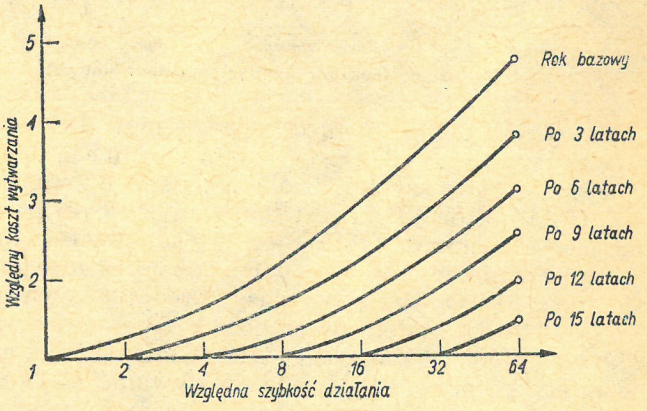
Szybko działającymi pamięciami buforowymi nazwano umownie pamięci, które mają cykl pracy przynajmniej o rząd krótszy od cyklu pracy pamięci operacyjnej, czyli cykl z przedziału 70÷150 nanosekund. Oczywiście wydajność systemu obliczeniowego w istotny sposób w przeszłości zależała i wciąż jeszcze zależy od szybkości pamięci operacyjnych. Dzieje się tak dlatego, że cykl pracy procesora jest przeciętnie o rząd krótszy od cyklu pracy pamięci operacyjnej. Zatem cykle te nie są zbilansowane. Niemniej zwiększenie wydajności maszyny przez zwiększenie szybkości działania pamięci operacyjnej nie prowadzi do ekonomicznych rozwiązań. Skrócenie bowiem cyklu pracy pamięci powoduje wzrost kosztu jej wytwarzania.



Rys. 6.1. Zależność kosztu wytwarzania od szybkości działania pamięci

Również logiczne metody zwiększenia szybkości działania pamięci, np. przez zrównoleglenie pracy wielu bloków, też nie prowadzą do ekonomicznych rozwiązań, a ponadto komplikują strukturę.

Dane literaturowe podają, że podwojenie szybkości działania pamięci pociąga za sobą 30-procentowy wzrost jej ceny (rys. 6.1 ilustruje tę ogólną zależność). Oznacza to, że w dowolnym etapie technicznego rozwoju pamięci można zbudować pamięć wolniejszą i tańszą od danej pamięci o określonej charakterystyce. Oczywiście charakterystyka techniczna przeciętnej produkcyjnej pamięci nieustannie polepsza się; np. średnio co 3 lata szybkość pamięci w stosunku do ceny podwaja się. Rozwój ten pokazano na rys. 6.2.



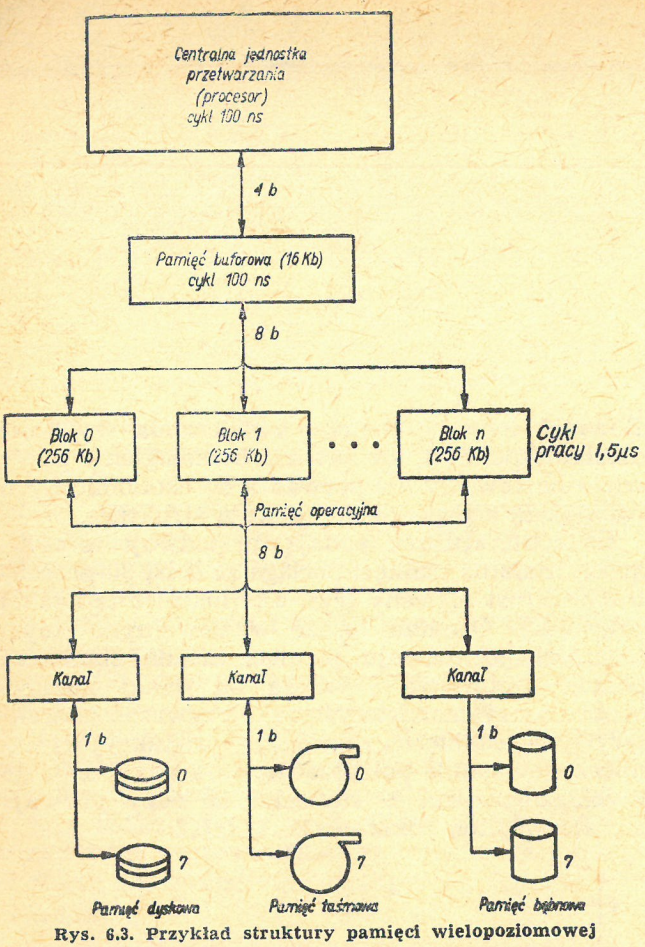
Rys. 6.2. Zmiany zależności szybkości działania i kosztów wytwarzania pamięci

W celu osiągnięcia wysokiej łącznej wydajności maszyny przy niskich kosztach jej wytwarzania przyjmuje się w maszynach JS EMC hierarchiczne organizacje pamięci maszyny. W organizacjach hierarchicznych (lub wielopoziomowych) występuje kilka typów pamięci o zróżnicowanej szybkości działania i kosztach wytwarzania, przy czym w wielopoziomowej organizacji pamięci maszyny daje się na ogół uzyskać zadowalające dla użytkownika rozwiązanie. Użytkownik bowiem otrzymuje:

— dużą łączną pamięć,
— dostatecznie szybką pamięć roboczą (zbliżoną np. do szybkości procesora),
— stosunkowo tanią pamięć.

Przykład struktury wielopoziomowej pamięci maszyny przedstawia rys. 6.3.

Pamięć operacyjna, zawierająca programy i dane biorące udział w przetwarzaniu w pewnym przedziale czasu, jest w wielu przypadkach zbyt wolna w stosunku do szybkości procesora. W naszym przykładzie cykl procesora jest 15 razy



Rys. 6.3. Przykład struktury pamięci wielopoziomowej

krótszy od cyklu pamięci operacyjnej. Dlatego wstawia się pomiędzy procesorem i pamięcią operacyjną (PAO) szybko działającą pamięć buforową (PAB), której cykl jest taki sam jak cykl procesora. Pojemność PAB jest stosunkowo mała i przeważnie wynosi 8—16 Kb, podczas gdy pojemność PAO maszyn klasy średniej wynosi 512—1024 Kb. Zatem pojemność PAB stanowi zaledwie 1,5% pojemności PAO. Główną funkcją pamięci buforowej jest przechowywanie najbardziej aktywnych fragmentów programów w procesie liczenia. Dzięki temu większość odwoływań po instrukcje lub dane programu będzie kierowana do szybkiej PAB zamiast do wolnej PAO. W ten sposób średni cykl obu pamięci zbliża się do cyklu PAB. Według danych literaturowych zależność między pojemnością PAB (podzieloną np. na bloki po 16 bajtów) a procentową ilością odwoływań do niej jest następująca:

pojemność PAB	4 Kb	8 Kb	16 Kb	32 Kb
procent odwołań do PAB	69%	77%	84%	90%

Jeśli np. cykl PAO wynosi 1500 ns, zaś cykl PAB wynosi 100 ns, a 84% odwoływań ma miejsce względem PAB, średni użytkowy cykl pamięci wynosi:

$$324\,\text{ns}\left(=1500\cdot\frac{16}{100}+100\cdot\frac{84}{100}\right)$$

Zatem użytkowy cykl pamięci roboczej został polepszony prawie o 500% przy minimalnym wzroście kosztów związanych z wprowadzeniem pamięci buforowej (tj. około 10% kosztu PAO).

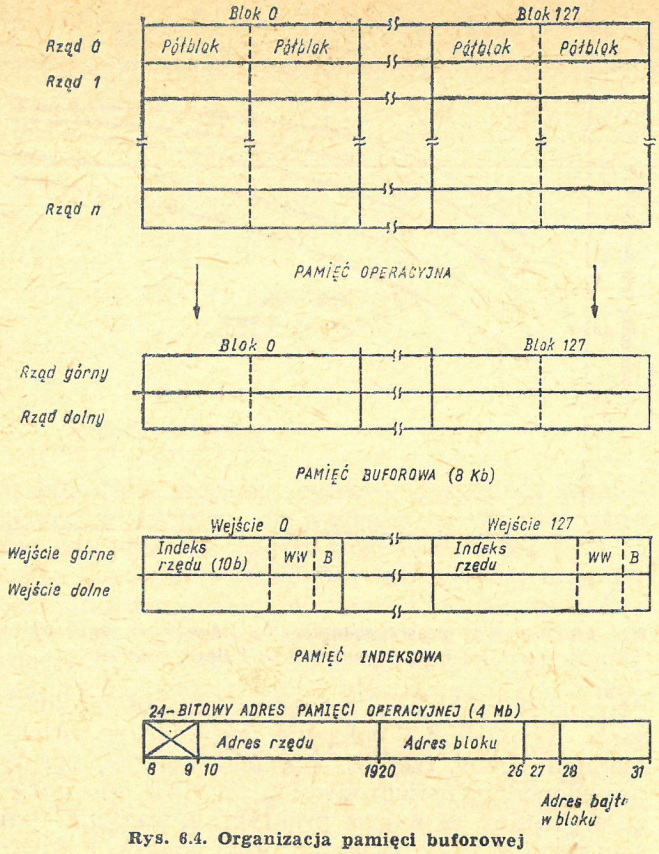
Informacja przechowywana w PAO wystarcza na ogół tylko na kilkusekundową ciągłą pracę danego programu, którego większa część znajduje się w pamięci zewnętrznej. W takich przypadkach zachodzi konieczność wprowadzania do PAO nowych porcji informacji związanych z dalszą pracą programu oraz wyprowadzania innych informacji z PAO do pamięci zewnętrznej w celu zwolnienia wymaganego miejsca w PAO dla tych nowych informacji. W ten sposób oprócz dwóch pierwszych poziomów pamięci, tj. pamięci buforowej i operacyjnej, występuje trzeci poziom pamięci, mianowicie poziom dużej pamięci zewnętrznej (np. pamięci dyskowej, taśmowej lub bębnowej). Pamięć zewnętrzna pełni przeważnie funkcję archiwum programów i danych, przy czym procesor nie ma bezpośredniego dostępu do informacji przechowywanych w pamięci zewnętrznej. Informacje te stają się dla niego dostępne dopiero po wprowadzeniu ich z pamięci zewnętrznej do PAO za pomocą odrębnych operacji kanałowych.

6.2. Organizacja pamięci buforowej

Pamięć buforowa, która zawiera najbardziej aktywne fragmenty programów, jest wykorzystywana w sposób następujący: jeśli procesor żąda pobrania kolejnych informacji (instrukcji lub danych), to jednostka sterująca pamięci buforowej sprawdza, czy żądane informacje znajdują się w PAB. Jeśli informacje te znajdują się w PAB, natychmiast zostają pobrane do procesora bez zainicjowania zwrotu do PAO. Jeśli natomiast żądane informacje nie znajdują się w PAB, zostają zapoczątkowane czytanie tych informacji z PAO, po czym przesyła się je bezpośrednio do procesora. Jednocześnie czytane z PAO informacje zostają zapisane do PAB. W ten sposób, przy ponownym żądaniu pobrania do procesora tych samych informacji, będzie można natychmiast pobrać je z PAB. Zawartość PAB jest dynamicznie aktualizowana w miarę żądań procesora. Zapełnianie PAB informacją wykorzystywaną przez procesor odbywa się w pełni sprzętowo i bez ingerencji programisty. W przypadku żądania procesora przesłania informacji do określonych komórek PAO, zapamiętuje się te informacje do PAO, a ponadto aktualizuje się PAB, jeśli zawartość rozpatrywanych komórek znajduje się także w PAB.

Operacje kanałowe natomiast, które wprowadzają lub wyprowadzają dane nie związane na ogół z aktualnym procesem obliczeń, nie korzystają bezpośrednio z pamięci buforowej. Oznacza to, że kanały czytają dane bezpośrednio z PAO, a także piszą dane tylko do PAO (z ewentualnym unieważnieniem informacji odpowiedniego fragmentu PAB).

Organizacja pamięci buforowej może być różna w różnych maszynach, zależnie od przyjętych metod przesyłania informacji pomiędzy blokami maszyny oraz od pojemności PAB. Rys. 6.4. przedstawia przykładową organizację pamięci buforowej. W organizacji tej zarówno pamięć operacyjna jak i pamięć buforowa zostały umownie podzielone na pewną ilość rzędów i bloków (lub kolumn). Rząd zawiera 128 bloków, a każdy blok składa się z 32 bajtów. Zatem pojemność jednego rzędu wynosi 4 Kb. Pamięć operacyjna może mieć dowolną



Rys. 6.4. Organizacja pamięci buforowej

ilość rzędów, np. od 64 do 1024 rzędów, co odpowiada pojemności od 256 do 4096 Kbajtów (przy czym organizacja pamięci buforowej nie zależy od pojemności PAO). Pojemność pamięci buforowej może składać się z 4 lub 8 Kbajtów, czyli z jednego lub dwóch rzędów.

Informacje z PAO są pobierane do PAB w sposób „równoległy", tzn. j-ty blok (j=0,...,127) z dowolnego rzędu PAO pobierany jest zawsze do j-tego bloku PAB (do rzędu górnego lub dolnego według określonego algorytmu).

Do identyfikacji numeru rzędu pamięci operacyjnej, z którego pochodzi dana informacja znajdująca się w PAB, wprowadzono specjalną pamięć (lub tablicę) zwaną pamięcią indeksową (PAI). Pamięć indeksowa składa się z pewnej ilości szybkich rejestrów, w których zapamiętuje się numery lub indeksy rzędów. Ilość tych rejestrów musi być taka, jaka jest liczba bloków w PAB, czyli 128 lub 256. Każdy rejestr pamięci indeksowej składa się z 13 bitów, z których 10 określa indeks rzędu PAO, a dwa bity WW oraz bit B podają dodatkowe informacje dotyczące danego bloku w PAB. Adresy rejestrów (tzw. wejścia pamięci indeksowej) są jednoznacznie przyporządkowane blokom pamięci buforowej. Jeśli zatem PAB ma podwójną pojemność, tj. dwa rzędy, to również występuje podwójna ilość rejestrów (tzw. wejścia górne i wejścia dolne).

W celu zmniejszenia czasu pobrania informacji z PAO do PAB (gdyż na ogół potrzebny jest tylko mały fragment informacji) pobiera się jednorazowo do PAB nie pełny blok, lecz tylko żądany półblok, tj. 16 bajtów. Pobranie to zajmuje dwa cykle PAO, jeśli szerokość szyny przekazania informacji z PAO do PAB wynosi 8 bajtów (jak pokazano na rys. 6.3). Powstaje zatem sytuacja, w której tylko jeden półblok z danego bloku znajduje się w PAB. Z drugiej strony indeks rzędu w pamięci indeksowej związany jest z całym blokiem, innymi słowy jest taki sam dla obu półbloków. W związku z tym wprowadzono w każdym rejestrze PAI dwa bity WW, które określają ważność informacji odpowiedniego półbloku pamięci buforowej. Na przykład kombinacje: 10, 01, 11 i 00 określają odpowiednio, że: tylko lewy półblok zawiera ważną informację, tylko prawy półblok zawiera ważną informację, w obu półblokach znajduje się informacja do wykorzystania i oba półbloki zawierają nieważną informację (tzn. niezgodną z informacją w PAO). Bity WW przyjmują stan 1, gdy ich półblok w PAB zostaje załadowany, natomiast stan 0, gdy do półbloku odpowiedniego rzędu PAO wpisuje się nową informację (np. z kanału).

Rolą bitu B rejestrów pamięci indeksowej jest sygnalizowanie stałych uszkodzeń odpowiedniego bloku pamięci buforowej lub rejestru pamięci indeksowej (wykrytych np. metodą sprawdzania nieparzystości jedynek w bajcie). Jeśli zatem wykryje się w procesie obliczeń trwały błąd w PAB, to wpisuje się 1 do bitu B danego bloku. W ten sposób wyłącza się z pracy nie całą PAB, lecz tylko blok, w którym wystąpił trwały błąd. Oczywiście zwroty do uszkodzonego bloku PAB nie zostają realizowane, lecz kierowane są zawsze do PAO.

6.3. Algorytmy przydzielania bloków pamięci buforowej

Jak już wiemy, półbloki informacji z PAO pobierane są do PAB w sposób równoległy, tzn. pobiera się z PAO półblok określony przez bity 20÷27 adresu efektywnego i należący do rzędu określonego przez bity 10÷19 tego adresu.

Pobrany z PAO półblok może zostać umieszczony w górnym lub dolnym rzędzie pamięci buforowej bez naruszenia zasady równoległości. Przy przydzielaniu bloków pamięci buforowej uwzględnia się również bity WW pamięci indeksowej oraz dodatkowy ogólny przerzutnik znajdujący się w jednostce sterującej PAB. Przerzutnik ten, zwany GD (górny—dolny), zmienia się dynamicznie w procesie korzystania (czytania) z pamięci buforowej i wskazuje rząd PAB (0 lub 1), z którego pobrano ostatnie informacje do procesora. Jeśli zatem przerzutnik GD=0 (odpowiednio GD=1), oznacza to, że ostatnia informacja z PAB została pobrana z górnego (dolnego) rzędu PAB. W czasie pisania informacji z procesora do PAB, które zachodzi rzadko w stosunku do ilości odczytów, przerzutnik GD nie zmienia swojego stanu. Negacja stanu tego przerzutnika wskazuje (w niektórych przypadkach) rząd, do którego należy wprowadzić kolejny pobrany z PAO półblok. Pobranie informacji z PAO odbywa się półblokami, lecz przydzielanie miejsca w PAB dotyczy całego bloku. Pełny algorytm przydzielania bloków PAB informacjom pobranym z PAO może być następujący:

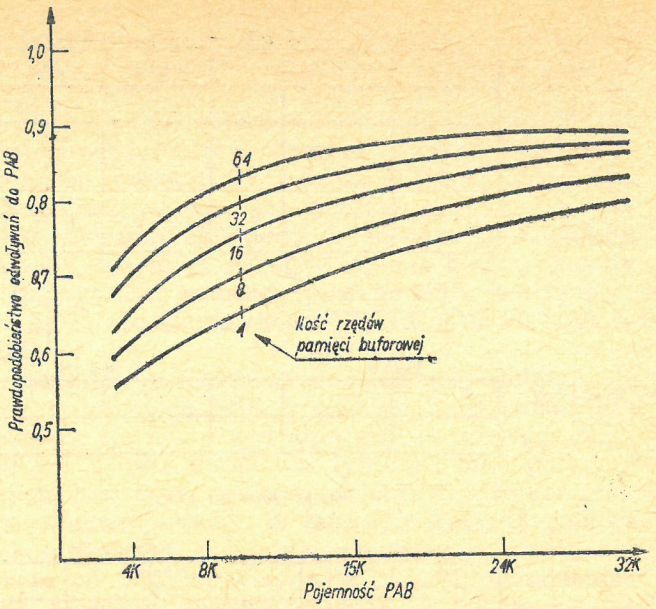
1) jeśli adres rzędu pobranej z PAO informacji jest równy indeksowi rzędu z górnego lub dolnego wejścia pamięci indeksowej, to przydziela się ten rząd pamięci buforowej, która zawiera zgodny indeks; dzięki temu oba półbloki danego bloku PAO pobrane zostaną do półbloków PAB tego samego rzędu;

2) jeśli adres rzędu pobranej z PAO informacji nie jest zgodny z żadnym indeksem rzędu pamięci indeksowej, rozpatruje się następujące przypadki:
   a) jeśli odpowiednie bloki w obu rzędach PAB, tj. zarówno w górnym jak i w dolnym rzędzie, nie są zajęte (wówczas występują zera w bitach WW), przydziela się rząd zgodnie ze wskazaniem przerzutnika GD;
   b) jeśli odpowiednie bloki w obu rzędach PAB mają przydzielone przynajmniej po jednym półbloku (czyli żaden blok nie jest wolny), przydziela się rząd zgodnie ze wskazaniem przerzutnika GD;
   c) jeśli odpowiedni blok jednego rzędu PAB ma już przydzielony przynajmniej jeden półblok, zaś odpowiedni blok drugiego rzędu jest w całości wolny, przydziela się rząd wolny niezależnie od wskazań przerzutnika GD.

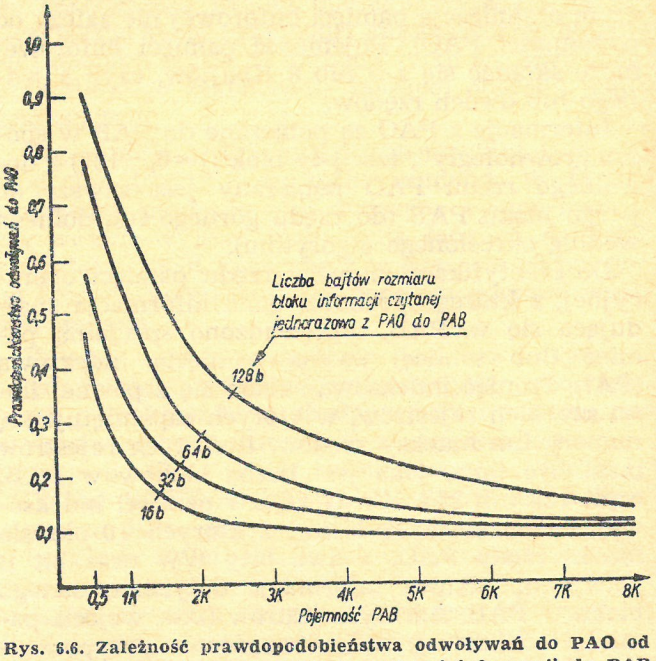
Przyjęta organizacja pamięci buforowej oraz sposób przydzielania bloków PAB nowo pobranym informacjom z PAO pozwalają na jednoczesne przechowanie w pamięci buforowej informacji z dwóch bloków mających ten sam numer bloku i należących do różnych rzędów PAO. Właściwość ta jest bardzo istotna, gdyż pozwala na jednoczesne przechowywanie w PAB zarówno aktywnych fragmentów instrukcji, jak i aktywnych fragmentów danych, należących normalnie do różnych rzędów PAO. Brak tej właściwości powodowałby bardzo częste ładowania pamięci buforowej, zapoczątkowane instrukcjami tego samego programu. Jeśli bowiem ciąg instrukcji typu PAMIĘĆ—REJESTR znajduje się w pewnym bloku PAO, a ciąg danych związanych z tymi instrukcjami przechowany jest w „równoległym" bloku innego rzędu, to pobranie kolejnej instrukcji do procesora będzie powodować pobranie do PAB całego jej półbloku wymazując z PAB dane, które będą jeszcze potrzebne w tej samej instrukcji.

Przydział pamięci buforowej z dokładnością do półbloku jest również możliwy, jeżeli przeznaczy się podwójną ilość wejść (czyli nie 128, lecz 256). W takim przypadku informacje przechowywane w jednym bloku PAB nie muszą pochodzić z półbloków tego samego rzędu PAO. Efektywność pracy PAB również się polepszy, jeżeli wprowadzi się oddzielne przerzutniki GD dla każdego wejścia pamięci indeksowej (czyli wbuduje się 256 przerzutników typu GD zamiast jednego wspólnego dla całej PAB). Można również zwiększyć ilość rzędów pamięci buforowej np. z dwóch do czterech lub do ośmiu. W takich przypadkach nieco komplikuje się sprawdzenie zawartości pamięci indeksowej z bieżącym adresem oraz przydzielanie miejsc w pamięci buforowej, lecz pozwala to na jednoczesne przechowywanie w PAB więcej niż dwu aktywnych obszarów.

Przy zadanych rozmiarach bloku wymiany informacji i szybkości jej przesyłania najważniejszym parametrem wydaje się być pojemność pamięci buforowej. Oczywiście w idealnym przypadku pamięć buforowa powinna zawierać całą informację, po którą procesor zwraca się do pamięci operacyjnej. W rzeczywistości efektywność pracy pamięci buforowej zależy od wielu czynników, jak np. od pojemności PAB, od liczby rzędów, od ilości bloków w rzędzie, od sposobu adresacji. Rysunek 6.5 pokazuje, że wraz ze zwiększeniem liczby rzędów, np. od 4 do 64, rośnie prawdopodobieństwo odwołań do PAB, ale pokazuje również, że można praktycznie otrzymać ten sam rezultat przy mniejszej ilości rzędów, a większej pojemności pamięci buforowej. Z kolei z rys. 6.6 wynika, że dla różnych pojemności pamięci buforowej i przy odpowiednim doborze wielkości bloków wymiany informacji można uzyskać jednakowe prawdopodobieństwo zgłoszeń do pamięci operacyjnej (przy braku żądanej informacji w PAB). Zatem przyjęta organizacja pamięci buforowej jest na ogół kompromisem pomiędzy wydajnością (na wzrost której wpływ mają: duża ilość rzędów, mały rozmiar bloku, duża pojemność PAB), a ilością wbudowanego sprzętu zarówno w samej pamięci buforowej, jak i w pamięci indeksowej oraz w układach równoległego porównania indeksów rzędów.



Rys. 6.5. Zależność prawdopodobieństwa odwoływań do pamięci buforowej i od jej pojemności i liczby rzędów



Rys. 6.6. Zależność prawdopodobieństwa odwoływań do PAO od pojemności PAB i rozmiaru bloku czytanej informacji do PAB

7. Pamięć wirtualna

7.1. Wprowadzenie

Współczesne maszyny cyfrowe dysponują pamięciami, w których ze względu na wykonywane funkcje wyróżnia się pamięć operacyjną (zwaną także pamięcią wewnętrzną lub główną) oraz jeden lub kilka poziomów pamięci zewnętrznej. Pamięć operacyjna jest przeważnie realizowana na rdzeniach ferrytowych lub na półprzewodnikach, zaś pamięć zewnętrzna to głównie pamięci dyskowe, bębnowe lub taśmy magnetyczne, a czasem nawet bardzo wolna i bardzo duża pamięć rdzeniowa. Pamięć operacyjna jest na ogół bardzo szybka (zwykle o cyklu 0,5 do 2 mikrosekund na jedno słowo), lecz droga i o niezbyt dużej pojemności (zwykle do 2048 Kbajtów w dużych maszynach), a pamięć zewnętrzna jest w porównaniu z nią tańsza i większa (zwykle 10-500 Mbajtów), lecz charakteryzuje się istotnie dłuższym czasem dostępu do informacji. Z tego względu pamięć zewnętrzna jest wykorzystywana w systemach jako magazyn do przechowywania programów i danych aktualnie nie wykorzystywanych.

Oprócz powszechnego stosowania pamięci operacyjnych i pamięci zewnętrznych w systemach liczących stosuje się także tzw. pamięci buforowe, które są najszybszymi — zwykle o cyklu 50—200 nanosekund — pamięciami systemu liczącego, lecz o niezbyt dużej pojemności (4—16 Kbajtów). Pamięci te, zwykle półprzewodnikowe, są wykorzystywane jako robocze, tzn. przechowuje się w nich tylko te fragmenty programów i danych, które aktualnie są przetwarzane. Dzięki temu praktycznie około 90% kontaktów pamięciowych ma miejsce w odniesieniu do pamięci buforowej. Zatem we współczesnych systemach liczących pamięć jest hierarchicznie zorganizowana (pamięć buforowa, pamięć operacyjna, pamięć zewnętrzna), dzięki czemu uzyskuje się wysoką wydajność sumaryczną systemu przy optymalnym wykorzystaniu wszystkich typów pamięci.

Podział pamięci na różne poziomy oraz organizacja pracy programu wykorzystującego jawnie zarówno pamięć operacyjną jak i pamięć zewnętrzną jest rzeczą zbyt skomplikowaną dla użytkownika. Dlatego też wprowadza się pamięć jednolicie adresowaną, to jest pamięć wirtualną. Przez pamięć wirtualną rozumie się jednolitą pamięć, jaką dysponuje użytkownik, przy czym nie jest on świadomy jej podziału na pamięć operacyjną i zewnętrzną, ani faktu, że dzieli on tę pamięć z innymi użytkownikami.

7.1.1. Techniki przeadresowania

Stosowana w maszynach cyfrowych pamięć operacyjna nie jest na ogół dostatecznie duża, aby można było przechowywać w niej w całości wszystkie programy. Dlatego normalnie programy przechowuje się w pamięci zewnętrznej, a do pamięci operacyjnej sprowadza się tylko te programy (lub ich części), które będą wykonywane. Wykonywanie programu w maszynie cyfrowej odbywa się na stałych adresach, oznaczających określone fizyczne komórki pamięci operacyjnej, aczkolwiek z wielu względów często wygodnie jest przechowywać program w adresach względnych (relatywnych). W związku z tym wbudowuje się w maszynie cyfrowej różne mechanizmy, które automatycznie zmieniają adresy względne na stałe lub też reorganizują pamięć operacyjną.

7.1.1.1. Reorganizacja pamięci

Jest to najprymitywniejsza metoda polegająca na tym, że gdy dany program (o stałych adresach) zostaje pobrany z pamięci zewnętrznej do wykonania, jest wprowadzany do określonego (stałym adresem) pola, a programy znajdujące się w tym polu zostają przepisane albo do innego pola, albo do pamięci zewnętrznej. Taki system manipulowania programami powoduje ogromne straty.

7.1.1.2. Przeadresowanie

Zakłada się tutaj, że rozpatrywany program jest tak przygotowany, że wykonanie go nie zależy od miejsca pamięci operacyjnej, w którym zostanie przechowany w czasie wykonania. Taki program nazywa się przemieszczalny i zawiera wyłącznie adresy względne (np. od zera). Jest on ładowany do pamięci operacyjnej przez odpowiednią część systemu operacyjnego maszyny do dowolnego wolnego obszaru pamięci operacyjnej. Natomiast zamiana względnych adresów (rozkazów i argumentów) na stałe adresy określające fizyczny obszar pamięci operacyjnej, do którego dany program został załadowany, odbywa się albo w czasie ładowania programu, albo w czasie wykonywania. W związku z tym mówi się o programowym lub o bieżącym (dynamicznym) przeadresowaniu. Metoda bieżącego przeadresowania ma tę przewagę nad metodą programowaną, że wykonywany program można przerwać i przenieść z jednego obszaru pamięci operacyjnej do drugiego i dalej kontynuować wykonanie tego programu.

7.1.1.3. Przeadresowanie dynamiczne z jednym rejestrem

W przypadku, gdy dany program znajduje się we wspólnym obszarze pamięci operacyjnej, do wyznaczenia początkowego adresu bezwzględnego tego obszaru wystarczy jeden techniczny rejestr. Zawartość tego rejestru (np. DATUM w maszynie ODRA 1300) zostaje dodana do adresów względnych programu w czasie jego wykonywania.

7.1.1.4. Przeadresowanie dynamiczne z kilkoma rejestrami

Często w maszynie cyfrowej dopuszcza się przypadki, w których wykonywany program zostaje ulokowany w kilku rozdzielnych obszarach pamięci operacyjnej (np. program o długości 1000 słów znajduje się w obszarach 2000—2499 i 7000—7499). W takich przypadkach wprowadza się tyle rejestrów typu DATUM, ile obszarów rozdzielnych (segmentów) zawiera dany program. Przy takim rozwiązaniu trudność polega na tym, aby w czasie dynamicznego przeadresowania wybierać właściwy rejestr DATUM. Problem rozwiązuje się na ogół łącznie z problemem protekcji pamięci operacyjnej. Wówczas każdy rejestr DATUM składa się z 3 części:

— w pierwszej części pamięta się początkowy adres względny danego segmentu programu (np. 0 lub 500),
— w drugiej części pamięta się końcowy adres względny danego segmentu programu (np. 499 lub 999),
— w trzeciej części — wielkość przeadresowującą adresy programu (np. 2000 lub 7000).

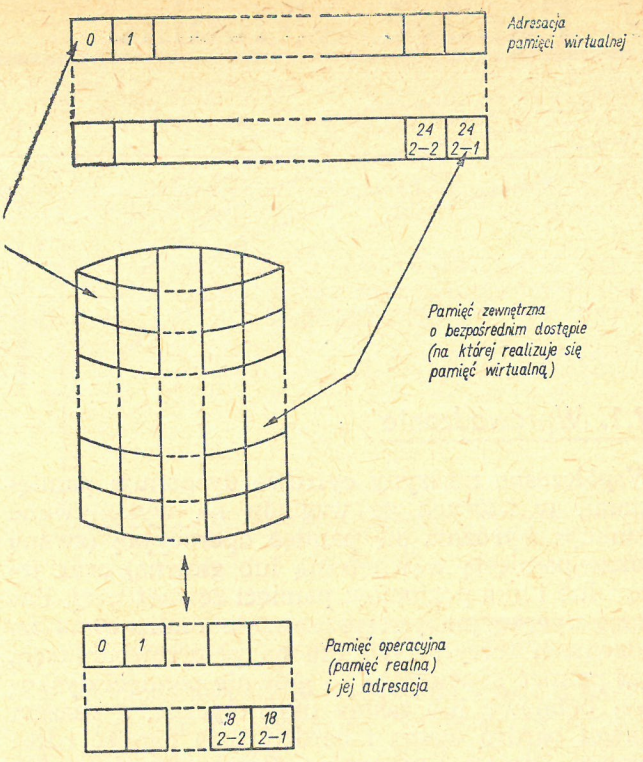
Zatem wielkość adresu względnego określa rejestr DATUM, przy czym procedura wyboru rejestru DATUM jest tutaj dłuższa i zależy ogólnie od ilości segmentów programu.

7.1.2. Stronicowa organizacja pamięci (paginacja)

We współczesnych maszynach cyfrowych istnieje tendencja dzielenia programu na nieduże części, przy czym wielkość tych części jest jednakowa dla wszystkich programów. Innymi słowy, dzieli się całą pamięć operacyjną na równe bloki zawierające po N słów (lub bajtów czy liter). Bloki te nazywa się stronami. Wielkość strony jest różna w różnych maszynach cyfrowych (np. 512, 1024, 2048, 4096), a w niektórych maszynach Jednolitego Systemu istnieje możliwość wyboru długości stron (np. 2048 lub 4096).

7.1.2.1. Programy użytkowe w systemie o stronicowej organizacji pamięci

Jak już powiedziano, pamięć operacyjną dzieli się na równe strony. Na takie same strony dzieli się również pamięć zewnętrzną (tj. pamięci dyskowe, bębnowe i inne typy pamięci o bezpośrednim dostępie), przy czym każdej stronie przyporządkowuje się określony numer (rys. 7.1). W ten sposób



Rys. 7.1. Ogólna koncepcja pamięci wirtualnej

ma się do czynienia z polem pamięci o różnorodnej strukturze z technicznego punktu widzenia i o jednorodnej strukturze z punktu widzenia użytkownika. Pole to nazywa się wirtualną lub matematyczną pamięcią systemu.

W tych systemach użytkownik dysponuje wielokrotnie większą pamięcią niż pamięć operacyjna i jest ograniczony jedynie możliwościami adresacji danej maszyny (np. długością efektywnego adresu instrukcji). Zatem pamięć wirtualna to pamięć użytkownika i dlatego jej adresy nazywa się adresami użytkownika (zwane również wirtualnymi lub matematycznymi), a programy użytkowe stosują właśnie takie adresy. Po wprowadzeniu programu użytkowego do maszyny cyfrowej z pamięcią wirtualną użytkownik nie zna miejsca przechowania poszczególnych części swego programu.

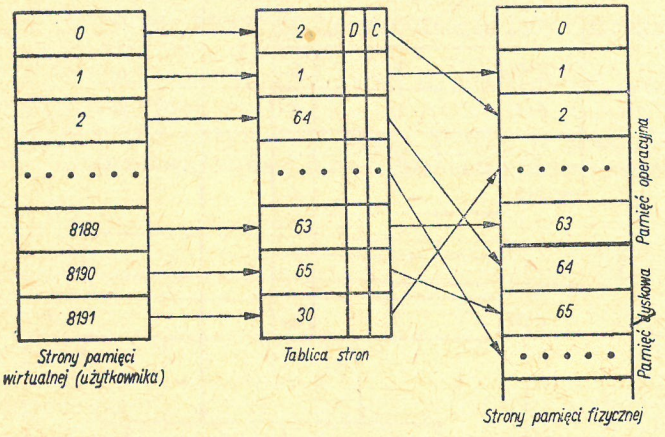
Sterowaniem i wykorzystaniem różnych poziomów pamięci zajmuje się system operacyjny maszyny. System operacyjny zajmuje się również podziałem programów na strony oraz ich rozmieszczeniem w pamięci w taki sposób, aby optymalizować pracę całego systemu (na przykład kilka stron danego programu może znajdować się w pamięci operacyjnej, a pozostałe strony tego programu znajdują się w pamięci zewnętrznej). Natomiast z punktu widzenia użytkownika program wraz z danymi stanowi zwarty zbiór w jednopoziomowej pamięci (w pamięci wirtualnej) i w tej właśnie postaci jest wykonywany.

Zatem systemy z pamięcią wirtualną mają następujące zalety:

— kolejne strony użytkownika (pamięci wirtualnej) nie muszą zajmować kolejnych stron pamięci operacyjnej;
— w pamięci operacyjnej nie musi znajdować się cały program użytkownika, lecz tylko te strony, które aktualnie są potrzebne;

— użytkownik ma jednolicie adresowany duży obszar pamięci dla programów i danych;
— system operacyjny organizuje podział i wykorzystanie pamięci w sposób dynamiczny, tj. w procesie wykonywania programów.

W związku z tym, że w czasie wykonania programu tylko jego część (tj. strony aktualnie przetwarzane) znajduje się w pamięci operacyjnej, a pozostała część znajduje się w pamięci zewnętrznej oraz że znajdujące się w pamięci operacyjnej (lub zewnętrznej) strony mogą być rozmieszczone w dowolnych i niespójnych obszarach — powstaje potrzeba zorganizowania tablicy opisującej przyporządkowanie adresów rzeczywistych, tj. adresów fizycznych w pamięci (operacyjnej lub zewnętrznej), poszczególnym stronom jednolitej (wirtualnej) pamięci użytkownika. Tablica ta nazywa się tablicą stron i zorganizowana zostaje przez system operacyjny. Przechowuje się w niej także różne dodatkowe informacje (np. o dostępności danej strony w obszarze pamięci operacyjnej, o stopniu aktywności danej strony itp.).



Rys. 7.2. Przykład tablicy stron

Rysunek 7.2. podaje przykład tablicy stron, tj. przyporządkowanie adresów fizycznych adresom wirtualnym stron programu użytkowego. W tym przykładzie pamięć wirtualna użytkownika ma 8192 strony, pamięć operacyjna 64 strony, a zewnętrzna pamięć dyskowa ponad 40 000 stron (wielkości *D* i *C* zawierają dodatkowe informacje związane z daną wirtualną stroną).

7.1.2.2. Układy związane ze stronicową organizacją pamięci

Każdy rozkaz lub argument programu użytkownika ma w rzeczywistości dwa adresy:
— adres wstawiony w programie w czasie translacji, tzn. adres wirtualny, który może przyjmować dowolną wartość z dziedziny adresów użytkownika;
— adres rzeczywisty, określający fizycznie miejsce, pod którym znajduje się dany rozkaz lub argument, przy czym adres ten nie musi być adresem pamięci operacyjnej.

Rolą układów realizujących funkcje stronicowej organizacji pamięci (tj. paginacji) jest zapewnienie właściwego wykonania programu, tzn. zamiana adresów wirtualnych na adresy rzeczywiste. W przypadku gdy adres rzeczywisty dotyczy pamięci zewnętrznej, spowodowane zostaje odpowiednie przerwanie w celu ściągnięcia z pamięci zewnętrznej do pamięci operacyjnej odpowiedniej strony zawierającej adresowany rozkaz lub argument.

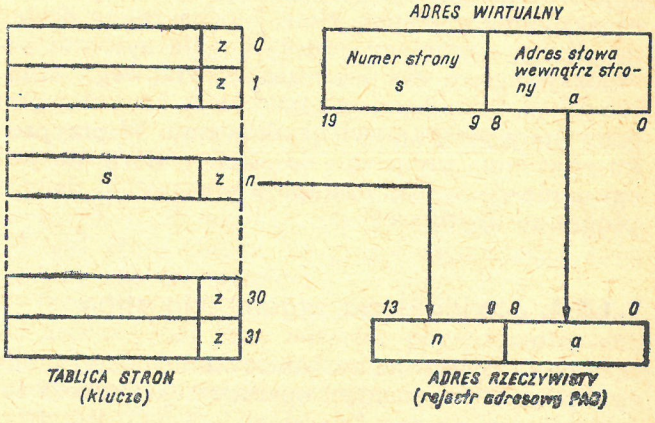
Zamiana adresów wirtualnych na rzeczywiste odbywa się, jak już wspomniano, za pomocą specjalnych tablic stron sporządzonych i aktualizowanych przez system operacyjny. Tablice te zawierają informacje o rzeczywistych adresach programu bieżącego.

Poniżej podano opis niektórych metod stronicowej organizacji pamięci, różniących się głównie strukturą tablic stron i sposobem zamiany adresów.

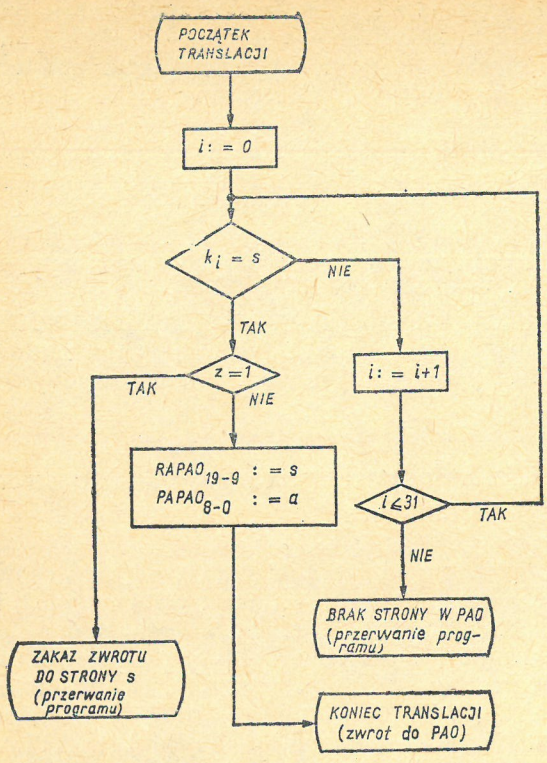
7.1.2.3. Jednopoziomowa organizacja stron pamięci wirtualnej

Jednopoziomowa organizacja stron, zastosowana w projekcie angielskiej maszyny ATLAS [1], była jedną z pierwszych. Polega ona na tym, że każdej stronie pamięci operacyjnej przyporządkowuje się pewien wewnętrzny rejestr zwany kluczem, zawierający adres wirtualny strony przechowanej w danym bloku pamięci operacyjnej (w ATLASIE były 32 takie bloki o pojemności 512 słów każdy) oraz informację zakazującą lub zezwalającą na kontakt programu bieżącego z danym blokiem pamięci operacyjnej. Pamięcią zewnętrzną w ATLASIE było kilka bębnów o pojemności 96 Ksłów każdy, o łącznej pojemności 1024 Ksłów. Natomiast pamięć operacyjna miała pojemność 16 Ksłów, czyli mogła przechowywać 32 strony.

Mechanizm translacji adresów (rys. 7.3 i 7.4). Adresy wirtualne występujące w programie użytkowym (tj. adresy rozkazów lub argumentów operacji) są 20-bitowe, przy czym 11 najbardziej znaczących bitów określa adres strony, zaś pozostałe 9 bitów określa adres wewnątrz danej strony. W czasie wykonania programu adres strony jest kolejno porównywany z 11 lewymi bitami klucza z tablicy stron. Mogą przy tym mieć miejsce dwa przypadki:
— zachodzi zgodność z jednym kluczem tablicy stron; wówczas przekształca się adres wirtualny programu na adres rzeczywisty (pamięci operacyjnej) w sposób pokazany na rys. 7.3, tzn. przepisuje się 9 prawych bitów z adresu wirtualnego na rzeczywisty, a za adres strony



Rys. 7.3. Jednopoziomowa organizacja stronic (ATLAS)



Rys. 7.4. Schemat czynnościowy translacji adresów wirtualnych na rzeczywiste adresy PAO (ATLAS)
i — wskaźnik bieżący (0, 1, ..., 31), k_i — zawartość *i*-go klucza tablicy stron, *s* — starsza część (numer strony) adresu wirtualnego, RAPAO — rejestr adresowy pamięci operacyjnej, *z* — wskaźnik zakazu zwrotu do danej strony (1 bit)

rzeczywistej przyjmuje się numer klucza, przy którym zaszła zgodność;

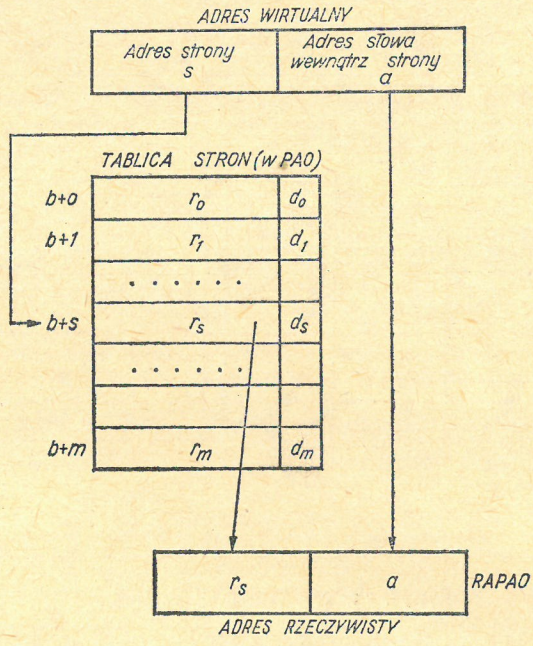
— brak zgodności; oznacza to, że klucz tablicy stron nie zawiera szukanego numeru strony, czyli dana strona nie znajduje się w pamięci operacyjnej; wówczas przerywa się program bieżący i następuje przejście do systemu operacyjnego, który zajmie się sprowadzeniem wymaganej strony z pamięci zewnętrznej do pamięci operacyjnej.

Należy zwrócić uwagę, że sekwencyjne porównanie numeru strony programu z zawartością tablicy stron jest procesem czasochłonnym, dlatego w praktyce stosuje się rejestry i układy z natychmiastowym dostępem. Ochrona pamięci jest tu zorganizowana z dokładnością do strony dzięki wprowadzeniu do każdego klucza dodatkowego bitu z (wskaźnika zakazu). Stan tego bitu wskazuje, czy informacje umieszczone w danej stronie są aktualnie dostępne. Stan 1 oznacza, że dostęp do strony dozwolony jest tylko dla systemu zarządzającego, a dla programu użytkowego oznacza zakaz. Stan 0 oznacza natomiast, że dostęp jest dozwolony programowi użytkowemu. W przypadku odwołania programu do strony, która nie jest aktualnie dostępna, następuje przerwanie danego programu użytkowego.

7.1.2.4. Standardowa organizacja stron

Metoda ta polega na zorganizowaniu pełnej tablicy stron (TS) w pamięci operacyjnej. Oznacza to, że tablica stron ma tyle słów (opisów), ile stron ma cała pamięć wirtualna.

Jeśli dla przykładu weźmiemy pamięć wirtualną o pojemności 16384 Kbajtów (wówczas oczywiście adres bajtu jest 24-bitowy), gdzie przyjęto wielkość strony 2 Kbajtów, to ilość stron wynosi 8192. Zatem tablica stron jest duża i składa się z 8192 słów. Początek tablicy stron (który jest na ogół dowolny) załadowany zostaje przez system operacyjny do specjalnego rejestru *B*, zwanego bazą tablicy stron. Tablica stron jest tak zorganizowana, że w kolejnych słowach tablicy przechowuje się rzeczywiste adresy (*r*) kolejnych stron pamięci wirtualnej oraz informacje (*d*) o dostępności stron w pamięci operacyjnej. Umożliwia to prostą translację adresów wirtualnych na adresy rzeczywiste (fizyczne) pamięci operacyjnej. Jeśli mianowicie numer strony adresu wirtualnego zostanie oznaczony przez *s* (patrz rys. 7.5), to słowo



Rys. 7.5. Schemat translacji adresów wirtualnych za pomocą pełnej tablicy stron ulokowanej w PAO
b — zawartość rejestru bazy, *s* — adres bieżący strony wirtualnej ($0 \leqslant s \leqslant m$), r_s — adres rzeczywisty strony o adresie wirtualnym *s*, *ds* — wskaźnik dostępności w PAO strony o adresie wirtualnym *s*

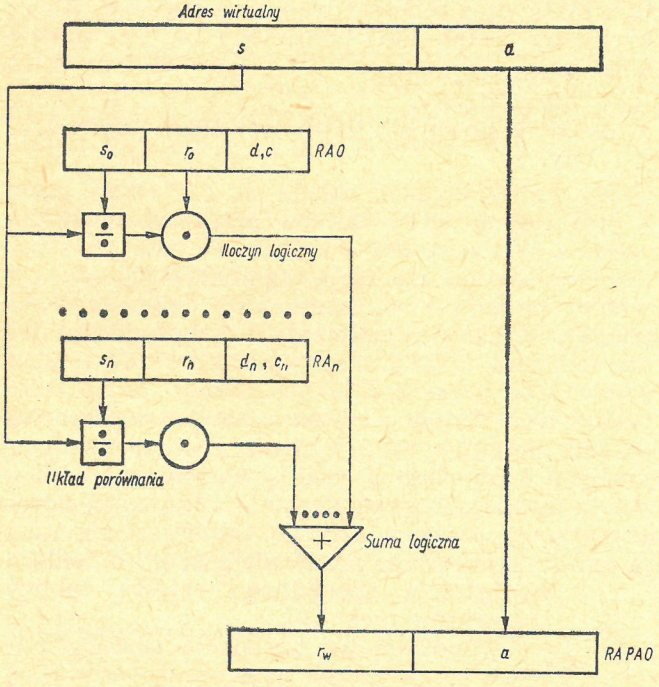
o adresie $b+s$ w tablicy TS zawiera adres rzeczywisty strony (tj. adres r_s). Adres r_s łącznie z adresem *a* i informacją *d* określają jednoznacznie adres rzeczywisty szukanej informacji w PAO. Informacja *d* określa dostępność danej strony. Jeśli np. $d=0$, oznacza to, że w danym momencie zwroty do strony *s* są niedozwolone. Próba zwrotu do takiej strony powoduje przerwanie programu i przejście do tej części systemu operacyjnego, która zajmuje się wymianą stron. Jeśli natomiast $d=1$, to tworzy się adres rzeczywisty komórki (z części r_s i *a*) w rejestrze RAPAO, po czym następuje zwrot do tej komórki PAO.

Opisany sposób translacji adresów wirtualnych jest bardzo prosty w realizacji technicznej, lecz wydłuża czas wykonania instrukcji. Każdy bowiem zwrot do PAO (po instrukcję lub argument operacji) wymaga dodatkowego zwrotu do PAO (np. rzędu 1500 nanosekund) w celu przeczytania odpowiedniego słowa z tablicy stron. W ten sposób czas wykonania instrukcji wydłuża się średnio dwa razy.

7.1.2.5. Translacja adresów z zastosowaniem pamięci asocjacyjnej

Straty czasowe związane z translacją adresów można istotnie zmniejszyć, jeśli zastosuje się szybką pamięć buforową (np. rzędu 100 ns) do przechowywania pełnego stanu tablicy stron. Oczywiście pełne szybkie pamięci buforowe są drogie. Dlatego we współczesnych maszynach cyfrowych stosuje się małe zestawy bardzo szybkich rejestrów, zwanych często pamięcią asocjacyjną (ze względu na kojarzeniowy sposób wyboru informacji, co oznacza, że wybór rejestru odbywa się na podstawie jego zawartości, a nie na podstawie adresu) do przechowywania adresów rzeczywistych nie wszystkich stron znajdujących się w pamięci operacyjnej, lecz tylko adresów rzeczywistych (i innych opisów) „bardziej aktywnych" stron. Pamięci asocjacyjne lub bufory adresów translacji (BAT) składają się przeważnie z małej ilości bardzo szybkich rejestrów, rzędu 8÷128 rejestrów, w których przechowuje się (rys. 7.6):

s — adres wirtualny strony „aktywnej",
r — adres rzeczywisty odpowiadający stronie o adresie wirtualnym s,
d — cechę dostępności danej strony pamięci operacyjnej,
c — cechę aktywności danej strony w procesie przetwarzania i inne informacje pomocnicze.

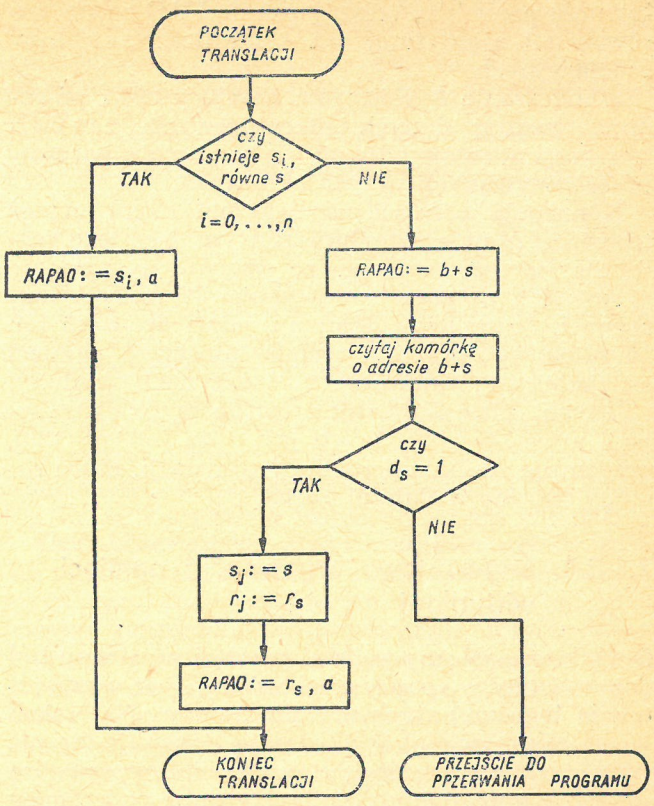


Rys. 7.6. Schemat strukturalny translacji adresów wirtualnych za pomocą rejestrów asocjacyjnych
s — adres wirtualny strony, r — adres rzeczywisty strony r_w — adres rzeczywisty rejestru RA, w którym znajduje się adres wirtualny s_i równy s (gdzie i=0, ..., n)

Informacje s, r, d i c związane z jedną stroną są na ogół przechowywane w jednym rejestrze pamięci asocjacyjnej (lub BAT). Oczywiście wielkość powyższych parametrów oraz sposób ich wykorzystania jest różny w różnych maszynach cyfrowych.

Mechanizm translacji adresów wirtualnych, w systemach wyposażonych w pamięć asocjacyjną, może działać w niżej opisany sposób (rys. 7.6 i 7.7). W celu odwołania się do słowa pamięci wirtualnej o adresie strony s i adresie słowa a (wewnątrz strony s), porównuje się jednocześnie zawartość odpowiedniej części wszystkich rejestrów asocjacyjnych (tj. części s_0, s_1 ..., s_n; gdzie $n+1$ jest ilością rejestrów asocjacyjnych) z adresem bieżącym s danej strony. Porównanie to zachodzi jednocześnie w n niezależnych układach porównania. Jeśli w którymś układzie porównania zajdzie zgodność adresów, to układ ten generuje odpowiedni sygnał pozwalający przepisać adres rzeczywisty (r_w) tego rejestru asocjacyjnego do starszej części rejestru adresowego pamięci operacyjnej (RAPAO). Natomiast adres słowa wewnątrz danej strony (a) zostaje przepisany bez zmian z adresu wirtualnego do rejestru RAPAO, tworząc w ten sposób pełny adres szukanego słowa w PAO. Jeśli natomiast w żadnym układzie porównania nie zajdzie zgodność adresów, oznacza to, że aktualny adres rzeczywisty nie znajdzie się w pamięci asocjacyjnej. W tym przypadku sięga się do pełnej tablicy stron znajdującej się w pamięci operacyjnej, z której czyta się informacje r, d i c związane z daną stroną s. Informacje te zapisuje się do któregoś z rejestrów asocjacyjnych (RA) według odpowiedniego algorytmu. Dzięki temu osiąga się pełny dynamiczny przydział i ochronę pamięci.

Stosuje się różne algorytmy usuwania z pamięci asocjacyjnej pary adresów przyporządkowania danej stronie i wprowadzenia w to miejsce nowej pary adresów żądanej strony, np. strony najmniej aktywnej lub strony, której para adresów najdłużej znajduje się w rejestrze asocjacyjnym. Efek-



Rys. 7.7. Schemat czynnościowy translacji adresów wirtualnych za pomocą rejestrów asocjacyjnych
s — bieżący adres wirtualny strony, s_i — adres wirtualny zapisany w i-tym rejestrze asocjacyjnym, d_s=1 oznacza dostępność strony s (w PAO), j — numer rejestru asocjacyjnego najmniej „aktywnego" (j=0, ..., n)

tywność procedury odwoływań do PAO zależy zarówno od algorytmu, jak i od ilości rejestrów asocjacyjnych.

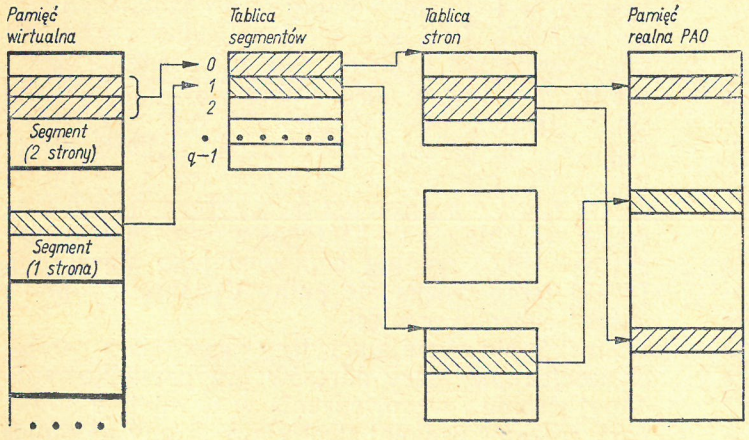
Wbudowanie 8 rejestrów asocjacyjnych powoduje, według danych literaturowych, że 90% translacji adresów odbywa się za pomocą adresów przechowywanych w tych rejestrach.

Koncepcja pamięci asocjacyjnej czyni proces translacji prawie natychmiastowym, lecz jego realizacja jest bardzo droga ze względu na istnienie n szybkich układów porównania. W małych maszynach Jednolitego Systemu, gdzie np. $n=8$ lub 16 rejestrów, przyjęta została koncepcja pamięci asocjacyjnej. Natomiast w maszynach średnich i dużych, gdzie ilość rejestrów do szybkiej translacji adresów wirtualnych jest duża (np. $n=128$), przyjmuje się inne procedury translacji, które zostaną opisane w dalszej części.

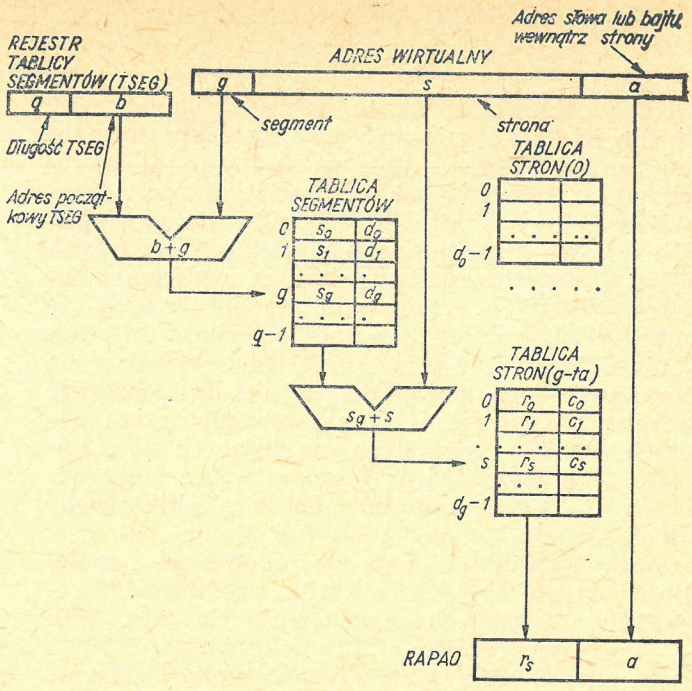
7.1.2.6. Segmentowa organizacja pamięci wirtualnej

Dotychczas opisany podział pamięci na strony był dokonywany w sposób mechaniczny, bez uwzględnienia logicznej struktury programów. Natomiast w metodzie zwanej segmentacją dokonuje się nie tylko podziału pamięci, lecz również wprowadza się możliwość dokonywania podziału programów według ich struktury wewnętrznej. Programista lub system kompilujący dzieli program i dane na części zwane segmentami. Każdy segment ma własne wejście, a każdy program dysponuje pewną ilością odrębnych segmentów. Długość segmentu bywa na ogół dowolnie określana. Segment może oznaczać zbiór danych, które można tylko czytać; czystą procedurę programową, której stanu nie można zmieniać (wówczas taka procedura może być wykorzystywana jednocześnie przez kilka programów) itp.

W metodzie segmentacji pamięć zostaje podzielona na q zadań przetwarzanych jednocześnie w systemie wieloprogramowym. Każde takie zadanie (segment) zajmuje na ogół kilka kolejnych stron pamięci wirtualnej (czyli o ciągłej adresacji). Segmentom zatem można przyporządkować adresy $0,1,2,\ldots,g,\ldots,q-1$, gdzie g jest numerem bieżącym. Podział pamięci wirtualnej na segmenty oraz jej powiązanie z pamięcią rzeczywistą (PAO) poprzez dwupoziomową tablicę (tj. tablicę segmentów i tablicę stron) pokazano na rys. 7.8.



Rys. 7.8. Podział pamięci na segmenty i dwupoziomowa tablica przyporządkowania stronom adresów realnych



Rys. 7.9. Schemat dwupoziomowej translacji adresów
s_g — adres początkowy g-tej tablicy stron, d_g — długość g-tej tablicy stron, r_s — adres rzeczywisty strony o wirtualnym adresie s, c_s — cechy sterujące strony s

Natomiast rys. 7.9 ilustruje przykładową strukturę tablic oraz schemat translacji adresów. Adres wirtualny został tutaj podzielony na trzy części:

q — adres segmentu,
s — adres strony,
a — adres słowa lub bajtu wewnątrz danej strony.

Zatem każdy segment składa się z 2^k stron, gdzie k jest ilością bitów adresu przeznaczonych dla części s. W ten sposób wyróżnia się tutaj dwa poziomy adresacji: adresacja segmentów (zadań) wewnątrz pamięci i adresacja stron wewnątrz segmentu. Z każdym segmentem związany jest jeden element tzw. tablicy segmentów. W tablicy segmentów przechowuje się adresy początkowe tablic stron. Zatem element s_g tablicy segmentów określa początek tablicy stron zawartych w segmencie o numerze g, zaś d_g długość tej tablicy stron, czyli ilość stron segmentu. Również adres początkowy (baza) i długość tablicy segmentów są dowolnie określone przez wielkość q i b, będące zawartością specjalnego rejestru tablicy segmentów (rys. 7.9).

Dwupoziomowa translacja adresów odbywa się w sposób następujący: czyta się zawartość słowa o adresie $b+g$ (tj. g-ty element tablicy segmentów), w którym zarejestrowany jest bazowy adres (s_g) tablicy stron oraz jej długość (d_g). Z kolei suma s_g+s określa element (wejście) tablicy stron, w którym znajduje się rzeczywisty adres (r_s) strony w PAO.

Parametry q i d, tj. długość tablicy segmentów i długość tablicy stron, stosuje się do kontroli poprawności bieżących wielkości g i s adresu wirtualnego, czyli sprawdza się, czy zachodzą relacje: $g<q$ oraz $s<d$. W ten sposób zapewnia się ochronę pamięci przed niepowołanymi kontaktami.

7.1.3. Procedury transmisji stron

W maszynach z pamięcią wirtualną mała część informacji programu znajduje się w pamięci operacyjnej (PAO), zaś pozostała część mieści się w pamięci zewnętrznej (PAZ). Jeśli w czasie pracy danego programu żąda on zwrotu do informacji znajdującej się w PAZ, to układy maszyny powodują przerwanie tego programu i przejście do systemu operacyjnego, tj. do procedury zajmującej się wymianą stron pomiędzy PAO i PAZ. Jeśli w PAO istnieje aktualnie wolny obszar na jedną stronę, to system operacyjny organizuje czytanie PAZ i sprowadza żądaną stronę z PAZ do PAO. Jednocześnie system aktualizuje zawartość odpowiedniej tablicy stron, tzn. wpisuje do tej tablicy adres rzeczywisty sprowadzonej strony oraz ustawia jej cechę dostępności. Jeśli natomiast nie ma wolnego obszaru w PAO, to programowa procedura wymiany stron musi najpierw usunąć z PAO określoną stronę, aby zwolnić miejsce dla nowej żądanej strony. Wybór strony do usunięcia z PAO musi być dokonywany według rozsądnej strategii w celu zminimalizowania ilości transmisji stron z PAZ do PAO i na odwrót. Oczywiście idealna strategia byłaby następująca: usuwa się z PAO tę stronę, do której odwołania będą miały miejsce później niż do dowolnej innej strony. Niemniej strategia ta wymaga znajomości kolejności użycia stron w przyszłości, lecz w praktyce w niewielu zadaniach można określić taką kolejność. Dlatego przy usuwaniu stron z PAO stosuje się metody, które wykorzystują informacje opisujące dotychczasowy przebieg wykorzystania stron oraz ekstrapolację tego przebiegu. Poniżej opisano pokrótce niektóre z metod usuwania stron z PAO.

1. METODA NAJDŁUŻSZEGO PRZEBYWANIA W PAO

W tej metodzie, znanej również pod nazwą FIFO (*First In-First Out*), usuwa się stronę, która dłużej przebywa w PAO niż pozostałe. Metoda wynika z założenia, że program wykonuje swoje instrukcje w sposób sekwencyjny, zatem prawdopodobieństwo użycia strony, która najdłużej znajduje się w PAO, jest małe. Oczywiście metoda ta może spowodować w określonych zadaniach usunięcie strony, która jest najczęściej lub nawet stale wykorzystywana i dla takich zadań metoda FIFO nie jest optymalna.

2. METODA WSKAŹNIKÓW NIEWYKORZYSTANIA STRON

W tej metodzie usuwa się stronę, do której nie było ani jednego odwołania w określonym przedziale czasu. W przypadku istnienia kilku stron spełniających ten warunek, bierze się również pod uwagę czasy przebywania stron w PAO. W celu dynamicznej rejestracji odwoływań wprowadza się specjalne wskaźniki (cechy) techniczne, zwane bitami odwoływań lub bitami referencyjnymi. Na ogół jedna strona posiada jeden taki wskaźnik (bit), który zostaje ustawiony w stan 1, jeśli odwołanie do tej strony PAO miało miejsce (początkowy stan tego wskaźnika zostaje wyzerowany). Jeśli wskaźniki wszystkich stron w PAO są jedynkami, to ponownie ustawia się wszystkie wskaźniki w stan 0 (np. specjalnym sygnałem w pamięci asocjacyjnej). Zatem w czasie wymiany stron usuwa się z PAO tę stronę, której bit referencyjny równa się 0. Metoda ta znana jest również pod nazwą LRU (*Least Recently Used page*).

3. METODA NAJDŁUŻSZEJ BEZCZYNNOŚCI

W tej metodzie usuwa się stronę, do której ostatnie odwołanie miało miejsce wcześniej niż odwołanie do dowolnej innej strony. Jeśli istnieje kilka stron mających ten sam czas bezczynności, to usuwa się na przykład stronę o najmniejszym adresie wirtualnym (gdyż na ogół proces przetwarzania ma charakter sekwencyjny i to w kierunku wzrastających adresów). W celu rejestracji czasu bezczynności stron znajdujących się w PAO wprowadza się w systemie operacyjnym zmienne $T_0, T_1, \ldots, T_n$, gdzie n jest ilością stron w PAO.

Czas bezczynności stron można mierzyć różnie, na przykład na podstawie stanu bitów referencyjnych tych stron, co pewien czas t. Na początek danego przedziału czasu wszystkie bity referencyjne zostają wyzerowane, zaś po upływie czasu t następuje przerwanie (lub tylko „programowe" przerwanie zapoczątkowane przerwaniem zegara czasu realnego) i przeanalizowanie stanu bitów referencyjnych stron. Jeśli bit referencyjny j-tej strony jest 0, to $T_j := T_j - 1$; jeśli natomiast 1, to $T_j := 0$.

4. METODA PRAWDOPODOBIEŃSTWA AKTYWNOŚCI

W tej metodzie usuwa się stronę, która ma najmniejsze prawdopodobieństwo aktywności. Jeśli T oznacza czas pracy danego zadania, a t_i czas pracy (aktywności) i-tej strony tego zadania, to prawdopodobieństwo aktywności tej strony nazywa się $P_i = t_j/T$. Prawdopodobieństwa można na przykład określać w sposób przybliżony za pomocą bitów referencyjnych co określony przedział czasu. Analizując stan bitów referencyjnych na końcu każdego umownego przedziału czasu można przetasować kolejność stron w specjalnie skonstruowanej tablicy prawdopodobieństwa (priorytetów) stron. Na początek przyjmuje się dowolny porządek stron w tablicy prawdopodobieństw, a po każdym przeanalizowaniu bitów referencyjnych stron przesuwa się stronę z bitem 1 do przodu (w lewo), według przykładu ilustrowanego na poniższej tablicy, zaś strony z bitem 0 przesuwa się do tyłu.

Kolejny moment	Bity referencyjne stron 1 2 3 4 5	Kolejność (aktywność) stron
0	0 0 0 0 0	1 2 3 4 5
1	1 0 1 1 0	1 3 4 2 5
2	1 0 0 1 0	1 4 3 2 5
3	1 1 0 0 1	1 2 5 4 3
4	0 1 0 0 0	2 1 5 4 3
5	0 1 1 1 0	2 4 3 1 5
6	0 0 1 1 0	4 3 2 1 5

Strona, która jest ostatnia w kolejności powyższej tabeli, zostaje usunięta z PAO w przypadku żądania wymiany (np. po 4 momencie usunie się

stronę 3, zaś po 6 — stronę 5). Wadą powyższej metody jest to, że traci się dodatkowy czas na przetasowanie kolejności stron w powyższej tablicy.

5. METODA ZASTOSOWANA W MASZYNIE ATLAS

W maszynie ATLAS wprowadzono dwie wielkości opisujące czas bezczynności strony:

T_b — Czas bezczynności danej strony mierzony od ostatniego zwrotu do tej strony (tzw. bieżący czas bezczynności),

T_p — sumaryczny czas bezczynności tej strony w poprzedniej „kadencji" przebywania danej strony w PAO.

Czas T_b może być mierzony jak w metodzie 3, przy czym po przepisaniu strony do PAZ również kontynuuje się zliczanie czasu bezczynności. Natomiast po wezwaniu strony do PAO, przyjmuje się: $T_p := T_b$ oraz $T_b := 0$.

Strategia usuwania strony z PAO jest następująca:

a) usuwa się stronę, której wielkości T_p i T_b spełniają relację $T_b > T_p$, a jeśli jest więcej takich stron, to usuwa się stronę z maksymalną wartością różnicy $T_b - T_p$; kryterium to oparto na założeniu, że jest mało prawdopodobne użycie strony, której czas bieżącej bezczynności przekroczył całkowity czas bezczynności poprzedniej „kadencji";

b) jeśli nie ma stron spełniających warunek $T_b > T_p$, to usuwa się stronę mającą maksymalną wartość różnicy $T_p - T_b$ oraz $T_b = 0$.

Jeśli w wyżej opisanych przypadkach znajdzie się kilka stron z równą wartością różnicy, to wybiera się do usunięcia tę stronę spośród nich, która ma najmniejszy adres wirtualny.

Powyższa strategia uwzględnia zachowanie się strony w procesie przetwarzania przez dłuższy czas, co zapewnia większą wierność w prognozowaniu. Metoda ta daje zadowalające wyniki w cyklicznych procesach przetwarzania. Wadą jej jest stosunkowo pracochłonny proces ustalenia wartości różnic $T_p - T_b$ oraz wybór maksymalnej wartości.

Wymianę stron pomiędzy PAO i PAZ można istotnie zmniejszyć (niezależnie od metody wymiany), jeśli system operacyjny będzie posiadał dodatkowe informacje o typie strony, tzn. czy dana strona jest:

— stroną, której informacje tylko czyta się z PAZ (np. programy lub specjalne dane),

— stroną, której informacje mogą zmienić się w procesie przetwarzania (np. dane podlegające aktualizacji).

Jeśli bowiem strona jest typu pierwszego, nie zachodzi potrzeba przesyłania strony z PAO do PAZ, jeśli w PAZ przechowuje się przez cały czas jej kopię. Wówczas wystarczy tylko dokonać odpowiedniej korekty cechy dostępności w tablicy stron.

W niektórych maszynach Jednolitego Systemu informacja o tym, czy dana strona w PAO uległa zmianie czy nie, jest tworzona dynamicznie dzięki wprowadzeniu dodatkowego jednobitowego wskaźnika (tzw. bit zmiany) dla każdej ze stron. Zatem bit zmiany strony w PAO mówi o tym, czy jej kopia w PAZ jest aktualna.

7.1.4. Zalety i efektywność systemów z pamięcią wirtualną

Szybki rozwój licznych typów zastosowań w dziedzinie przetwarzania danych stawia coraz większe wymagania w zakresie swobody projektowania i programowania systemów EPD, tj. bez uwzględnienia różnych programowych, funkcjonalnych i technicznych ograniczeń. Jednym z istotnych ograniczeń występujących w poprzednich maszynach była wielkość pamięci roboczej przydzielona danemu programowi oraz sposób zarządzania tą pamięcią. Froste zwiększenie pojemności pamięci operacyjnej w danej maszynie, np. z 250 na 350 Kbajtów, nie rozwiązuje problemu radykalnie i optymalnie, albowiem wbudowanie do maszyny dodatkowego i kosztownego bloku 100 Kbajtów może okazać się nieopłacalne, jeśli będzie on wykorzystany tylko przez mały procent czasu lub zaledwie przez kilka programów. Poza tym zawsze można w praktyce spotkać zastosowanie, które także nie mieści się w pamięci o pojemności 300 Kbajtów. Również sposób wykorzystania pamięci nie powinien być krępowany techniczną realizacją różnych typów (poziomów) pamięci. W wielu zastosowaniach istnieje potrzeba operowania dużym polem dla programów i danych, przekraczającym wielkość fizyczną pamięci operacyjnej. Pole to powinno posiadać jednolitą adresację, co pozwala organizować program w sposób jednakowy i niezależny od tego, gdzie poszczególne części programów i danych znajdują się w procesie przetwarzania.

Powyższe wymagania zostały spełnione przez wprowadzenie koncepcji pamięci wirtualnej. Środki techniczne koncepcji pamięci wirtualnej oraz nowoczesne systemy operacyjne pozwoliły na stworzenie systemów komputerowych charakteryzujących się następującymi zaletami:

— dużą łatwością w programowaniu zadań EPD, co umożliwia zwiększenie wydajności programistów; programista może bardziej koncentrować się nad problemem samego zastosowania niż nad różnymi i złożonymi technikami programowania;

— dużą elastycznością w zastosowaniach, dzięki usunięciu zależności pomiędzy adresowalnym polem programu, a wielkością pamięci operacyjnej;

— lepszą efektywnością wykorzystania obszarów pamięci operacyjnej przez programy i lepszą łączną wydajnością systemu.

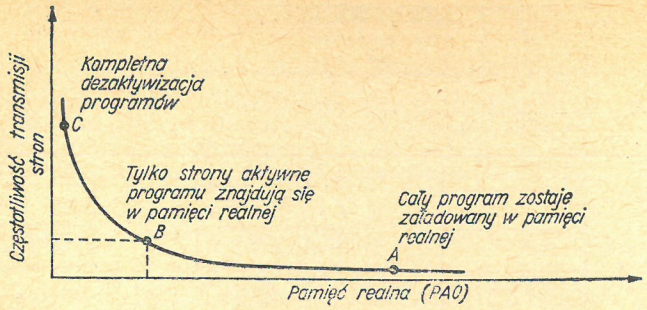
Efektywność maszyny z pamięcią wirtualną zależy od wielu czynników, na przykład od wielkości pamięci operacyjnej, od ilości transmisji stron z PAZ do PAO lub na odwrót (na ilość tych transmisji wpływa również rozmiar strony), od ilości i charakteru wykonywanych programów, od relacji szybkości procesora i pamięci zewnętrznej itp.

Właściwy dobór rozmiaru strony jest sprawą bardzo istotną, gdyż zwiększenie rozmiaru powoduje:

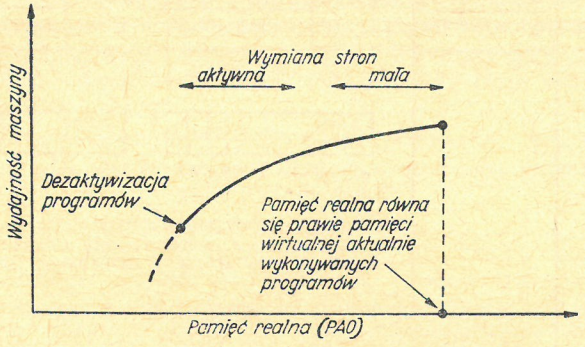
— zmniejszenie ilości transmisji między pamięcią operacyjną i zewnętrzną,

— zmniejszenie wymiaru tablic stron i czasu przeanalizowania ich wykorzystania w PAO,

— gorsze wykorzystanie pamięci operacyjnej przez programy (gdyż przydziela się progra-



Rys. 7.10. Zależność pomiędzy wielkością pamięci realnej a częstotliwością transmisji stron



Rys. 7.11. Wydajność maszyny w zależności od pojemności pamięci realnej i aktywności wymiany stron.

mom całkowitą liczbę stron zaokrągloną w górę, niezależnie od niewykorzystanych „resztek").

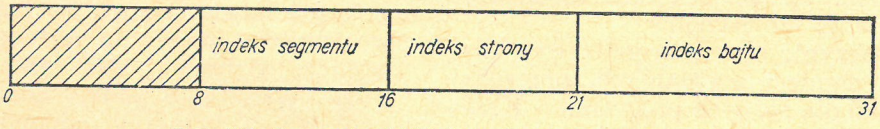
Ogólną zależność pomiędzy wielkością pamięci operacyjnej (liczoną w stronach) i częstotliwością transmisji stron przedstawia rys. 7.10, zaś rys. 7.11 — wydajność systemu. Punkt *A* wykresu określa taką wielkość pamięci realnej, przy której większość lub wszystkie strony aktualnie wykonywanych programów mieszczą się w niej, w dowolnym przedziale czasowym. Czas przebiegu programu w takiej pamięci jest minimalny, gdyż praktycznie transmisja stron ogranicza się tylko do czytania stron, od czasu do czasu przechodząc z jednej do drugiej fazy logicznej danego programu. W punkcie *B* wykresu pamięć operacyjna istotnie zmalała, lecz częstotliwość wymiany stron nieznacznie zwiększyła się. Natomiast po przekroczeniu tego punktu gwałtownie rośnie częstotliwość wymiany stron, która w punkcie *C* doprowadza do kompletnej dezaktywizacji programów z powodu bardzo małej pamięci operacyjnej. W takich przypadkach sprowadza się na przykład stronę z instrukcjami i wykonuje się tylko jeden jej rozkaz, który to rozkaz zapoczątkuje sprowadzenie strony z danymi usuwając stronę z instrukcjami. Zatem pamięć operacyjna nie może być zbyt mała. Wielkość otoczenia punktu *B* można przyjąć za optymalną wielkość pamięci operacyjnej. Wielkość ta, różna dla różnych maszyn i różnych zestawów wykonywanych programów, jest wyznaczana metodami symulacji.

7.2. Organizacja pamięci wirtualnej EMC JS

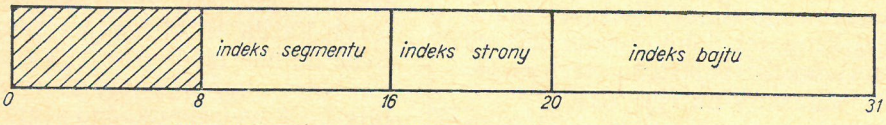
W organizacji logicznej nowszych wersji maszyn cyfrowych Jednolitego Systemu wprowadzono środki do realizacji koncepcji pamięci wirtualnej. W tej części pracy opisano pokrótce przyjęte środki i rozwiązania.

7.2.1. Dynamiczna translacja adresów

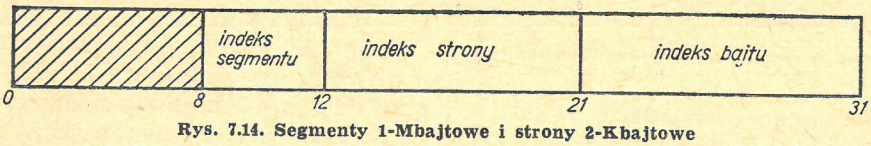
Środki dynamicznej translacji adresów umożliwiają, jak już wiadomo z części 7.1, techniczną zamianę adresów wirtualnych na adresy realne (rzeczywiste) w czasie wykonywania danego programu. Zamiana (translacja) adresów przebiega zgodnie z wcześniej przygotowanymi tablicami segmentów, stron i innych, w których to tablicach zapisywane jest pełne przyporządkowanie stron pamięci wirtualnej programów stronom (obszarom) pamięci fizycznej (operacyjnej lub zewnętrznej).



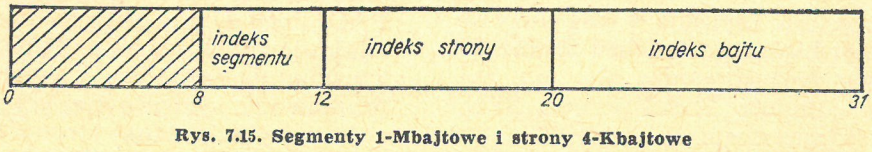
Rys. 7.12. Segmenty 64-Kbajtowe i strony 2-Kbajtowe



Rys. 7.13. Segmenty 64-Kbajtowe i strony 4-Kbajtowe



Rys. 7.14. Segmenty 1-Mbajtowe i strony 2-Kbajtowe



Rys. 7.15. Segmenty 1-Mbajtowe i strony 4-Kbajtowe

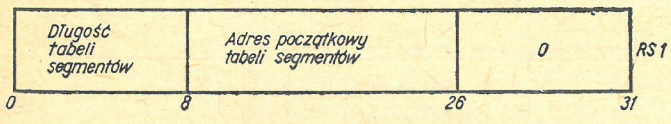
Struktura 24-bitowych adresów wirtualnych (logicznych) maszyn Jednolitego Systemu jest pokazana na rysunkach 7.12 do 7.15. Segment — to obszar programu mający kolejne wirtualne adresy. Pojemność segmentu może być dwóch typów: 64 Kbajtów lub 1024 Kbajtów. Indeks (lub adres) segmentu jest wielokrotnością jego rozmiaru. W czasie pracy programu wymiar jego segmentów jest określany przez bit 11 zerowego rejestru sterującego (tj. RSO). Strona jest ciągłym obszarem pamięci, a jej rozmiar wynosi 2 K lub 4 Kbajtów, zależnie od stanu bitów 8 i 9 rejestru RSO. Jak pokazują rysunki 7.12 do 7.15, adres wirtualny składa się z pola indeksu segmentu, pola indeksu strony i pola indeksu bajtu. Wielkości tych pól zależą oczywiście od rozmiaru segmentu i rozmiaru strony.

Poniższa tabela ilustruje rozmiar (w Kbajtach) oraz ilość segmentów i stron jednej 16-Mbajtowej pamięci wirtualnej (w systemie bowiem może jednocześnie istnieć kilka pamięci wirtualnych, każda o pojemności po 16 Mb).

Rozmiar segmentu	Ilość segmentów	Rozmiar strony	Ilość stron w segmencie	Bity 8 i 9 rej. RSO	Bit 11 rej. RSO
1024	16	2	512	01	1
1024	16	4	256	10	1
64	256	2	32	01	0
64	256	4	16	10	0

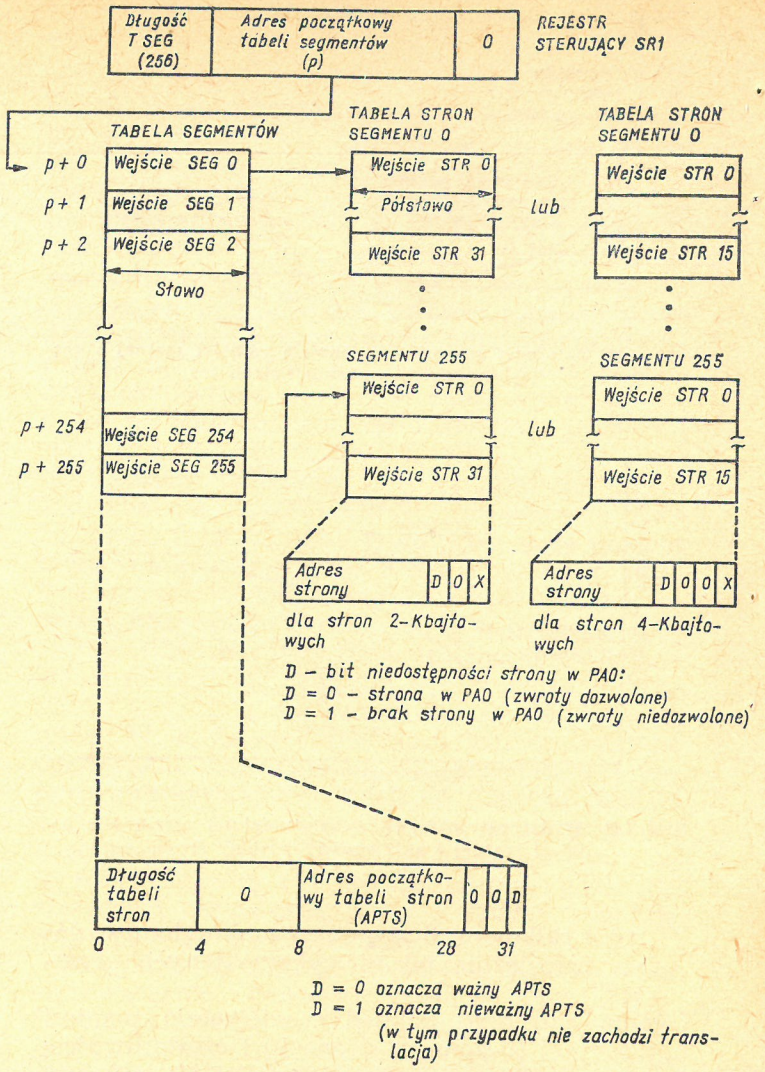
Przebieg procesu translacji adresów jest sterowany słowem stanu programu (SSP) oraz zawartością rejestrów RSO i RS1. Maszyny, w których wbudowano środki dynamicznej translacji adresów, niekoniecznie muszą pracować w reżimie TRANSLACJI. Mogą one pracować również w tradycyjny sposób, tzn. otrzymane adresy instrukcji lub operandów traktują jako adresy rzeczywiste. W ten sposób programy i systemy starych maszyn mogą funkcjonować, bez dokonania zmian, w nowych maszynach JS. Do zadawania reżimu TRANSLACJI przeznaczono piąty bit rozszerzonego słowa SSP. Jeśli bit ten jest 1, to translacja zachodzi, jeśli zaś 0, to nie zachodzi.

W wirtualnej maszynie można organizować kilka samodzielnych pamięci wirtualnych, przeznaczonych na przykład dla różnych programów. Każda taka pamięć wirtualna ma w PAO własną tabelę segmentów i własny ciąg tabel stron. Określenie adresu początkowego (bazy) bieżącej tabeli segmentów oraz jej długości dokonywane jest za pomocą stanu pierwszego rejestru sterującego (tj. RS1), w sposób pokazany na rys. 7.16.



Rys. 7.16. Strukutra rejestru RS1

Długość tabeli segmentów zapisana w RS1 służy do kontroli poprawności bieżącego indeksu segmentu adresu wirtualnego, tzn. sprawdza się, czy zachodzi nierówność: długość TSEG indeks SEG. Strukturę natomiast tabel segmentów (TSEG) i tabel stron oraz ich powiązań ilustruje rysunek 7.17. Tabele te zostają przygotowane przez wirtualny system operacyjny, a wykorzystywane są przez techniczne procedury translacji adresów.



Rys. 7.17. Struktura tabeli segmentów i tabeli stron

Każda tabela segmentów ma jedno jednosłowowe wejście (opis) dla każdego segmentu danej pamięci wirtualnej. Maksymalnie może być 256 wejść. Wejścia te podają adresy rzeczywiste początkowych komórek odpowiednich tabel stron (z dokładnością do 64 bajtów). Z każdym wejściem tabeli segmentów związana jest jedna tabela stron (czyli każda 16-Mbajtowa pamięć wirtualna może mieć do 256 tabel stron). Każda tabela stron może mieć dowolny wymiar, który określany jest przez bity 0÷3 wejścia danego segmentu (z dokładnością do jednej szesnastej maksymalnego wymiaru tabeli stron liczonego w bajtach).

Każda tabela stron ma jedno półsłowowe wejście dla każdej strony odpowiedniego segmentu. Wejście to podaje 13 lub 12 najstarszych bitów (2-K lub 4-Kbajtowe strony) rzeczywistego adresu obszaru PAO, zawierającego odpowiednią stronę oraz określa niedostępność (bit D) danej strony w PAO. Jeśli D=1, oznacza to, że dana strona nie znajduje się aktualnie w PAO i tym samym próby zwrotów do tej strony powodują przerwanie programowe do systemu operacyjnego, który zorganizuje wczytanie danej strony do PAO z PAZ.

7.2.2. Proces translacji adresów

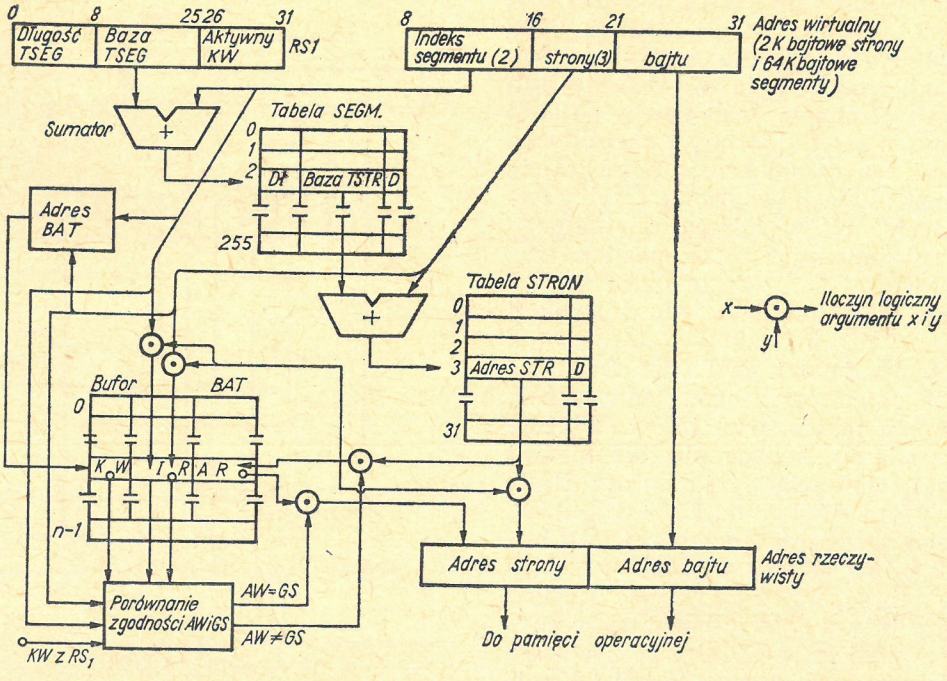
Proces translacji adresów wirtualnych na rzeczywiste może mieć miejsce w sposób bezpośredni lub pośredni. Bezpośrednia translacja jest uruchamiana przez programistę za pomocą specjalnej instrukcji ŁADUJ ADRES RZECZYWISTY. Instrukcja ta powoduje uruchomienie technicznych mechanizmów translacji (niezależnie od SSP), z zaangażowaniem rejestrów RSO i RS1 oraz tabeli segmentów i tabeli stron, w celu zamiany efektywnego adresu instrukcji (traktowanego jako adres wirtualny) na adres rzeczywisty i załadowania go do określonego rejestru ogólnego. Natomiast procedura translacji pośredniej polega na automatycznym tłumaczeniu wszystkich adresów instrukcji i operandów w procesie pracy programu. Pośrednia translacja odbywa się tylko wówczas gdy piąty bit rozszerzonego SSP jest jedynką (tzn. w reżimie translacji).

Proces translacji pośredniej pokazany jest na rys. 7.18. Może on przebiegać w dwojaki sposób:
— bez zastosowania bufora BAT,
— z zastosowaniem bufora BAT.
W przypadku braku bufora BAT (lub braku odpowiedniego adresu w buforze) proces translacji przebiega następująco: w pierwszej fazie, za pomocą indeksu segmentu adresu wirtualnego, wybiera się odpowiednie wejście tabeli segmentów z uwzględnieniem bazowego adresu tej tabeli (przechowanego w rejestrze RS1). W ten sposób wybrany zostaje bazowy adres rzeczywisty odpowiedniej tabeli stron. W drugiej fazie suma bazowego adresu tabeli stron i bieżącego indeksu strony określa wejście odpowiedniej strony (czyli adres rzeczywisty strony), po czym do adresu rzeczywistego strony dołącza się indeks bajtu, otrzymując w ten sposób pełny adres rzeczywisty.

W celu przyśpieszenia procesu translacji, ponieważ tabele segmentów i tabele stron znajdują się w PAO, wprowadzono zestaw szybkich rejestrów (zwany buforem adresów translacji BAT), w których przechowuje się adresy pary odpowiedności ADRES WIRTUALNY — ADRES RZECZYWISTY oraz inne pomocnicze informacje (np. numer lub kod pamięci wirtualnej) najbardziej aktywnych stron w procesie przetwarzania.

Jeśli system operacyjny organizuje kilka niezależnych 16-Mbajtowych pamięci wirtualnych, to pamięciom tym przypisuje się kody (np. 1, 2, 3) i przechowuje się je w odpowiedniej tabeli systemu. Kod aktualnie przetwarzanej pamięci wirtualnej zapisuje się w RS1. Schemat translacji adresów z zastosowaniem bufora BAT przebiega następująco: porównuje się w pierwszym kroku bieżący adres wirtualny (z dokładnością do adresu strony) z adresem lub adresami wirtualnymi zapisanymi w buforze BAT, z uwzględnieniem kodu pamięci wirtualnej, tzn. porównuje się również bieżący (aktywny) kod pamięci wirtualnej (KW) z kodem wirtualnym rejestru bufora BAT, w którym znajduje się adres wirtualny równy bieżącemu adresowi. Jeśli pełna zgodność porównywanych wielkości zachodzi, to adres rzeczywisty (AR) zapisany w tym rejestrze bufora BAT jest brany jako finalny adres rzeczywisty (łącznie z indeksem bajtu) i na tym proces translacji kończy się. Zatem w tym przypadku nie korzysta się z tabeli segmentów i tabeli stron znajdujących się w PAO, co oznacza, że translacja jest praktycznie natychmiastowa. Jeśli natomiast powyższe porównanie nie zachodzi, to uruchamia się normalną procedurę translacji, która korzysta ze stanu rejestru RS1, tabeli segmentów i tabeli stron. W ostatniej fazie tej normalnej procedury, tzn. po ustaleniu adresu rzeczywistego strony, adres ten zostaje również zapamiętany w odpowiednim rejestrze bufora BAT łącznie z właściwym adresem wirtualnym i kodem. W ten sposób przy ponownym odwołaniu do tej strony proces translacji przebiegać będzie za pośrednictwem bufora BAT.

W przypadku konieczności uruchomienia normalnej procedury translacji można wprowadzić drobne udoskonalenie, które zmniejszy ilość zwro-



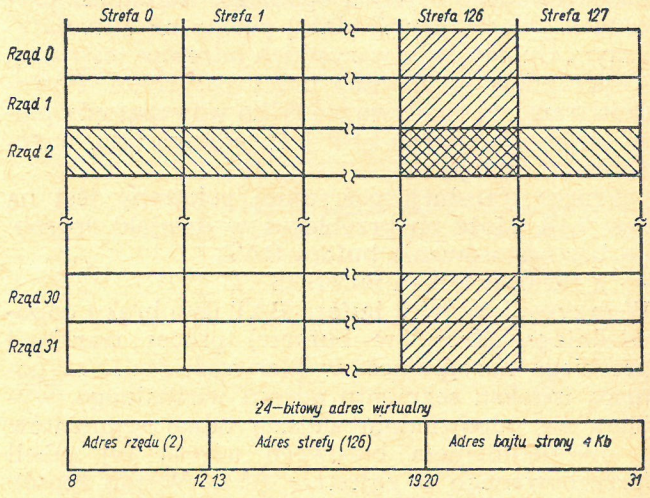
Rys. 7.18. Schemat translacji adresów (za pomocą lub bez pomocy bufora BAT)
IR — indeks rzędu pamięci wirtualnej w BAT, AR — adres rzeczywisty (strony), KW — kod pamięci wirtualnej, GS — bieżący indeks segmentu i strony

tów do PAO o 50%, tzn. z dwóch do jednego. Udoskonalenie to polega na wprowadzeniu dodatkowego pojedynczego rejestru, w którym zapamiętuje się ostatni pobrany z PAO indeks segmentu oraz jego wejście z tablicy segmentów. W ten sposób jeśli następne odwołanie do tablicy segmentów dotyczy tego samego segmentu, co jest bardzo prawdopodobne, to nie organizuje się zwrotu do tablicy, lecz przyjmuje się za wejście segmentu zawartość dodatkowego rejestru. W ten sposób otrzymuje się adres bazowy tabeli stron bez zwrotu do PAO. Oczywiście, jeśli odwołanie dotyczy innego segmentu niż poprzedni, wymagane jest uruchomienie pełnej procedury translacji.

7.2.3. Metody przeszukiwania bufora BAT

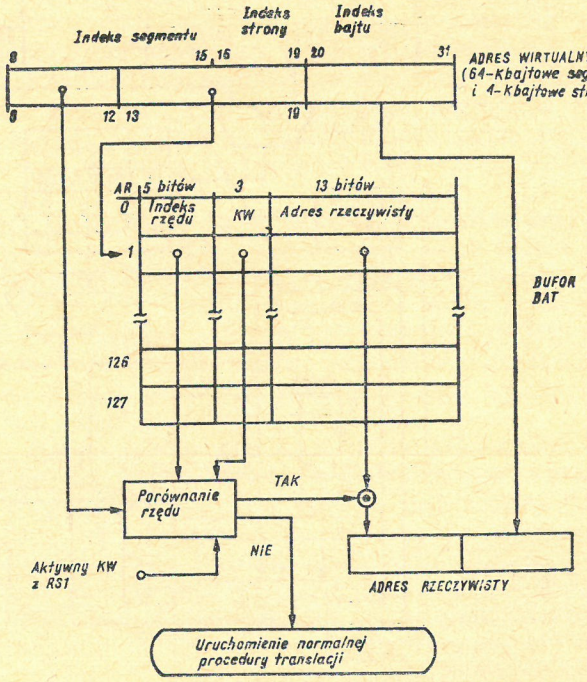
Wielkość bufora BAT jest różna w różnych maszynach; przeważnie przyjmuje się od 8 do 128 rejestrów. W celu porównania zawartości bufora BAT z bieżącym adresem wirtualnym stosuje się różne algorytmy. Dla małych pojemności BAT, np. 8 lub 16 rejestrów, przyjmuje się klasyczne metody pamięci asocjacyjnej, tzn. przez jednoczesne porównanie zawartości rejestrów BAT. Natomiast dla dużych buforów BAT, np. dla 128 rejestrów, przyjmuje się tańsze w porównaniu z metodami asocjacyjnymi rozwiązania. W rozwiązaniach tych nie stosuje się n szybkich układów porównania (gdzie n jest ilością rejestrów bufora), lecz jeden wspólny układ. W tym przypadku trudność polega na wyborze rejestru bufora BAT, którego zawartość należy porównać z bieżącym adresem wirtualnym. Oczywiście nie wchodzi w grę kolejne porównanie wszystkich rejestrów bufora w celu rozstrzygnięcia, czy w buforze znajduje się szukany adres wirtualny, czy też nie, gdyż taka procedura mogłaby trwać dłużej (mimo układowej realizacji) niż normalna translacja za pomocą pełnych tabel w PAO. Dlatego przy zastosowaniu dużych buforów przyjęto metodę macierzy indeksów. Metoda ta opiera się na założeniu sekwencyjności wykonania programu. Oznacza to, że istnieje duże prawdopodobieństwo, iż kolejne instrukcje lub argumenty operacji znajdują się w tej samej lub w kilku sąsiednich stronach. Jeśli zatem pojawił się w instrukcji adres pewnej strony z określonego segmentu, to adres ten często będzie pojawiał się i w następnych instrukcjach programu. Czyli pewna część „starszych" bitów adresu wirtualnego stanowić będzie niezmienniki przez pewien czas procesu obliczeniowego. Oczywiście może to być inna grupa bitów-niezmienników dla instrukcji programu głównego, dla instrukcji podprogramu, dla danych ALFA, dla danych BETA itd. Dla BAT o pojemności 128 rejestrów można jednocześnie mieć do 128 różnych takich grup bitów. Właśnie te „starsze" bity adresu, będące niezmiennikami przez pewien czas, stanowią adres bufora BAT. Jeśli pojemność bufora BAT wynosi 128 rejestrów, dla jednoznacznego określenia adresu rejestru bufora wystarczy 7 bitów adresu wirtualnego. Bity te można różnie dobrać, np. 4 bity z pola indeksu strony i 3 młodsze bity z pola indeksu segmentu. Zaś pozostałe bity indeksu segmentu, tj. pozostałe 5 bitów dla 64-Kbajtowych segmentów i 4-Kbajtowych stron, służą do porównania z zawartością 5-bitowej części rejestru bufora BAT, w której przechowuje się pozostałą część indentyfikatora segmentu, czyli tzw. indeks rzędu (patrz rysunek 7.20). Zatem długość rejestrów bufora BAT może składać się z bitów: 5+13+bity kontrolne i sterujące. Pierwszych 5 bitów — to identyfikator starszych bitów segmentu, zaś pozostałych 13 bitów określa adres rzeczywisty danej strony.

Oczywiście wybór bitów adresujących rejestr bufora BAT oraz struktura programu wpływa na liczbę przeładowań rejestrów bufora BAT. Tak skonstruowany adres bufora BAT jest adresem grupy stron lub segmentów należących do jednego rzędu.



Rys. 7.19. Podział 16-Mbajtowej pamięci wirtualnej na rzędy i strefy pod względem organizacji bufora BAT

Rys. 7.19 ilustruje umowny podział pamięci wirtualnej na strefy, tj. grupy stron lub segmentów i na rzędy z punktu widzenia adresacji bufora BAT. Zatem strefa jest grupowym adresem ciągu stron należących do różnych segmentów i różnych pamięci wirtualnych. Pełnym indeksem lub identyfikatorem segmentu jest początkowa za-



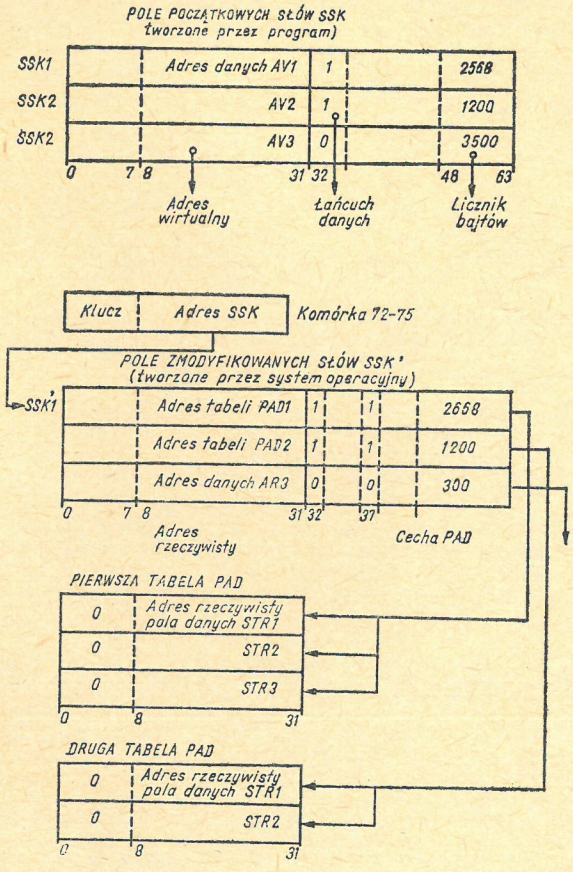
Rys. 7.20. Schemat wyboru rejestru bufora adresów translacji AR — adres rejestru bufora BAT

wartość rejestru bufora BAT, tj. indeks rzędu (IR) i kod pamięci wirtualnej (KW). Części IR i KW wszystkich rejestrów bufora BAT można traktować jako tabelę lub macierz indeksów indentyfikacji (rys. 7.20).

Przyjęcie bufora BAT o pojemności 128 rejestrów pozwala na pamiętanie adresów rzeczywistych od 100 do 25% stron pamięci rzeczywistej. Jeśli bowiem przyjmiemy, że strona ma 4 Kb, a PAO ma 512 Kb, to adresy rzeczywiste (oraz ich wirtualne identyfikatory) wszystkich stron można pomieścić w buforze BAT. Jeśli zaś PAO ma 2048 Kb, to odpowiednie miejsce w BAT dzielą między sobą 4 strony, gdyż wówczas PAO ma cztery rzędy po 512 Kb. Pojawienie się strony w procesie obliczeń z innego rzędu powoduje przeładowanie odpowiedniego rejestru BAT.

7.2.4. Pośrednie adresowanie danych w kanale

W maszynach wyposażonych w mechanizmy pamięci wirtualnej zachodzi potrzeba przekształcania adresów wirtualnych — występujących przy przesyłaniu danych pomiędzy PAO a urządzeniami zewnętrznymi — na adresy rzeczywiste. Ponadto pole danych traktowane przez program jako obszar spójny nie jest nim w rzeczywistości. Pole to jest bowiem mechanicznie dzielone na stronice (np. po 2 Kbajty) i znajduje się na ogół w niespójnych obszarach PAO. W związku z tym powstaje potrzeba wyposażenia kanałów maszyn wirtualnych w specjalne dodatkowe cechy i mechanizmy, które powodują automatyczny przeskok z jednego fragmentu pola danych na drugi, gdy fragmenty te leżą na różnych stronicach pamięci (które nie muszą być kolejne w realnej pamięci). Na przykład pole danych — spójne z punktu widzenia programu — o długości 2568 bajtów może składać się z trzech fragmentów leżących na trzech różnych stronicach (np. na pierwszej 120 bajtów, na drugiej 2048 i na trzeciej 400).



Rys. 7.21. Uproszczona struktura tabel używanych w pośrednim adresowaniu danych (PAD) w kanałach „wirtualnych" (przykład łańcucha danych, składającego się z trzech SSK)
STR — stronica, AR3 — adres rzeczywisty pola danych znajdujących się na jednej stronicy

Przyporządkowanie stronicom wirtualnym pola w pamięci rzeczywistej (PAO) jest oczywiście dokonywane przez system operacyjny i dlatego system ten tworzy specjalne tabele słów sterujących kanałów (SSK), gdzie zastępuje się adresy wirtualne — występujące w SSK przygotowywanych przez program — adresami rzeczywistymi. Przykładowa struktura takich tabel pokazana jest na rys. 7.21. Jak widać z tego rysunku, system operacyjny przygotowuje tzw. pole zmodyfikowanych słów SSK', gdzie wirtualny adres danych j-tego SSK zostaje zastąpiony przez adres j-tej tabeli pośredniego adresowania danych (PAD_j). Tabela PAD jest tworzona przez system operacyjny i składa się z rzeczywistych adresów fragmentów pól danych rozmieszczonych na różnych stronicach po 2 Kbajty. Jeśli określone — przez adres danych i licznik bajtów słowa SSK — pole danych znajduje się na s stronicach, to tabela PAD tego pola również składa się z s wierszy. Zatem każdemu SSK przyporządkowuje się jedną tabelę PAD. Na rys. 7.21 występują trzy SSK, które tworzą łańcuch danych.

Mechanizm pośredniego adresowania danych w kanale zadawany jest za pomocą bitu 37 słowa SSK (tzw. cecha PAD). Jeśli w tym bicie jest zero, to kanał działa normalnie. Oznacza to, że bity 8—31 słowa SSK' zawierają rzeczywisty adres danych. Jeśli natomiast w bicie tym jest jedynka, to adres zapisany w bitach 8—31 zmodyfikowanego słowa SSK' nie określa adresu pola danych, lecz podaje adres początkowy tabeli PAD zawierającej cały ciąg adresów fragmentów pola danych. Pierwszy wiersz (słowo) tabeli PAD określa początek pola danych. Pozostałe wiersze tej tabeli określają początkowe adresy stronic, na których znajdują się kolejne porcje pola danych (zatem bity 21—31 tych wierszy zawierają zera).

Działanie mechanizmu pośredniego adresowania danych jest następujące:

a) kanał pobiera — po zapoczątkowaniu jego pracy przez instrukcję START WE-WY — słowo adresowe SAK z komórki 72;
b) kanał pobiera podwójne słowo SSK' (według adresu słowa SAK);
c) kanał bada stan bitu 37 słowa SSK'; jeśli w bicie tym jest zero, to rozpoczyna się normalny proces przesyłania danych, jeśli natomiast w bicie tym jest jedynka, to adres zapisany w bitach 8—31 słowa SSK' traktowany jest jako adres początku tabeli PAD;
d) kanał pobiera pierwszy wiersz tabeli PAD — gdzie znajduje się początkowy adres pola danych — i rozpoczyna się proces przesyłania danych.

Proces przesyłania trwa aż do momentu otrzymania adresu danych, będącego ostatnim adresem

stronicy. Po dojściu do takiego adresu proces przesyłania danych zostaje chwilowo wstrzymany w celu pobrania kolejnego wiersza z tabeli PAD, po czym proces przesyłania zostaje wznowiony z nowym adresem danych (który należy do nowej stronicy). Powyższy mechanizm jest powtarzany aż do przesłania całego pola danych, tj. aż do wyzerowania się licznika bajtów lub aż do otrzymania sygnału „urządzenie zakończyło pracę". Po czym, jeśli w przesyłaniu występuje również cecha „łańcuch danych" lub „łańcuch komend", pobiera się nowe SSK'. Czyli kanał powtarza swoje działanie od punktu *b* (gdzie adresem SSK' jest adres poprzedniego SSK' plus 8).

Po zakończeniu operacji wejścia-wyjścia i po przyjęciu przerwania programu zachodzi potrzeba zamiany z kolei adresu rzeczywistego ze słowa stanu kanału (SSTK) na adres wirtualny oraz „odblokowanie ruchu" stronic zawierających pola danych już przesłanych. W czasie bowiem przesyłania danych do określonych stronic musi obowiązywać „blokada ruchu" tych stronic, w algorytmie wymiany stronic (pomiędzy PAO i pamięcią dyskową).

8. Urządzenia zewnętrzne

Urządzenia zewnętrzne Jednolitego Systemu EMC to pamięci zewnętrzne oraz różne inne środki wprowadzania (wyprowadzania) informacji do jednostki centralnej danego systemu od jego otoczenia. Urządzeniami wprowadzania-wyprowadzania informacji są w szczególności takie urządzenia, jak: pamięci o dostępie bezpośrednim (pamięci dyskowe i bębnowe), pamięci taśmowe magnetyczne, czytniki i dziurkarki kart, różne typy drukarek, czytniki i dziurkarki taśmy papierowej, elektryczne maszyny do pisania, urządzenia wprowadzania i wyprowadzania informacji w postaci graficznej, urządzenia teleprzetwarzania i inne [8] [10].

Urządzenia zewnętrzne są na ogół dołączane bezpośrednio do jednej Jednostki Sterującej, która to jednostka współpracuje z wybranym kanałem we-wy. Niemniej istnieją także środki przełączające, które pozwalają podłączyć określone urządzenia zewnętrzne do różnych (przeważnie do dwóch) jednostek sterujących lub podłączyć jednostki sterujące do różnych kanałów wejścia-wyjścia, przy czym przełączenie urządzenia zewnętrznego z kanału na kanał (tej samej lub innej jednostki centralnej) jest dokonywane dynamicznie i — praktycznie — natychmiast.

8.1. Pamięci zewnętrzne o dostępie bezpośrednim

Pamięci zewnętrzne o dostępie bezpośrednim — opracowane w JS EMC — to pamięci dyskowe i pamięci bębnowe, gdzie względnie szybko — w porównaniu na przykład z pamięcią taśmową — można uzyskać dostęp do informacji zapisanej na powierzchni wirującej dysku lub bębna.

W nomenklaturze pamięci zewnętrznych JS EMC znajdują się zarówno pamięci dyskowe wymienne jak i stałe. Pamięci dyskowe JS-5050, JS-5052, JS-5055, JS-5056 i JS 5058 (tabela 8.1) są typu wymiennego i stosują pakiet dysków składający się z 6 dysków (gdzie wykorzystano 10 powierzchni dla zapisu informacji). Natomiast pamięć dyskowa JS-5051 jest typu stałego. Składa się ona z dwóch zespołów — po 18 dysków każdy — i ma łącznie 64 powierzchnie robocze.

Pamięciami bębnowymi są dwie jednostki: JS-5033 i JS-5035. Zapis informacji na powierzchnię nośnika pamięci o bezpośrednim dostępie jest szeregowy, tzn. informacje zapisuje się bit po bicie (z wyjątkiem pamięci JS-5033, gdzie pisze się po 8 bitów na raz).

Pamięci dyskowe JS EMC są wyposażone w pewną ilość dodatkowych ścieżek, traktowanych jako zapasowe. Oznacza to, że jeśli jakaś robocza ścieżka na dysku zostanie uszkodzona, usuwa się

Pamięci o dostępie bezpośrednim

Tabela 8.1

Charakterystyka	JS-5050 ZSRR	JS-5051 ZSRR	JS-5052 LRB	JS-5055 NRD	JS-5056 ZSRR	JS-5058 CSRS	JS-5033 ZSRR	JS-5035 PRL
Pamięć dyskowa:								
wymienna	tak	—	tak	tak	tak	tak	—	—
stała	—	tak	—	—	—	—	—	—
Pamięć bębnowa	—	—	—	—	—	—	tak	tak
Pojemność pakietu dyskowego lub bębna, Mbajtów	7,25	100	7,25	7,25	7,25	7,25	6	2
Liczba powierzchni roboczych	10	64	10	10	10	10	—	—
Liczba ścieżek informacyjnych na każdej powierzchni	200	128	200	200	200	200	1000	480
Pojemność ścieżki, bajtów	36250	51210	36250	36250	36250	36250	6000	4170
Szybkość obrotów, obr/min	2400	900	2400	2400	2400	2400	1500	1500
Średni czas dostępu do informacji, ms	90	250	90	90	90	90	20	20
Średni czas znalezienia cylindra, ms	75	250	75	75	75	75	—	—
Szybkość przesyłania danych, Kbajtów/s	156	83	156	156	156	156	1250	800
Szyfr jednostki sterującej (JS)	5551	5551	5552	5555	5551	5558	5533	5551
Maksymalna liczba pamięci dyskowych podłączonych do JS	8	8	8	8	8	8	8	8

ją, a jej funkcję przyjmuje jedna z zapasowych ścieżek.

Pakiet dysków dla pamięci typu JS-5050, JS-5052, JS-5055, JS-5056 i JS-5058 składa się z 203 cylindrów, każdy z 10 ścieżkami. Maksymalna pojemność cylindra wynosi 36 250 bajtów, zaś pojemność całego wymiennego pakietu dyskowego — 7,25 Mbajtów. Natomiast pojemność stałej pamięci dyskowej JS-5051 wynosi 100 Mbajtów.

Ważną cechą charakterystyczną pamięci o dostępie bezpośrednim jest całkowity czas dostępu i przesyłania danych. Czas całkowity jest na ogół sumą następujących składowych:

— czas przesuwu mechanizmu dostępu,
— czas wyboru głowicy czytająco-piszącej,
— czas oczekiwania na nośnik (w ruchu obrotowym),
— czas przesyłania danych.

Czas przesuwu mechanizmu dostępu, tj. czas znalezienia określonego cylindra, jest różny i zależy od położenia szukanego cylindra względem początkowej pozycji mechanizmu dostępu. Minimalny czas przesuwu pamięci JS-5050 wynosi 25 ms, zaś maksymalny — 150 ms (w tabeli 8.1 przyjęto średni czas znalezienia cylindra równy 75 ms, czyli połowie czasu maksymalnego).

Czas wyboru głowicy czytająco-piszącej zależy od szybkości działania różnych elektronicznych układów wybierania i przez to jest bardzo mały w porównaniu z innymi składowymi.

Przyjmuje się, że czas oczekiwania na obrót nośnika (tj. do chwili, kiedy początek danych znajdzie się pod głowicą) wynosi połowę czasu pełnego obrotu nośnika (patrz tabela 8.1). W związku z tym przyjęto w przybliżeniu, że średni czas dostępu do informacji (jako suma trzech pierwszych czasów) w pamięci JS-5050 wynosi 90 ms.

Czas przesyłania danych pomiędzy pamięcią zewnętrzną a pamięcią operacyjną zależy oczywiście zarówno od szybkości obrotów nośnika, jak i od gęstości zapisu informacji na nośniku. Szybkość przesyłania danych dla różnych typów pamięci o dostępie bezpośrednim jest podana w tabeli 8.1 i dla większości pamięci wynosi 156 Kbajtów na sekundę. Pamięci dyskowe wymienne o większej szybkości przesyłania (np. 312 i 806 Kbajtów/s) oraz o większej pojemności (np. 30 lub 100 Mbajtów) są również przygotowywane do produkcji.

Na wystawie Jednolitego Systemu EMC w Moskwie (maj 1973) zademonstrowana została, w stoisku Ludowej Republiki Bułgarii, pamięć dyskowa o pojemności 30 Mbajtów.

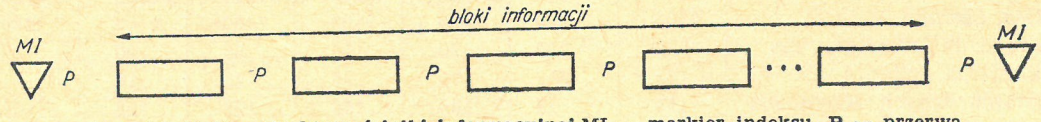
Pamięci o dostępie bezpośrednim podłącza się do kanałów we-wy za pośrednictwem grupowych jednostek sterujących, których szyfry zawiera tabela 8.1. Jednostki te mogą pracować wyłącznie w trybie selektorowym. Jednostki sterujące mogą natomiast być podłączone zarówno do kanału selektorowego, jak i do kanału multiplekserowego (o ile przepustowość kanału przewyższa szybkość przesyłania urządzenia).

Główną funkcją jednostki sterującej (JS) jest rozszyfrowywanie oraz wykonywanie komend i poleceń otrzymywanych od kanału, organizowanie przesyłania danych na drodze JS-KANAŁ i JS-URZĄDZENIE oraz przesyłanie do kanału informacji o stanie urządzenia zewnętrznego. Jednostka sterująca przeprowadza również operacje związane z kontrolą przesyłanych danych. Przy zapisie informacji i po sprawdzeniu poprawności cechy parzystości na drodze KANAŁ-JS usuwa się z 9-bitowych bajtów danych bit parzystości i tworzy się nową cechę kontroli, zwaną kontrolą cykliczną. Dwa bajty kontroli cyklicznej (KC) zapisuje się zawsze na końcu każdego bloku informacji.

Natomiast przy czytaniu danych tworzy się ponownie bajty kontroli cyklicznej i porównuje się je z bajtami KC zapisanymi na końcu bloku oraz uzupełnia się każdy 8-bitowy bajt danych — przesyłany do kanału — dodatkowym bitem cechy parzystości i usuwa się bajty KC z końca bloku. Powyższa metoda kontroli nie tylko daje duże oszczędności w wykorzystaniu miejsca pamięci dyskowej, lecz także wykrywa większą ilość błędów w porównaniu z tradycyjną cechą poprzecznej parzystości (gdzie na każdy bajt danych przeznacza się jeden bit kontrolny).

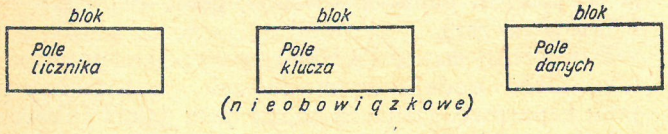
Organizacja danych. Ścieżki informacyjne powierzchni pamięci dyskowej są synchronizowane za pomocą wyróżnionego punktu zwanego markierem indeksu (MI). Punkt ten jest umownym fizycznym początkiem wszystkich ścieżek nośnika i rozpoznawany jest przez elektronikę urządzenia. Dane są zapisywane na ścieżki w formie ciągu bajtów zwanych b l o k a m i. Każdy blok kończy się przerwą — zwaną przerwą międzyblokową — w której nie ma żadnych danych (patrz rys. 8.1).

Zapis danych (lub rekord danych) składa się na ogół z trzech kolejnych bloków (rys. 8.2), w których zapisuje się takie informacje, jak:



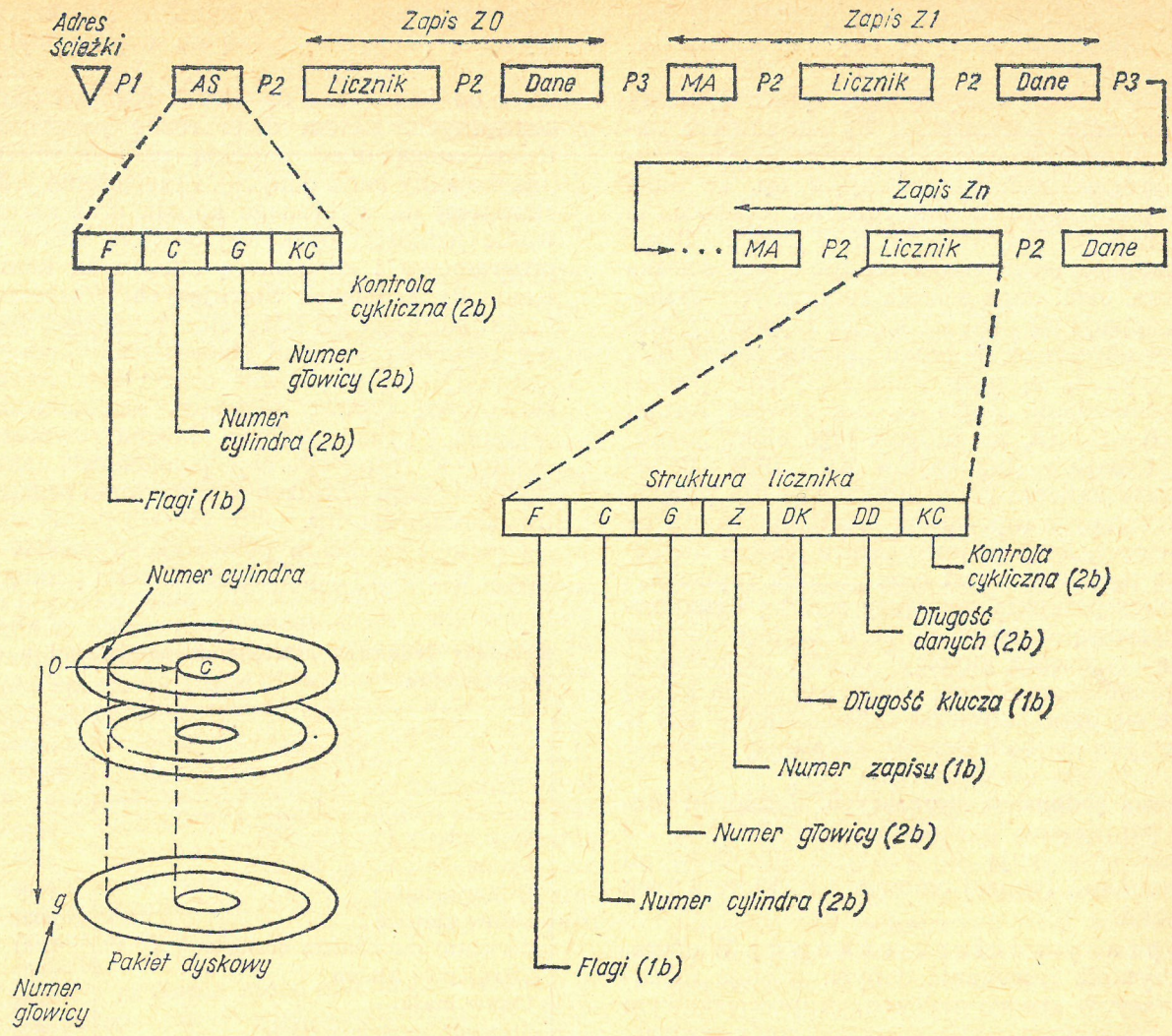
Rys. 8.1. Ogólna struktura ścieżki informacyjnej MI — markier indeksu, P — przerwa

— licznik, znajdujący się zawsze na początku zapisu i określający między innymi długość dwóch pozostałych bloków zapisu, tj. bloku klucza i bloku danych; w tym polu zapisuje się również numer cylindra, numer głowicy i numer danego zapisu;
— klucz, w którym zapisuje się dowolne informacje identyfikacji zapisu kartoteki; blok ten może nie występować w zapisie danych;
— dane właściwe danego zapisu kartoteki, zakończone dwoma bajtami KC.



Rys. 8.2. Struktura zapisu danych

Format danych na ścieżce ma określoną strukturę formowaną przez jednostkę sterującą. Wszystkie jednostki sterujące JS EMC tworzą identyczne struktury danych na ścieżce (rys. 8.3). Każda ście-



Rys. 8.3. Rozmieszczenie informacji na ścieżce (zapisy nie zawierają bloku klucza)

żka zawiera na początku określone informacje sterujące — tj. adres ścieżki i zapis Z_0 — oraz ciąg zapisów zawierających właściwe dane kartoteki (tj. $Z_1, Z_2, \ldots, Z_n$). Przerwy międzyblokowe mają różną długość, zależną od typu urządzenia, od położenia przerwy oraz od długości poprzedzającego zapisu. Przerwy typu P1 i P2 — pokazane na rysunku 8.3 — mają stałą długość dla wszystkich typów urządzeń, natomiast długość przerwy typu P3 zależy od długości poprzedzającego zapisu.

Początkowy blok na ścieżce, tj. blok „adres ścieżki", zawiera informacje niezbędne do identyfikacji danej ścieżki oraz informacje o stanie technicznym danej ścieżki. Blok „adresu" zawiera 7 bajtów informacji i kończy się — jak każdy blok — cechą kontroli cyklicznej. Natomiast początkowy bajt tego bloku określa różne cechy techniczne danej ścieżki, np. ścieżka uszkodzona lub dobra, informacyjna lub zapasowa. Cztery środkowe bajty określają kolejno numer cylindra, do którego należy dana ścieżka, oraz numer głowicy czytająco-piszącej danej ścieżki. Informacje zawarte w bloku „adres ścieżki" są zapisywane i czytane specjalnymi komendami.

Zapisy danych użytkownika składają się z trzech lub czterech części:

— markier adresu, który zajmuje dwa bajty i jest znacznikiem początku danego zapisu; jest zapisywany przez jednostkę sterującą danego urządzenia;
— licznik, którego długość wynosi 11 bajtów i określa różne cechy związane z danym zapisem (patrz rys. 8.3); blok „licznik" określa głównie format zapisu danych:
  ● format LICZNIK—DANE,
  ● format LICZNIK—KLUCZ—DANE,
  oraz długość bloku „klucz" (DK) i długość bloku „dane" (DD).
  Jeśli wielkość DK=0, nie występuje odrębny blok „klucz". Wówczas po bloku „licznik" następuje blok „dane", czyli wybiera się format LICZNIK—DANE. Jeśli natomiast wielkość DD=0, wówczas oznacza to końcowy zapis danego ciągu zapisów (w tym przypadku blok danych redukuje się do dwóch bajtów KC);
— klucz — to blok, w którym zapisuje się klucz użytkownika;
— dane — to blok, składający się z DD-2 bajtów właściwych danych oraz z dwóch bajtów cechy kontroli cyklicznej; zatem wielkość DD (rys. 8.3) określa ilość bajtów w bloku danych łącznie z bajtami KC.

Zapisy danych użytkownika poprzedza się specjalnym zapisem Z_o, który jest opisem ścieżki. Zapis Z_o występuje bezpośrednio po bloku „adres ścieżki" i jest wykorzystywany przez systemy programowania. Zatem zapis Z_o nie zawiera danych użytkownika, lecz tylko informacje dla systemu programowania określające np. że dana ścieżka jest zapasowa lub robocza. Uszkodzona ścieżka robocza ma wielkość 2 w bajcie „flagi" bloku „adres ścieżki" oraz wielkość 0, gdy ścieżka jest dobra. Jeśli dana ścieżka robocza jest uszkodzona, zapisuje się dane na ścieżce zapasowej, której adres określa się w bloku „licznik zapisu Z_o". Zapisy typu Z_o są zapisywane i czytane specjalnymi komendami. Zapis Z_o ma zawsze format LICZNIK—DANE, zaś długość jego bloku „dane" wynosi 8 bajtów (bez bajtów KC). Blok „dane" tego zapisu zawiera różne informacje dla programów sterujących (np. numer ostatniego zapisu na danej ścieżce, ilość bajtów obszaru jeszcze nie zapisanego itp.). Natomiast blok „licznik zapisu Z_o" jest tablicą przejścia pomiędzy ścieżką uszkodzoną a zapasową. Oznacza to, że w bajtach „numer cylindra" i „numer głowicy" ścieżki uszkodzonej znajduje się adres ścieżki zapasowej — na której zapisuje się dane ze ścieżki uszkodzonej — zaś w tych samych bajtach ścieżki zapasowej zapisuje się adres ścieżki uszkodzonej.

Możliwości jednostek sterujących. Jednostki sterujące pamięciami o dostępie bezpośrednim JS EMC charakteryzują się różnymi dodatkowymi możliwościami. Poniżej omówimy niektóre z tych możliwości.

Tryb pracy wielościeżkowej. W operacjach czytania i szukania danych istnieje możliwość pracy w dwóch trybach: jednościeżkowym i wielościeżkowym. Tryb wielościeżkowy pozwala automatycznie przechodzić z jednej ścieżki do następnej, gdy napotka markier indeksu. Przejście to, tj. przejście z jednej głowicy do następnej, ogranicza się tylko do ścieżek tego samego cylindra. Tryb wielościeżkowy jest zwykle stosowany w operacjach szukania, gdyż jedną operacją można przeszukać cały cylinder pamięci dyskowej.

Przeniesienie zapisu. W operacjach zapisu istnieje możliwość rozmieszczenia informacji na więcej niż jednej ścieżce pamięci. Zatem można operować zapisami danych, których długość przekracza pojemność jednej ścieżki (lecz nie jednego cylindra). Zapisy nie mieszczące się na jednej ścieżce mają inną strukturę niż zapisy standardowe. W tej strukturze każdy segment — tj. każda część zapisu umieszczona na jednej ścieżce — ma swoje pole typu licznik, w którym określa się długość danych tylko tego segmentu. Wszystkie segmenty mają odpowiednią cechę w bajcie „flagi", która przy stanie 1 oznacza, że zapis ma dalsze segmenty. Natomiast w ostatnim segmencie cecha ta ma stan 0. Zapisy nie mieszczące się na jednej ścieżce mają jedno pole typu klucz; pole to znajduje się zawsze w pierwszym segmencie. Zapisu segmentów zapisów z nadmiarem dokonuje się za pomocą odrębnych komend. Komendy te ustawiają między innymi cechy w bajcie „flagi".

Przełącznik dwukanałowy. Jednostki sterujące JS EMC wyposażone są w środki techniczne pozwalające na podłączenie danej jednostki sterującej do dwóch różnych kanałów wejścia-wyjścia. Dzięki temu można przydzielić jednostkę sterującą do dwóch kanałów i zarezerwować wybraną pamięć dyskową do współpracy z jednym kanałem. Natomiast same kanały mogą należeć do jednej lub do dwóch różnych jednostek centralnych. Zatem dwie różne jednostki centralne mogą mieć wspólną pamięć zewnętrzną (np. wspólny bank danych). Przełączenie jednostki sterującej do wybranego kanału oraz zarezerwowanie jednostki dyskowej odbywa się w sposób programowy i za pomocą odrębnych komend.

Komendy Jednostek Sterujących. Wszystkie jednostki sterujące JS EMC stosują na ogół jednakowy repertuar komend (tabela 8.2). Komendy te pozwalają zapoczątkować operacje czterech podstawowych typów: sterujące, szukania danych, czytania i pisania danych. Można również zapoczątkować różne operacje pomocnicze, takie jak: czytania stanu urządzenia, ustawiania przełącznika dwukanałowego itp.

Operacje szukania pozwalają wyszukać informacje wcześniej zapisane na nośniku pamięci o do-

Tabela 8.2

Komendy jednostek sterujących pamięciami o dostępie bezpośrednim

Komendy	Kod komendy w trybie jednościeżkowym
1	2
Bieg jałowy (nic nie rób)	0000 0011
Ustaw mechanizm dostępu	0000 0111
Ustaw cylinder	0000 1011
Ustaw głowicę	0001 1011
Zorientuj mechanizm dostępu	0001 0011
Steruj pole klucza	0000 1111
Ustaw maski	0001 1111
Sprawdź wejście-wyjście	0000 0000
Szukaj adres ścieżki	0011 1001
Szukaj identyfikator równy (danemu)	0011 0001
Szukaj identyfikator większy	0101 0001
Szukaj identyfikator równy lub większy	0111 0001
Szukaj klucz większy (danemu)	0100 1001
Szukaj klucz równy	0010 1001
Szukaj klucz równy lub większy	0110 1001
Szukaj klucz i dane równe	0010 1101
Szukaj klucz i dane większe	0100 1101
Szukaj klucz i dane równe lub większe	0110 1101
Kontynuuj szukanie typu „większy"	0100 0101
Kontynuuj szukanie „równy"	0010 0101
Kontynuuj szukanie „równy" lub „większy"	0110 0101
Kontynuuj szukanie przy niespełnionym warunku	0101 0101
Kontynuuj szukanie przy spełnionym warunku	0011 0101
Czytaj program pierwotny	0000 0010
Czytaj dane	0000 0110
Czytaj klucz i dane	0000 1110
Czytaj licznik, klucz i dane	0001 1110
Czytaj zapis Z_o	0001 0110
Czytaj adres ścieżki	0001 1010
Czytaj licznik	0001 0010
Pisz adres ścieżki	0001 1001
Pisz Z_o	0001 0101
Pisz licznik, klucz i dane	0001 1101
Pisz spec. licznik, klucz i dane	0000 0001
Kasuj	0001 0001

Tabela 8.2 (cd.)

1	2
Pisz dane	0000 0101
Pisz klucz i dane	0000 1101
Zarezerwuj jednostkę pamięci	1011 0100
Zwolnij jednostkę pamięci	1001 0100
Czytaj stan szczegółowy	0000 0100

stępie bezpośrednim. Argumentem szukania może być określony identyfikator zapisu, klucza, klucza i danych lub identyfikator formatu zapisu. Proces szukania określonej informacji jest realizowany w jednostce sterującej, gdzie bajt po bajcie porównuje się — według określonej relacji typu „równy", „większy" lub „równy lub większy" — identyfikator otrzymany z pamięci operacyjnej z informacją przeczytaną z pamięci dyskowej lub bębnowej. Proces porównywania jest kontynuowany do momentu spełnienia się warunku danej relacji porównania lub do momentu natrafienia na koniec obszaru szukanego. Jeśli dany warunek zostanie spełniony, jednostka sterująca wysyła do kanału specjalną cechę (tzw. modyfikator), która powoduje, że kanał opuszcza kolejną komendę występującą w łańcuchu komend. Opuszczona komenda jest z reguły typu PRZEŁĄCZ W KANALE. Jeśli natomiast dany warunek nie zachodzi — w szukanym polu — to kanał wykonuje kolejną komendę w łańcuchu komend, mianowicie komendę PRZEŁĄCZ W KANALE, która poprzez kolejne słowo sterujące kanału (SSK) zadaje znowu operację szukania, dotyczącą kolejnego zapisu na dysku lub bębnie.

Operacja szukania może przebiegać zarówno w trybie jedno- jak i wielościeżkowym. W pierwszym trybie proces szukania ograniczony zostaje do zapisów jednej tylko ścieżki, zaś w trybie wielościeżkowym przechodzi się automatycznie z jednej ścieżki — jeśli na tej ścieżce nie znajduje się szukana informacja — do ścieżki kolejnej tego samego cylindra.

Operacje czytania pozwalają przeczytać — z pamięci dyskowej lub bębnowej do pamięci operacyjnej — odrębne bloki danego zapisu, cały zapis danych lub bloki organizacyjne danej ścieżki (tj. adres ścieżki i zapis Zo). W czasie czytania automatycznie wykrywa się ewentualne błędy, a także wykrywa się koniec ciągu zapisów danych. Koniec ten określają bajty DD równe zeru w polu „licznik" danego zapisu. Po przeczytaniu zapisu końcowego urządzenie przekazuje do kanału cechę „urządzenie zakończyło pracę".

W repertuarze komend czytania znajduje się wyróżniona komenda o nazwie CZYTAJ PROGRAM PIERWOTNY. Komenda ta ma specjalne przeznaczenie i powoduje przeczytanie programu pierwotnego — tj. programu typu bootstrap — który przechowywany jest zawsze w zapisie Z_1 ścieżki zerowej nośnika. Jednostka sterująca ustawia — po otrzymaniu tej komendy — mechanizm dostępu na zerowej ścieżce zerowego cylindra, po czym czyta dane z zapisu Z_1 i przesyła je do kanału.

Operacje pisania pozwalają zapisać na nośniku dysku lub bębna pojedyncze bloki lub całe zapisy danych. W czasie zapisu tworzy się dwubajtową cechę kontroli cyklicznej, którą zapisuje się na końcu każdego bloku.

Komendy pisania są dwojakiego rodzaju: komendy formatowania zapisu i komendy zapisu danych. Komendy pierwszego rodzaju pozwalają przygotować ścieżki nowego nośnika do przyjęcia danych o wymaganym formacie. W szczególności istnieją komendy formatowania adresu ścieżki zapisu Zo, wymazania starej informacji ze ścieżki oraz komendy formatowania normalnych zapisów.

Komendy zapisu danych powodują zapisanie danych w polu „klucz" i w polu „dane", które to pola zostały wcześniej sformatowane przy pomocy komend formatowania zapisu.

Informacje o stanie urządzenia. Jednostka sterująca pamięciami o dostępie bezpośrednim wysyła do kanału przeważnie dwa bajty stanu, w odpowiedzi na przyjęcie kodu komendy lub przy zakończeniu danej operacji. Pierwszy bajt, tzw. bajt stanu podstawowego, zawiera najważniejsze informacje o stanie jednostki sterującej i samego urządzenia zewnętrznego, natomiast dalsze bajty stanu — zwane bajtami stanu szczegółowego — podają szczegółowe informacje dotyczące stanu urządzenia lub szczegółowo opisują zdarzenia, jakie miały miejsce w czasie wykonania operacji wejścia-wyjścia. Ilość bajtów stanu jest różna w różnych urządzeniach i zależy ona od typu urządzenia zewnętrznego. W przypadku pamięci o dostępie bezpośrednim ilość bajtów stanu jest stosunkowo duża (6 bajtów).

Bajt stanu podstawowego zawiera takie informacje, jak: urządzenie jest zajęte lub wolne, urządzenie zakończyło lub jednostka sterująca zakończyła zleconą pracę, urządzenie wykryło błąd (programowy lub techniczny), miał miejsce warunek porównania w operacjach szukania (modyfikator), przeczytany zapis był zapisem końcowym kartoteki itp. Natomiast bajty stanu szczegółowego zawierają dalsze informacje, takie jak: przesłana komenda nie została przyjęta do wykonania (np. niewłaściwy kod lub adres), wymagana jest interwencja operatora (np. nie włączone urządzenie lub nie zamknięta pokrywa jednostki dyskowej), wykryto błąd parzystości sprzętu, wykryto błąd w danych itp.

8.2. Pamięci taśmowe magnetyczne

Pamięci taśmowe Jednolitego Systemu EMC stanowią liczny zbiór jednostek pamięci taśmowej oraz jednostek sterujących pamięciami taśmowymi. Zestaw powyższych urządzeń oraz ich charakterystyka techniczna jest pokazana w tabeli 8.3. Jak z niej wynika, wszystkie pamięci taśmowe JS EMC stosują 9-ścieżkową taśmę magnetyczną o szerokości 12,7 mm. Również metoda zapisu jest jednakowa dla większości pamięci taśmowych; jest nią NRZI — czyli metoda zapisu bez powrotu do zera, w której binarna jedynka przedstawiana jest przez zmianę kierunku namagnesowania wzdłużnego, zaś zero — przez brak zmiany namagnesowania. Wyjątek stanowią pamięci taśmowe JS-5014 i JS-5015, które stosują metodę modulacji fazy, pozwalającą uzyskać większe gęstości zapisu (63 bity na mm).

Informacje na taśmie magnetycznej są zapisywane w postaci bloków rozdzielonych przerwami międzyblokowymi. Nominalna długość przerwy międzyblokowej wynosi 15,2 mm dla wszystkich

Tabela 8.3

Charakterystyka techniczna pamięci taśmowej JS EMC

Charakterystyka	JS-5010	JS-5012	JS-5014	JS-5015	JS-5016, 5021	JS-5017	JS-5019	JS-5022
	ZSRR	LRB	ZSRR	CSRS	NRD	ZSRR	PRL	CSRS
Szybkość przesuwu taśmy, m/s	2	2	2	4	1,5	2	3	4
Czas nawinięcia taśmy na szpuli standardowej, min	2,5	2,5	2,5	2	4,2 2,6	2,5	2,1	2,5
Gęstość zapisu, bit/mm	8 32	8 32	63	63	32	8 32	8 22, 32	8 32
Liczba ścieżek	9	9	9	9	9	9	9	9
Szybkość przesyłania danych, Kbajtów/s	64	64	126	240	48 96	64	30 82, 120	126
Nominalna przerwa międzyblokowa, mm	15,2	15,2	15,2	15,2	15,2	15,2	15,2	15,2
Wielkość taśmy:								
— długość maksymalna, m	750	750	750	750	750	750	750	750
— szerokość, mm	12,7	12,7	12,7	12,7	12,7	12,7	12,7	12,7
Szyfr jednostki sterującej (JS)	5511	5512	5514	5514	5516 5521	5517	5519	5517 5515

pamięci taśmowych JS EMC. Właściwość ta jest bardzo istotna, gdyż pozwala zachować zgodność taśm magnetycznych. Długość bloku — na taśmie magnetycznej — może być dowolna i określana jest przez dwubajtowy licznik słowa sterującego kanału. Niemniej rekomenduje się stosowanie bloków nie krótszych od 18 bajtów (głównie ze względów na niezawodność pracy).

Pliki danych oddziela się na taśmie magnetycznej specjalnym blokiem zwany markierem pliku (nie mylić ze znacznikiem odblaskowym początku i końca taśmy!). Markier ten jest zapisywany przez jednostkę sterującą w wyniku otrzymania od kanału specjalnej komendy o nazwie PISZ MARKIER PLIKU. W czasie czytania z taśmy w sposób techniczny odróżnia się markier pliku od pliku danych, a fakt napotkania tego markiera jest sygnalizowany kanałowi.

Kontrola poprawności zapisu i odczytu informacji jest dokonywana przez jednostkę sterującą poprzez wprowadzenie: poprzecznej kontroli nieparzystości jedynek, wzdłużnej kontroli parzystości oraz kodu cyklicznego. Kontrola wzdłużna polega na tym, że na końcu każdego bloku informacji dopisuje się dodatkowy bajt, którego bity ustala się w ten sposób, aby suma jedynek na danej ścieżce bloku była parzysta. W czasie zapisu informacji na taśmie magnetycznej generuje się cechy poprzecznej i wzdłużnej parzystości i zapisuje się je na taśmie, zaś w czasie odczytu określone układy ponownie tworzą te cechy kontrolne i porównują je z zapisanymi cechami. Wykrycie niezgodności cech jest sygnalizowane jako błąd. Ponadto wprowadzono kontrolę cykliczną polegającą na tym, że w czasie zapisu bloku oblicza się — za pomocą specjalnego algorytmu kodu cyklicznego — dodatkowy bajt kontrolny, który zapisuje się po ostatnim bajcie danych i przed bajtem wzdłużnej kontroli parzystości. W czasie czytania danych bajt kontroli cyklicznej jest ponownie tworzony i porównywany z bajtem zapisanym na taśmie magnetycznej. Jeśli porównanie nie wykaże zgodności tych bajtów, sygnalizuje się błąd wykryty przez kontrolę cykliczną. Kontrola cykliczna pozwala na poprawienie błędów powstałych na jednej ścieżce w czasie czytania. Jeśli jednostka sterująca wykryje błąd w czasie pierwszego czytania danego bloku, to zapoczątkowane zostaje drugie czytanie tego bloku, podczas którego neguje się przeczytany stan przekłamanych bitów danej ścieżki. Numer ścieżki zawierającej błędy określają specjalne układy. Kontrola cykliczna jest bardzo istotna, gdyż pozwala automatycznie poprawiać dowolną ilość błędów powstałych na

Tabela 8.4

Repertuar komend pamięci taśmowych JS EMC

Nazwa komendy	Kod komendy
Komendy podstawowe	
— czytaj (blok danych)	02
— czytaj wstecz	0C
— pisz (blok danych)	01
— sprawdź wejście-wyjście (czyta się bajt stanu podstawowego)	00
— czytaj stan szczegółowy	04
Komendy sterujące (brak przesyłania danych)	
— przewiń taśmę do początku	07
— przewiń i rozładuj taśmę	0F
— przeskocz blok do przodu	37
— przeskocz blok wstecz	27
— przeskocz plik do przodu	3F
— przeskocz plik wstecz	2F
— pisz markier pliku	1F
— kasuj odcinek taśmy	17
Komendy ustawienia trybu pracy i komendy pomocnicze	
— włącz układy korekcji	1B
— ustaw tryb diagnostyki (w tym trybie nie zapisuje się bitów parzystości)	0B
— bieg jałowy	03
— ustaw gęstość zapisu 8 znaków/mm	FB
— ustaw gęstość zapisu 32 znaki/mm	CB
— ustaw gęstość zapisu 63 znaki/mm	C3
— rezerwuj jednostki pamięci	F4
— zwolnij jednostki pamięci	D4

jednej ścieżce (w praktyce często błędy rozkładają się wzdłuż jednej ścieżki).

Komendy pamięci taśmowych. Program steruje pracą pamięci taśmowej za pomocą trzech grup komend: podstawowych, sterujących i ustawiania trybu pracy, pokazanych w tabeli 8.4. Ostatnie dwie komendy dotyczą jednostki sterującej wyposażonej w przełącznik dwukanałowy. Po otrzymaniu komendy REZERWUJ od danego kanału, przełącza się wszystkie jednostki pamięci taśmowej — podłączone do danej jednostki sterującej — do tego kanału aż do momentu zwolnienia rezerwacji. Powyższa możliwość pozwala traktować pamięć taśmową jako wspólną pamięć zewnętrzną dwóch systemów liczących (tj. dwuprocesorowych lub dwumaszynowych).

8.3. Czytniki i dziurkarki kart

W Jednolitym Systemie EMC istnieje wiele czytników kart dziurkowanych i dziurkarek kart (tabele 8.5 i 8.6). Urządzenia te różnią się między sobą parametrami technicznymi, zasadą działania oraz szczegółowymi rozwiązaniami konstrukcyjnymi.

Czytniki i dziurkarki kart pracują w oparciu o standardowe karty 80-kolumnowe i 12-wierszowe. Istnieje również możliwość wprowadzania informacji z 45- i 90-kolumnowych kart. Urządzenia kart dziurkowanych mają zintegrowaną (z mechanizmem) jednostkę sterującą, która może być podłączona zarówno do kanału selektorowego jak i do kanału multiplekserowego. Czytniki kart podłącza się zazwyczaj do kanału multiplekserowego, który może jednocześnie obsłużyć wiele takich urządzeń (w trybie multiplekserowym). Ponieważ czytniki kart nie są wyposażone w bufory, przydziela się im zazwyczaj w kanale multiplekserowym wysoki priorytet i dzięki temu — w przypadku przyjęcia zgłoszeń od kilku urządzeń — obsługiwane są w pierwszej kolejności przez kanał.

Czytanie danych odbywa się w jednym z dwóch trybów: czytanie w kodzie KPK-12 lub czytanie

Tabela 8.5

Charakterystyka czytników kart JS EMC

Charakterystyka	JS-6012	JS-6013	JS-6016
	ZSRR	ZSRR	CSRS
Szybkość czytania, kart/min	500	1000	1000
Typ kart (ilość kolumn)	45, 80	80	80, 90
Wielkość pojemnika kart:			
— podającego karty	1000	2000	2000
— odbierającego karty	1000	2000	2500
Zasada czytania (F — fotoelektryczna)	F	F	F
Ilość bajtów stanu	2	2	2
Zasada podania kart (S — synchroniczna, A — asynchroniczna, M — mechaniczna)	S	A	M
Kod wejściowy (standardowy)	KPK-12	KPK-12	KPK-12
Kod wyjściowy	DKOI	DKOI	DKOI
Moc pobierana, kV·A	1	1,5	0,75

Tabela 8.6

Charakterystyka dziurkarek kart JS EMC

Charakterystyka	JS-7010	JS-7012	JS-7013	JS-7014
	ZSRR	ZSRR	CSRS	CSRS
Szybkość dziurkowania, kart/min	100	250	250	90
Typ karty (ilość kolumn)	80	80	80	80
Wielkość pojemnika kart:				
— podającego karty	700	1200	1500	1500
— odbierającego karty	2×500	2×1200	2×1300	2×1300
Kod wejściowy	DKOI	DKOI	DKOI	DKOI
Kod wyjściowy	KPK-12	KPK-12	KPK-12	KPK-12
Ilość bajtów stanu	2	2	2	2
Moc pobierana, kV·A	1	2	2	1

w dowolnym kodzie kartowym. W przypadku czytania w kodzie KPK-12, który jest standardowym kodem kartowym w JS EMC, sprzęt jednostki sterującej dokonuje przekształcenia z kodu PKP-12 na standardowy wewnętrzny kod jednostki centralnej, tj. na tzw. dwójkowy kod wymiany informacji JS EMC — skrót rosyjski DKOI — odpowiadający kodowi EBCDIC i rozszerzony o dodatkowe znaki alfabetu rosyjskiego. Natomiast w przypadku czytania danych przedstawionych w innym kodzie nie zachodzi przekształcenie tego kodu. Wybór trybu czytania dokonywany jest za pomocą komend: CZYTAJ Z PRZEKSZTAŁCENIEM KODU i CZYTAJ BEZ PRZEKSZTAŁCENIA KODU. W czasie czytania karty z przekształceniem kodu przenosi się informację (znak) z jednej kolumny karty na jeden bajt pamięci operacyjnej. Natomiast w czasie czytania informacji w kodzie dowolnym przenosi się informację z jednej kolumny karty na dwa kolejne bajty. W pierwszym bajcie wpisuje się stan wierszy: 12, 11, 0, 1, 2 i 3, zaś w drugim bajcie stan wierszy: 4, 5, 6, 7, 8 i 9.

W czytnikach kart wprowadzono kontrolę poprawności przyjętego z kart kodu informacji (tylko w kodzie DKOI), kontrolę działania różnych układów elektronicznych oraz kontrolę pracy mechanizmu czytnika.

Dziurkarki kart JS EMC wyprowadzają informacje na kartach 80-kolumnowych. Jednostki sterujące tych dziurkarek mogą być podłączone zarówno do kanału selektorowego, jak i do kanału multiplekserowego. Wszystkie jednostki sterujące są wyposażone w układy przekształcające kod wewnętrzny jednostek centralnych JS EMC, tj. kod DKOI, na kod zewnętrzny (wyjściowy) KPK-12. Można również wyprowadzić informacje w kodzie binarnym, wtedy zawartość dwóch kolejnych bajtów pamięci operacyjnej (bity 2÷7 bajtu) zostaje wydziurkowana na jednej kolumnie karty: w pozycjach 12—3 kolumny dziurkowany zostaje pierwszy bajt, zaś drugi bajt w pozycjach 4—9 tej samej kolumny.

Dziurkarki kart są wyposażone, w odróżnieniu od czytników kart, w specjalne bufory do zapa-

Tabela 8.7

Drukarki wierszowe JS EMC

Charakterystyka	JS-7030	JS-7031	JS-7032	JS-7033	JS-7034	JS-7035	JS-7038
	ZSRR	NRD	ZSRR	PRL	CSRS	NRD	CSRS
Szybkość drukowania, wierszy/min	650–900	900	900	600–1100	900	600	750–1000
Liczba znaków w wierszu	128	156	128	120, 128, 160	132	120	160
Repertuar znaków	84	64	84	64	64	64	64
Kod wejściowy	DKOI	EBCDIC	DKOI	DKOI	DKOI	EBCDIC	DKOI
Bufor (na jeden wiersz)	TAK	TAK	TAK	TAK	TAK	TAK	TAK
Tryby pracy	multiplekserowy i selektorowy						
Sposób podania papieru	asynchroniczny						
Czas zmiany wiersza papieru, ms	10	6	10	4–5	8	6	8
Zasada drukowania	bęben wirujący						
Jednostka sterująca	zintegrowana z mechanizmem druku						

miętania zawartości dwóch kart. Dzięki temu kontakt dziurkarki z kanałem jest bardzo krótki.

8.4. Drukarki wierszowe

Drukarki wierszowe zajmują się szybkim wyprowadzaniem informacji z pamięci operacyjnej oraz drukowaniem tych informacji w postaci alfanumerycznej. Zbiór oraz charakterystyka techniczna drukarek wierszowych Jednolitego Systemu EMC pokazana jest w tabeli 8.7. Wszystkie drukarki wierszowe wyposażone są w bufor lokalny o pojemności jednego wiersza. Dzięki temu jednostka sterująca drukarki odbiera bardzo szybko — w wyniku otrzymania komendy DRUKUJ — dane z kanału do jej bufora, po czym zwalnia kanał i przystępuje do drukowania danych w sposób autonomiczny. Po zakończeniu wydruku jednego wiersza i przesunięciu papieru jednostka sterująca drukarki zgłasza jej gotowość — poprzez wysłanie cechy urządzenie zakończyło pracę — do przyjęcia kolejnego wiersza danych do druku.

8.5. Czytniki i dziurkarki taśmy

Czytniki i dziurkarki taśmy Jednolitego Systemu EMC (tabele 8.8 i 8.9) przystosowane są do pracy z taśmą 5- i 8-ścieżkową (niektóre czytniki także z taśmą 6- i 7-ścieżkową) i mogą być podłączone do dowolnej jednostki centralnej JS EMC. Czytniki i dziurkarki taśmy podłącza się zazwyczaj do kanału multiplekserowego, a pracować mogą zarówno w trybie multiplekserowym, jak i w trybie selektorowym.

Jednostki sterujące czytników i dziurkarek są zintegrowane z mechanizmami. W jednostkach sterujących czytników wprowadzono kontrolę poprawności ich pracy, obejmującą między innymi:
— kontrolę stanu taśmy,
— kontrolę regulacji mechanizmu,
— kontrolę parzystości (lub nieparzystości) czytanych znaków.

Program steruje pracą modułu czytnika taśmy za pomocą czterech podstawowych komend:
— czytaj blok danych według licznika; oznacza to, że czytanie kończy się, gdy licznik bajtów — ze słowa sterującego kanału — osiągnie zero;
— czytaj blok danych do znaku końca; w tym przypadku czytanie kończy się, gdy jednostka sterująca napotka wyróżniony „znak kojca" lub gdy licznik bajtów osiągnie stan zero; znakiem końca może być dowolny znak, którego stan został nastawiony na przełącznikach pulpitu operatora;
— sprawdź wejście-wyjście; operacja ta powoduje przeczytanie stanu podstawowego urządzenia, który zawiera takie informacje, jak:

Tabela 8.8

Czytniki taśmy dziurkowanej JS EMC

Charakterystyka	JS-6022			JS-7902*
	ZSRR	PRL	WRL	NRD
Szybkość czytania, znaków/s	1500	1800	1500	1000
Liczba ścieżek informacyjnych	5, 6, 7, 8	5/8	5, 6, 7, 8	5, 6, 7, 8
Zasada czytania	fotoelektryczna			
Rodzaj pracy	asynchroniczny			
Tryb przesyłania	multiplekserowy lub selektorowy			
Pobór mocy, kV·A	0,8	0,5	0,4	1

* Stacja ta jest wyposażona w mechanizm czytnika taśmy papierowej typu CT 1001 polskiej produkcji

Tabela 8.9

Dziurkarki taśmy papierowej JS EMC

Charakterystyka	JS-7021	JS-7022	JS-7902*
	PRL	ZSRR	NRD
Szybkość dziurkowania, znaków/s	150	150	100
Liczba ścieżek informacyjnych	5, 8	5, 8	5, 6, 7, 8
Rodzaj pracy	asynchroniczny		
Rodzaj perforacji	elektromechaniczny		
Rodzaj przesyłania	multiplekserowy lub selektorowy		
Pobór mocy, kV·A	0,8	0,8	1

* Stacja ta jest wyposażona w mechanizm dziurkarki taśmy typu D 102 polskiej produkcji

urządzenie zajęte, kanał zakończył, urządzenie zakończyło pracę, błąd w urządzeniu itp.;
— czytaj bajt stanu szczegółowego urządzenia; bajt ten zawiera takie informacje, jak: nie przyjęto operacji do wykonania, wymagana jest interwencja operatora, błąd danych itp.

Uwaga. Kody komend czytania informacji z taśmy papierowej mogą zawierać tzw. bity modyfikujące, które bliżej precyzują sposób czytania (np. czytanie z przekształceniem kodu wejściowego lub bez przekształcenia tego kodu).

Praca dziurkarek taśmy sterowana jest za pomocą 4 podstawowych komend:
— pisz blok danych (według wskazań licznika),
— pisz blok danych oraz znak końca (nastawiony na pulpicie operatora),
— sprawdź wejście-wyjście,
— czytaj bajt stanu szczegółowego.

Urządzenie JS-7902 jest samodzielną „stacją taśmy", która w jednej konstrukcji zawiera dwa czytniki taśmy i jedną dziurkarkę taśmy, przy czym jednostka sterująca jest wspólna dla wszystkich trzech urządzeń. Każde z tych urządzeń ma swój własny adres, tzn. jest odrębnym podkanałem. Oczywiście w danym przedziale czasowym może pracować tylko jedno urządzenie.

8.6. Moduły bezpośredniej komunikacji operatora z maszyną

W Jednolitym Systemie EMC opracowano kilka typów modułów bezpośredniej komunikacji operatora z maszyną (tabela 8.10), które wykorzystuje się głównie jako pulpity operatorskie. Pulpity te pozwalają na bezpośrednie wprowadzanie i wyprowadzanie informacji występujących we współpracy operator — maszyna cyfrowa. Ponadto moduły bezpośredniej komunikacji mogą być wykorzystane jako lokalna końcówka dialogowa, jako zdalny punkt sterowania maszyną itp. Moduły bezpośredniej komunikacji podłącza się zazwyczaj do kanału multiplekserowego i pracować wówczas mogą zarówno w trybie multiplekserowym jak i selektorowym. Moduły te spełniają w zasadzie jednakowe funkcje i różnią się głównie typem mechanizmu (np. Consul 260, Marica 141, Soemtron 34-529, DZM-180) oraz szczegółowymi rozwiązaniami konstrukcyjnymi. Również szybkość drukowania wszystkich mechanizmów jest jednakowa i wynosi 10 znaków/s, z wyjątkiem polskiego urządzenia JS-7186, które drukuje z szybkością 180 znaków/s. W związku z tym urządzenie JS-7186 — które jest mozaikową drukarką typu szeregowego — może mieć szersze zastosowanie (np. jako drukarka do wyprowadzania danych w zestawach małych maszyn lub w punktach abonenckich).

8.7. Urządzenie wprowadzania i wyprowadzania informacji w postaci graficznej

W Jednolitym Systemie EMC opracowano szereg urządzeń wprowadzania i wyprowadzania informacji w postaci rysunków i w postaci tekstów (tabele 8.11, 8.12 i 8.13). Do tej grupy urządzeń należą różne monitory ekranowe i koordynatografy (pisaki x-y). Grupa monitorów ekranowych składa się z dwóch typów:
— monitory ekranowe alfanumeryczne (alfaskopy), do których należą urządzenia: JS-7061, JS-7063 (wersja WRL), JS-7063 (wersja CSRS) i JS-7066; monitory alfanumeryczne wprowadzają i wyprowadzają tylko informacje tekstowe i numeryczne;
— monitory ekranowe graficzne (grafoskopy), do których należą urządzenia JS-7064 i JS-7065; grafoskopy mogą przedstawiać informacje dowolnego typu.

Grupa opracowywanych alfaskopów i grafoskopów oraz całych systemów monitorowych JS EMC jest bardzo bogata. W tej części opisu zostaną podane informacje dotyczące tylko tych urządzeń, które zostały wdrożone do produkcji lub są w stadium wdrożenia.

Tabela 8.11

Monitory ekranowe alfanumeryczne JS EMC

Charakterystyka	JS-7061 WRL	JS-7063 WRL	JS-7063 CSRS	JS-7066 ZSRR
Pojemność ekranu, znaków	1024, 960	1024, 960	960	960, 480, 240
Format ekranu	16×64	16×64	12×80	12×80
Liczba wierszy x liczba znaków w wierszu	12×80	12×80	24×40 15×64	6×80 12×40 6×40
Repertuar znaków	64	96	96	96
Możliwość wykonania kopii	NIE	TAK	TAK	TAK
Pobór mocy, kV·A				0,25

Moduły bezpośredniej komunikacji operatora z maszyną

Tabela 8.10

Charakterystyka	JS-7070 ZSRR	JS-7071 CSRS	JS-7073 NRD	JS-7074 LRB	JS-7186* PRL
Szybkość drukowania, m/s	10	10	10	10	180
Repertuar znaków	93	93	92	93	64—128
Liczba znaków w wierszu	106	106	117	123	132; 158
Pamięć buforowa (ilość znaków)	—	—	—	—	256
Typ mechanizmu	Consul	Consul	Soemtron	Marica	DZM-180
Kod wymiany informacji	DKOI	DKOI	DKOI	DKOI	DKOI
Pobór mocy, kV·A	0,22	0,25	0,25	0,25	0,35

* Jest to drukarka szeregowa polskiej produkcji (moduł mechanizmu), znana również pod nazwą DZM-180

Tabela 8.12

Monitory ekranowe graficzne JS EMC

Charakterystyka	JS-7064	JS-7065
	ZSRR	WRL
Wymiar roboczy ekranu, mm	250×250	360×360
Częstotliwość generacji obrazów, obrazów/s	50	50
Wymiar rastru, punktów	1024×1024	1024×1024
Szybkość kreślenia, mm/μs	3	–
Liczba znaków w wierszu (wymiar normalny i duży)	74, 49	–
Liczba wierszy	52, 35	–
Liczba znaków na ekranie	2100	2000
Wielkość pamięci buforowej, bajtów	2×4096	–
Kod informacji	DKOI	ASCII
Pobór mocy, kV·A	2	–

Tabela 8.13

Koordynatografy JS EMC

Charakterystyka	JS-7051	JS-7052	JS-7053	JS-7054
	ZSRR	ZSRR	ZSRR	CSRS
Szybkość rysowania, mm/s	50	200	150	100
Krok elementu rysującego, mm	0,05	0,1 0,05	0,1	0,05
Typ koordynatografu (B — bębnowy, P — płaski)	P	B	B	P
Ilość kolorów	3	3	3	4
Wielkość rysunku, mm	1050×1000	380×600	841×1600	1200×1600
Orientacja znaków (kąty kierunków)	co 22,5°	0°; 90°; 180°	co 22,5°	co 22,5°
Typ kreślonej linii (C — ciągła, P — punktowa, K — kreskowa)	CPK	C	CPK	CPK
Liczba symboli	253	64	253	96
Skala kreślenia	1 : 1 1 : 2 2 : 1	1 : 1 1 : 2 2 : 1	1 : 1 1 : 2 2 : 1	1 : 1 1 : 2 2 : 1
Pobór mocy, kV·A	2	1,2	1,5	–

Koordynatografy (lub pisaki) kreślą rysunki, teksty alfanumeryczne i inne symbole graficzne na papierze. W Jednolitym Systemie EMC opracowano zarówno koordynatografy typu bębnowego (JS-7052 i JS-7053), jak i typu płaskiego (JS-7051 i JS-7054). W koordynatografach pierwszego typu stosuje się papier w formie zwoju, zaś drugiego typu — arkusze płaskie.

Alfaskopy. Alfaskopy są wykorzystywane — w różnych systemach wyszukiwania i przetwarzania informacji — w charakterze punktów dialogowych. Mogą one występować jako samodzielne punkty systemu lub jako grupa alfaskopów podłączonych do wspólnej jednostki sterującej, która może znajdować się w pewnym oddaleniu od samych alfaskopów (np. do 600 metrów). Do typowej struktury alfaskopu wchodzą takie bloki funkcjonalne, jak: ekran, pamięć buforowa do przechowywania informacji związanej z jednym obrazem, generator znaków, środki do formowania informacji na ekranie oraz środki do czytania informacji z klawiatury, skąd operator wprowadza żądane dane i zadaje różne operacje sterujące. Zazwyczaj do alfaskopu można również podłączyć elektryczną maszynę do pisania, na której wypisuje się informacje będące na ekranie monitora. Istniejące alfaskopy mają różną pojemność ekranu; wynosi ona: 240, 480, 960 i 1024 znaki, przy czym tekst ekranu może być rozmieszczony na 6, 12 lub 16 wierszach, a każdy wiersz może składać się z 40, 80 lub 64 znaków.

Alfaskopy mogą wykonywać szereg funkcji w trybie autonomicznym, tj. bez komunikacji z maszyną cyfrową. Mogą zwłaszcza czytać informacje z dowolnej części ekranu, wymazywać informacje z części lub z całego ekranu, zapisywać nowe informacje na ekranie, przesuwać informacje na ekranie, ustawiać wskaźnik tabulacji itp. Funkcje autonomiczne są inicjowane za pomocą specjalnych klawiszy znajdujących się na klawiaturze monitora.

Repertuar znaków alfaskopów jest bogaty i wynosi 96 symboli. Repertuar ten zawiera między innymi litery łacińskie, litery alfabetu rosyjskiego, cyfry, znaki specjalne i sterujące. We wszystkich alfaskopach Jednolitego Systemu EMC przyjęto jednakowe znaki sterujące:

— początek komunikatu — oznaczający początek pola ekranu, do którego wprowadza się informacje z klawiatury w wyniku wydania komendy CZYTAJ Z KLAWIATURY;
— koniec komunikatu — oznaczający koniec pola ekranu ładowanego z klawiatury lub koniec pola do wypisania na maszynie (tj. koniec kopii);
— nowy wiersz;
— błąd — oznaczający, że wystąpił kod nie należący do repertuaru przyjętych kodów.

Przedstawione w tabeli 8.11 monitory ekranowe alfanumeryczne podłączone są do kanału multiplekserowego lub selektorowego. Maksymalna szybkość przesyłania informacji do lub z alfaskopu wynosi około 100 Kb/s. W operacjach pisania dane są przesyłane z pamięci operacyjnej maszyny do pamięci buforowej alfaskopu (według wskazań kursora). Długość bloków przesyłanych określana jest przez licznik bajtów słowa sterującego kanału. Natomiast w operacjach czytania dane są przesyłane z pamięci buforowej alfaskopu do pamięci operacyjnej maszyny. Operacje czytania mogą być wykonywane w dwóch trybach:

— czytaj dane wprowadzone z klawiatury,
— czytaj pełny ekran.

W pierwszym trybie czyta się tylko to pole danych, które zaczyna się symbolem „początek komunikatu" i kończy się symbolem „koniec komunikatu", przy czym w tej operacji nie przesyła się danych zawartych pomiędzy symbolem „nowy wiersz" a końcem wiersza. Długość bloku przesyłanych danych określana jest przez licznik bajtów lub przez fizyczny koniec ekranu. Normalnie program zapoczątkowuje operację pierwszego trybu w odpowiedzi na naciśnięcie klawisza WPROWADŹ (który powoduje przerwanie programu).

Natomiast operacja „czytaj pełny ekran" powoduje przeczytanie wszystkich danych alfaskopu, tj. od pierwszej do ostatniej pozycji ekranu (niezależnie od obecności symboli sterujących). Operacja ta również kończy się po wyzerowaniu się licznika bajtów lub po napotkaniu końca ekranu.

Alfaskop z szyfrem JS-7066 nie jest bezpośrednio podłączany do kanału maszyny cyfrowej, lecz do grupowej jednostki sterującej JS-7566. Alfaskopy JS-7066 mogą być oddalone od ich jednostki sterującej do 600 metrów. Do jednostki sterującej JS-7566 można podłączyć maksymalnie 16 alfaskopów typu JS-7066, przy czym łączna pojemność pamięci buforowej dla wszystkich alfaskopów nie może przekroczyć 3840 znaków.

Grafoskopy. Na ekran grafoskopu można wprowadzać informacje — w postaci rysunków lub informacji tekstowej — za pomocą pióra świetlnego i klawiatury alfanumerycznej. Można również wyprowadzać informacje — graficzne lub tekstowe — z pamięci operacyjnej maszyny na ekran grafoskopu. Dzięki temu grafoskop jest uniwersalnym punktem dialogowym w dynamicznej współpracy człowiek — maszyna. Grafoskopy znajdują zastosowanie zwłaszcza w tzw. komputerowo-wspomaganych systemach projektowania i konstruowania — w płaszczyźnie lub przestrzeni trójwymiarowej — oraz w pracach naukowo-badawczych i modelowaniu procesów dynamicznych.

Programowe sterowanie pracą grafoskopu odbywa się za pomocą licznej grupy komend (ponad 20), które określają różne sposoby pracy grafoskopu. Można zadać następujące sposoby pracy grafoskopu:

— wyświetlanie punktów lub wektorów o określonych bezwzględnych współrzędnych rastru,
— wyświetlanie punktów i wektorów pewnego tylko otoczenia względem promienia wodzącego,
— formowanie i wyprowadzanie na ekranie znaków dwóch różnych wielkości (podstawowej i powiększonej) oraz założenie ewentualnej ochrony przed wymazaniem tych znaków (przez wprowadzenie znaków z klawiatury),
— praca z piórem świetlnym itp.

Koordynatografy. Koordynatografy JS EMC kreślą rysunki i znaki alfanumeryczne na papierze różnych formatów, których wielkość maksymalna podana jest w tabeli 8.13. Koordynatografy te znajdują zastosowanie przy automatycznym sporządzaniu różnych schematów konstrukcyjnych, elektrycznych, funkcjonalnych, map meteorologicznych itp.

W celu uproszczenia programów sterujących pracą koordynatorów wprowadzono bogaty i efektywny zbiór komend, takich jak: wybór elementu rysującego (1, 2, 3 lub 4), linia ciągła, linia punktowa, linia kreskowa, interpolacja liniowa, interpolacja typu okręgu (rysowana zgodnie lub przeciwnie w stosunku do ruchu wskazówki zegara), praca w absolutnych współrzędnych, ruch przyrostowy, wykreślanie symbolu, wybór skali (3,6 lub 12 mm dla znaku) itp.

Komendy sterujące pracą elementu rysującego wybierają jeden z możliwych elementów i ustawiają go w pozycji pisania. Wybór elementu rysującego jest związany zazwyczaj z wyborem koloru rysowania lub z wyborem grubości linii kreślenia. Z kolei komendy interpolacji określają sposób poruszania się elementu rysującego, tzn. po linii prostej lub po łuku okręgu (odpowiednio dla interpolacji liniowej lub dla interpolacji typu okręgu). Oczywiście łącznie z komendami interpolacji przekazuje się z pamięci operacyjnej ciąg danych określających współrzędne ruchu elementu rysującego. Współrzędne ruchu mogą być zadawane w postaci wielkości bezwzględnych lub w postaci przyrostu względem pozycji, w której znajduje się w danej chwili element kreślący. Jeśli docelowy punkt, do którego powinien dojść element kreślący, zadawany jest w bezwzględnych współrzędnych X i Y, to przeznacza się po dwa bajty do przedstawiania wielkości X i Y (w krokach ruchu elementu kreślącego). W praktyce wykorzystuje się 15 bitów tych bajtów, co pozwala zapisywać wielkości od 0 do 1638 mm (dla kroku 0,05 mm). Natomiast w przypadku pracy w trybie przyrostowym najstarszy bit każdej współrzędnej X i Y określa znak współrzędnej (1 — oznacza minus, 0 — plus), czyli kierunek ruchu.

Komenda WYKREŚLANIE SYMBOLU powoduje, że elektronika sterująca koordynatografu traktuje przyjęty ciąg danych jako kody symboli, których kształt należy narysować na papierze. Rysowanie symboli w JS-7052 i JS-7054 odbywa się za pomocą wbudowanego tzw. generatora znaków, który automatycznie kreśli dany symbol znaku. Natomiast koordynatografy JS-7051 i JS-7053 nie mają sprzętowego generatora, dlatego rysowanie symbolu odbywa się pod kontrolą odpowiedniego podprogramu, opisującego graficzny obraz symbolu na odpowiedniej siatce, który to program znajduje się w czasie wykonania w pamięci buforowej (4 Kbajty) koordynatografu. Oczywiście wybór odpowiedniego podprogramu jest dokonywany zgodnie z kodem symbolu. Repertuar rysowanych symboli pod kontrolą programów jest bardzo bogaty i wynosi 253 symboli (tabela 8.13).

Koordynatografy JS EMC mogą pracować dwiema metodami:

1) metodą pracy autonomicznej, tzn. bez bezpośredniej współpracy z maszyną cyfrową; w metodzie tej dane wejściowe do koordynatografu są zawczasu przygotowane przez maszynę na taśmie magnetycznej lub papierowej, a szpulę tych danych wkłada się odpowiednio do jednostki pamięci taśmowej (tj. do urządzenia JS-7050) lub do czytnika taśmy, które to urządzenia wchodzą do konfiguracji koordynatografu;
2) metodą pracy nieautonomicznej, tzn. w bezpośredniej współpracy z kanałem jednostki centralnej maszyny; w metodzie tej koordynatograf może być podłączony zarówno do kanału selektorowego, jak i do multiplekserowego.

8.8. Urządzenia teleprzetwarzania danych

W Jednolitym Systemie EMC, podobnie jak i w innych współczesnych systemach komputerowych, środki techniczne występujące w zdalnej współpracy człowiek—maszyna lub urządzenie końcowe—centralna EMC zostały podzielone na kilka grup funkcjonalnych:

— centralna EMC wraz z kanałami wejścia—wyjścia,
— multipleksery przekazywania danych (MPD),
— środki łączności,
— urządzenia końcowe (lub punkty abonenckie).

Multipleksery przekazywania danych uczestniczą w przesyłaniu danych pomiędzy pamięcią operacyjną centralnej EMC i urządzeniami końcowymi. Multipleksery te podłączone są z jednej strony do kanału wejścia—wyjścia, a z drugiej strony do środków łączności (np. modemy, łącza telefoniczne, łącza telegraficzne i inne) i za ich pośrednictwem do urządzeń końcowych. Główną funkcją MPD jest zorganizowanie jednoczesnej pracy wielu urządzeń końcowych współpracujących z jedną centralną EMC. Multiplekser MPD może tylko przekazywać dane pomiędzy wieloma urządzeniami końcowymi i centralną EMC lub też przeprowadzać określone procesy przetwarzaniowe na przesyłanych danych. W tym ostatnim przypadku MPD jest zazwyczaj małą i niezależną EMC, przystosowaną do współpracy z liniami transmisyjnymi. Dane przesyłane na drodze KANAŁ—MPD podawane są w postaci równoległej (bajtami), zaś na drodze MPD—środki łączności — w postaci szeregowej, tj. bit po bicie.

Urządzeniem końcowym (lub punktem abonenckim) może być proste urządzenie typu pytanie—odpowiedź (np. dalekopis lub monitor ekranowy), zwane również punktem dialogowym, lub rozbudowane urządzenie wprowadzania i wyprowadzania danych, tzw. zdalna stacja wprowadzania i wyprowadzania danych, albo też normalna EMC (przeważnie mała lub bardzo mała), zwana zdalną EMC. Jeśli punktem abonenckim jest maszyna cyfrowa, to można przeprowadzać szereg prostych operacji na danych przed ich przesłaniem do centralnej EMC, głównie w celu redukcji tych danych. Natomiast wyniki odebrane z centralnej EMC można zaopatrywać w dodatkowe dane oraz w rozmaite opisy, które przechowywane są tylko w banku zdalnej EMC. Niektóre małe EMC Jednolitego Systemu (np. JS-1010) mają wbudowane środki do współpracy z liniami transmisyjnymi, inne zaś tych środków nie mają i dlatego podłącza się je do linii transmisji (np. telefonicznej) poprzez multiplekser typu MPD.

Punkty abonenckie mogą być podłączane do multipleksera MPD dwoma sposobami:

— połączenie typu punkt—punkt (zwane też jednopunktowym), gdzie do określonego podkanału łączności MPD podłącza się tylko jeden punkt abonencki;
— połączenie wielopunktowe, gdzie do jednego podkanału łączności MPD podłącza się wiele punktów abonenckich.

Ze względu na czas zajętości linii łączności przez punkt abonencki wyróżnia się też dwie metody pracy:

— praca przez łącza komutowane, gdzie punkt abonencki nie zajmuje przez cały czas linii łączności, lecz linia ta zostaje mu przydzielona tylko na czas przekazywania danych; w tym przypadku punkt abonencki jest jednym z wielu użytkowników danej sieci łączności;
— praca przez łącza dzierżawione (lub wydzielone), gdzie punkt abonencki przez cały czas zajmuje daną linię łączności, niezależnie od tego, czy aktualnie przesyła dane, czy też ich nie przesyła.

Z punktu widzenia programowej organizacji współpracy ze zdalnym urządzeniem wyróżnia się tylko multiplekser MPD i punkt abonencki. Natomiast cały szereg pośrednich urządzeń łączności nie jest widziany przez programistę. Komendy przesyłane przez programy sterujące są różne i dotyczą różnych poziomów. Niektóre komendy są interpretowane i realizowane przez kanał we—wy, zaś inne przez multiplekser MPD lub przez punkt abonencki (PA). Współpraca MPD—PA, tj. sposób nawiązania bezpośredniego kontaktu pomiędzy MPD i PA lub zaprzestanie tego kontaktu, jest zależna od typu połączenia punktu abonenckiego z MPD (jednopunktowe lub wielopunktowe) i organizowana jest podobnymi metodami, jakie spotyka się w innych współczesnych systemach teleprzetwarzania.

Poniżej podano krótką charakterystykę urządzeń teleprzetwarzania w JS EMC, które to urządzenia zostały opracowane i znajdują się w produkcji lub w stadium wdrożenia do produkcji. Trzeba przy tym zaznaczyć, że rozwój sprzętu i systemów zastosowań teleprzetwarzania w JS EMC znajduje się na początkowym etapie. W związku z tym bardzo liczna grupa urządzeń teleprzetwarzania jest jeszcze w stadium opracowania lub w stadium prób i dlatego brak dokład-

Tabela 8.14

Multipleksery przekazywania danych JS EMC

Charakterystyka	JS-8400 ZSRR	JS-8402 ZSRR	JS-8403 ZSRR	JS-8410 WRL
Liczba kanałów łączności (linii)	do 15	do 176	do 4	do 32
Możliwość pracy z dwoma kanałami we-wy	TAK	TAK	TAK	
Typy kanałów łączności (K — komutowane, W — wydzielone)	KW	KW	KW	KW
Szybkość transmisji danych kanałów łączności, bitów/s	do 4800	50, 100, 200, 600, 1200, 2400, 4800	50, 100, 600, 1200, 2400, 4800, 48 000	50, 75, 100, 200, 600, 1200
Pamięć buforowa, bajtów	—	4096	—	256
Pobór mocy, kV·A	2	9	9	0,4

nych materiałów na temat tych urządzeń (dotyczy to szczególnie programowanych multiplekserów MPD, różnych zdalnych systemów monitorowych, inteligentnych urządzeń końcowych itp.).

Multipleksery przekazywania danych. Grupa multiplekserów JS EMC (tabela 8.14) składa się z następujących urządzeń: JS-8400 (MPD-1A), JS-8402 (MPD-2), JS-8403 (MPD-3), JS-8410 (MPD-1). Multipleksery te organizują przesyłanie danych pomiędzy pamięcią operacyjną, a punktami abonenckimi zgodnie z komendami otrzymanymi od jednostki centralnej maszyny. Komendy powodują wykonanie przez multiplekser takich operacji, jak: zarezerwowanie MPD dla określonego kanału (operacja ta dotyczy tych MPD, które mają wbudowany dwukanałowy przełącznik), anulowanie rezerwacji MPD, ustawienie MPD i środków łączności w stanie roboczym, wybranie numeru punktu abonenckiego, czytanie ciągu danych z kanału łączności do pamięci operacyjnej (tj. do napotkania jednego ze znaków sterujących koniec bloku, koniec tekstu, koniec transmisji), pisanie ciągu danych do kanału łączności (tj. do punktu abonenckiego), czytanie bajtu stanu szczegółowego urządzeń MPD i PA itp.

Multiplekser JS-8400 organizuje jednoczesną i niezależną pracę 15 kanałów (linii) łączności, przez które mogą przekazywać lub odbierać dane różne punkty abonenckie oraz dowolna EMC Jednolitego Systemu wyposażona w multiplekser typu MPD. Podstawowa charakterystyka multiplekserów przekazywania danych zawarta jest w tabeli 8.14.

Punkty abonenckie. Punkty abonenckie JS EMC mają różne szybkości przesyłania — od 50 do 2400 bitów/s — oraz wyposażone są w różnorodne urządzenia wprowadzania i wyprowadzania informacji, pokazane w tabeli 8.15. Punkty abonenckie, przesyłające dane za pośrednictwem linii telefonicznych, stosują modemy o różnych parametrach. W nomenklaturze JS EMC znajdują się modemy o następujących szybkościach transmisji (bity na sekundę): 200, 1200, 2400, 4800, 48000 oraz modemy do równoległej transmisji danych.

Tabela 8.15a

Punkty abonenckie w JS EMC

Charakterystyka	JS-8501 AP-1 LRB	JS-8502 AP-2 WRL, ZSRR	JS-8503 AP-3 LRB, WRL	JS-8504 AP-4 LRB, ZSRR	JS-8505 AP-5 NRD
Szybkość transmisji danych, bitów/s	50; 75 100; 200	100	600 1200	1200	200 600 1200
Wyposażenie w urządzenia:					
— elektryczna maszyna do pisania	+	+	+	+	+
— czytnik taśmy	+	+	+	+	+
— dziurkarka taśmy	+	+	+	+	+
— czytnik kart	+	+	—	+	—
— drukarka	—	—	—	—	+
— monitor ekranowy	—	—	—	—	—
— dziurkarka kart	+	+	—	—	—
— taśma kasetowa	—	—	—	—	+
— klawiatury alfanumeryczne (związane z monitorami)	—	—	—	—	+

Tabela 8.15b

Punkty abonenckie w JS EMC (cd.)

Charakterystyka	JS-8511 AP-11 ZSRR	JS-8514 AP-14 PRL	JS-8561 AP-61 WRL, ZSRR	JS-8563 AP-63 ZSRR	JS-8570 AP-70 WRL, ZSRR
Szybkość transmisji danych, bitów/s	1200 2400	1200 2400	1200 2400 4800	1200 2400	100
Wyposażenie w urządzenia:					
— elektryczna maszyna do pisania	—	+	+	+ 2	+
— czytnik taśmy	—	+	—	—	—
— dziurkarka taśmy	—	+	—	—	—
— czytnik kart	+	+	—	—	—
— drukarka	+	+	—	—	—
— monitor ekranowy	—	+	+	+16	—
— dziurkarka kart	+	—	—	—	—
— taśma magnetyczna (kasetowa)	—	+	—	—	—
— klawiatury alfanumeryczne	—	—	—	+16	—

Plus (+) w powyższych tabelach oznacza, że odpowiednie urządzenie może być podłączone do danego punktu abonenckiego. Liczba +16 oznacza, że maksymalna ilość urządzeń danego typu może wynosić 16. Natomiast minus (−) wskazuje, że danego urządzenia nie można standardowo podłączyć do odpowiedniego punktu abonenckiego.

9. Systemy wieloprocesorowe i wielomaszynowe

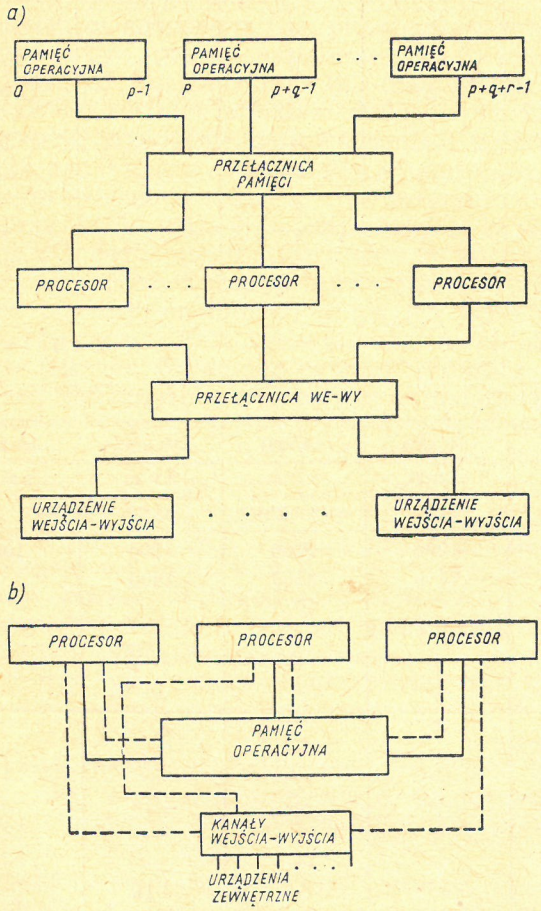
9.1. Wprowadzenie

Większość seryjnie produkowanych współczesnych komputerów składa się z jednego procesora i pewnej ilości bloków pamięci operacyjnej oraz z kilku kanałów, przez które wprowadza się (lub wyprowadza) dane z urządzeń zewnętrznych.

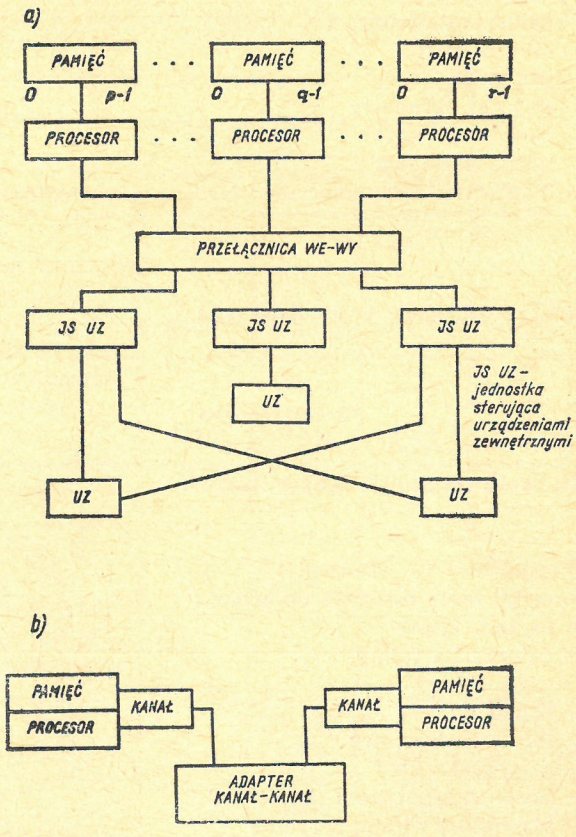
Komputery (lub maszyny cyfrowe) mające jeden procesor — tj. jedną jednostkę przetwarzającą — nazywamy komputerami jednoprocesorowymi. Komputery te wykonują jeden program w danej chwili, czyli realizują jeden niezależny ciąg instrukcji (lub rozkazów).

Komputery zawierające co najmniej dwa niezależnie pracujące procesory, z których każdy ma dostęp do wspólnej dla nich i jednolicie adresowanej pamięci operacyjnej, nazywa się komputerami wieloprocesorowymi (rys. 9.1). Zatem komputery wieloprocesorowe mogą jednocześnie wykonywać kilka programów na raz (np. tyle, ile jest procesorów), przy czym programy te wraz z danymi przechowywane są we wspólnej i jednolicie adresowanej pamięci operacyjnej. Oznacza to, że każdy z procesorów ma dostęp do całej wspólnej pamięci (z wyjątkiem małych obszarów przyporządkowanych ściśle poszczególnym procesorom). W celu uniknięcia częstych konfliktów występujących przy odwoływaniu się do pamięci operacyjnej dzieli się całą wspólną pamięć na mniejsze bloki mające niezależne układy odczytu i zapisu informacji.

Często w praktyce stosuje się także inne systemy, zwane systemami wielomaszynowymi lub wielokomputerowymi (rys. 9.2). Określa się tak systemy zawierające dwie lub więcej niezależnych maszyn cyfrowych (lub



Rys. 9.1. Przykłady struktur systemów wieloprocesorowych: a) ogólna struktura systemu wieloprocesorowego, b) struktura systemu wieloprocesorowego z bezpośrednim dostępem do pamięci operacyjnej

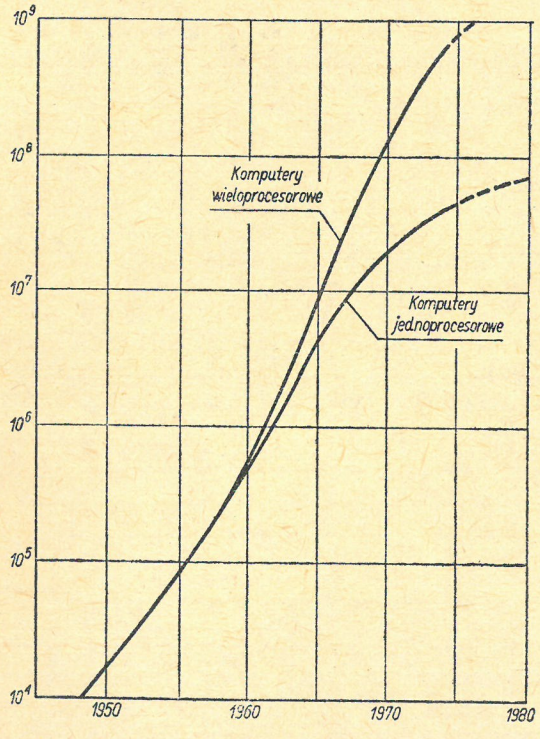


Rys. 9.2. Przykłady struktur systemów wielomaszynowych: a) ogólna struktura systemu wielomaszynowego, b) struktura systemu dwumaszynowego oparta o adapter kanał-kanał (występująca w EMC Jednolitego Systemu)

komputerów), z których każda ma własną oddzielną pamięć (o własnej adresacji) i które połączone są ze sobą poprzez różne układy współpracy (np. poprzez kanały wejścia—wyjścia). Zatem poszczególne EMC wykonują niezależne programy i każda z nich jest pod kontrolą własnego systemu sterującego. Część urządzeń zewnętrznych jest wspólna dla wszystkich EMC, inna zaś część tych urządzeń podłączona jest tylko do jednej EMC.

W praktyce „wspólnymi" lub „dzielonymi" urządzeniami zewnętrznymi są różne typy pamięci o bezpośrednim dostępie (np. dyski), w których organizuje się wspólne banki danych. Często dwie różne EMC mogą szybko wymieniać dane za pośrednictwem własnych kanałów (np. poprzez tzw. adapter kanał-kanał).

Pojawienie się systemów komputerowych o wieloprocesorowej lub wielomaszynowej organizacji podyktowane było różnymi wymogami technicznymi, wśród których najważniejsze to zwiększenie niezawodności systemu obliczeniowego oraz zwiększenie szybkości obliczeń drogą rozwiązań logicznych (rys. 9.3). Często bowiem osiągnięty poziom rozwoju technologicznego nie pozwala osiągnąć



Rys. 9.3. Rozwój szybkości komputerów jedno- i wieloprocesorowych (osiągnięcia szczytowe)

wymaganej szybkości w pojedynczym komputerze. Ponadto w przyszłości zostanie zahamowany dotychczasowy „oszałamiający" wzrost szybkości działania podstawowych elementów elektronicznych. Zatem o wzroście szybkości systemów komputerowych decydować głównie będzie ich organizacja logiczna, a nie szybkość propagacji elementów elektronicznych.

Dane literaturowe podają, że dotychczas czas propagacji układów technicznych zmniejszał się około 10 razy na dziesięciolecie, zaś czas wykonania jednej średniej instrukcji zmniejszał się około 10^3 razy na dziesięciolecie dzięki kolejnym strukturalnym udoskonaleniom oraz opracowaniu nowych algorytmów działań maszyn cyfrowych. Na przykład maszyna MARK I skonstruowana w 1944 roku (w technice przekaźnikowej) potrzebowała 333 milisekund na wykonanie jednego dodawania, zaś maszyna UNIVAC I (1951) zbudowana w technice lampowej potrzebowała tylko 283 mikrosekund na dodawanie. Z kolei maszyna CDC 6600 (1964) skonstruowana w technice tranzytowej wykonuje jedno dodawanie w czasie mniejszym od 300 nanosekund. Nasuwa się zatem pytanie: kiedy zbuduje się komputer wykonujący dodawanie dwóch liczb w ciągu 300 pikosekund i kiedy — w ciągu 0,3 pikosekundy (jeśli to w ogóle jest możliwe). Trzeba odpowiedzieć, że osiągnięcie czasu dodawania 300 pikosekund będzie sprawą bardzo trudną, gdyż zbliżamy się praktycznie do „granic" możliwości elektronicznych technik. Ograniczenia te wynikają z ograniczonej szybkości przesyłania sygnałów elektrycznych w przewodach, bowiem odcinek przewodu elektrycznego, przez który przebiegnie sygnał elektryczny w ciągu jednej nanosekundy wynosi zaledwie 12 cali, czyli około 30 cm. Zatem w ciągu 300 pikosekund przebiegnie mniej niż 10 cm, zaś w ciągu 0,3 pikosekundy mniej niż 0,1 mm drogi! Z drugiej strony zbudowanie dziś komputera, w którym łączna długość przewodów, przez które przebiegają dodawane informacje — będzie mniejsza od 10 cm, jest jeszcze wielkim problemem technicznym. Oczywiście dodawane informacje nie tylko mają pokonać ten 10-centymetrowy odcinek drogi, lecz także przejść przez dziesiątki układów elektronicznych, które to układy powodują dodatkowe opóźnienia. Na przykład w 1975 r. czas propagacji najszybciej działających laboratoryjnych elementów wynosił około 0,4 nanosekundy, natomiast elementów stosowanych w konstrukcjach szybkich komputerów — około 1÷2 nanosekund. Prognozowany rozwój najprostszych elementów elektronicznych (tzw. bramek) przedstawiony jest w tabelach 9.1 i 9.2. Prognozę szybkości działania komputerów jednoprocesorowych (uniprocesorów) i komputerów macierzowych (w których występuje zespół N procesorów) przedstawiono w tabelach 9.3 i 9.4. Parametry podane w tabelach dotyczą oczywiście szczytowych osią-

Tabela 9.1

Charakterystyka bipolarnych elementów logicznych

Rok	Opóźnienie bramki, ns	Moc, mW	Gęstość bramki, cm³	Koszt bramki, $
1975	0,4	55	55	0,30
1980	0,2	35	90	0,10
1985	0,1	25	125	0,05
1990	0,06	20	155	0,03

Tabela 9.2

Charakterystyka elementów MOS, CMOS, SOS
(prostsza struktura, mniejsze pole, mniejsza moc; tańsze, lecz wolniejsze)

Rok	Opóźnienie bramki, ns	Moc, mW	Gęstość bramki, cm³	Koszt bramki, $
1975	1,0	10	1100	0,02
1980	0,5	5	2800	0,005
1985	0,25	2	6000	0,003
1990	0,15	0,8	17500	0,0015

Tabela 9.3

Szybkość uniprocesorów
(jeden strumień instrukcji, jeden strumień danych, specjalne JS do jednoczesnej pracy CJP i operacji we-wy; przykłady: CDC 7600, IBM 360 i 370, ODRA 1300)

Rok	Czas dodawania, ns	Czas mnożenia, ns	Czas dostępu do PAO, ns	Szybkość linii procesora	Szybkość procesora* z linią przetwarzania, MIPS
1975	7	95	20	23	120
1980	3,5	32	10	66	330
1985	1,2	16	5,6	140	715
1990	0,7	6	3	145	740

* Np. CDC STAR-100 i IBM 360/195

Tabela 9.4

Szybkość procesorów macierzowych
(jeden strumień instrukcji i wiele strumieni danych, tzn. N jednakowych procesorów, np. ILLIAC-IV)

Rok	Czas dodawania, ns	Czas mnożenia, ns	Czas dostępu, ns	Szybkość w MIPS	
				$N=1$	$N=64$
1975	7,0	95	20	14	880
1980	3,5	50	10,0	26	1700
1985	1,8	25	5,6	50	3200
1990	0,8	12	3,0	98	6250

gnięć, natomiast seryjnie produkowane maszyny (klasy dużych EMC) mają parametry przeciętnie 10 razy gorsze (np. parametr szybkości podany w tabeli w MIPS, tj. w milionach instrukcji na sekundę).

Innym powodem pojawienia się systemów wieloprocesorowych (lub wielomaszynowych) jest wymagana wysoka niezawodność pracy systemu. Wymaganie to dla wielu użytkowników może być ważniejsze od uzyskania dużej szybkości przetwarzania. Sposoby uzyskania dużej niezawodności mogą być różne. Często stosuje się jedną z poniższych metod:

— systemy rekonfiguralne (które często są typowymi systemami wieloprocesorowymi),
— systemy dupleksowe (tj. systemy dublujące obliczenia).

Metoda pierwsza polega na zastosowaniu kilku jednakowych zespołów funkcjonalnych (procesorów), z których każdy wykonuje na ogół różne zadania (programy). Zatem łączna moc systemu zależy od ilości przyjętych zespołów funkcjonalnych. W takich systemach niezbędne są rozbudowane środki kontroli poprawności pracy poszczególnych zespołów, chodzi bowiem o natychmiastowe wykrycie uszkodzonego zespołu i wyłączenie go z systemu (wówczas jego programy będą realizowane przez pozostałe zespoły). Następuje zatem tzw. rekonfiguracja zespołów systemu.

W metodzie drugiej, tj. w systemie dupleksowym, stosuje się dwie lub więcej (przeważnie trzy) kompletne maszyny cyfrowe z których każda wykonuje to samo zadanie (problem) i na tych samych danych. W określonych momentach czasowych (albo stale) sprawdza się zgodność wyników obliczeń, a każda niezgodność wyników jest sygnałem błędnej pracy jednej z maszyn. Sygnał ten powoduje uruchomienie określonej procedury (technicznej lub programowej) mającej na celu wskazanie uszkodzonej maszyny i tym samym wyeliminowanie jej z systemu.

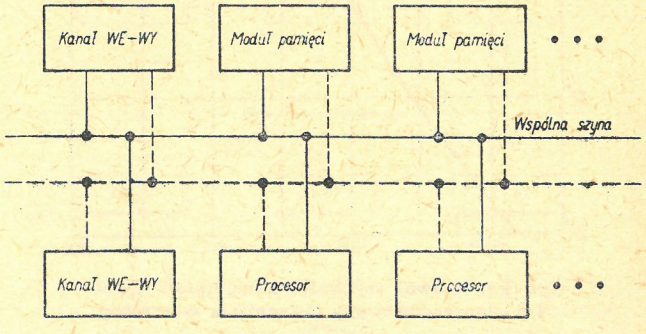
Metoda pierwsza jest bardziej ekonomiczna, gdyż pozwala na pełne wykorzystanie całego systemu i wyłączenie określonego zespołu z systemu tylko wówczas, gdy zostanie on uszkodzony.

9.2. Przykłady struktur wieloprocesorowych

Poniżej podano kilka uogólnionych struktur systemów wieloprocesorowych, które znalazły praktyczne zastosowania. W strukturach tych pokazany zostanie sposób łączenia poszczególnych zespołów funkcjonalnych występujących w systemach wieloprocesorowych.

9.2.1. Systemy wieloprocesorowe z szyną dzieloną

Zespoły funkcjonalne tych systemów (rys. 9.4) komunikują się ze sobą za pośrednictwem wspólnej (lub dzielonej) szyny. Oznacza to, że wszystkie systemy są podłączone do jednej szyny w sposób ujednolicony (np. za pomocą 40 przewodów). Zespół, który chce przekazać (lub odebrać) informacje do innego zespołu, zgłasza w ustalony sposób swoje żądanie do szyny, zaś sterowanie pracą szyny organizuje (według określonego priorytetu) przesłanie tej informacji. Szyna pracuje w ten sposób, że dzieli swój czas na małe odcinki (np. po 500 nanosekund), w których obsługuje odrębne zgłoszenia. Zatem w jednym odcinku czasowym obsługiwane jest tylko jedno zgłoszenie, pozostałe zaś czekają.



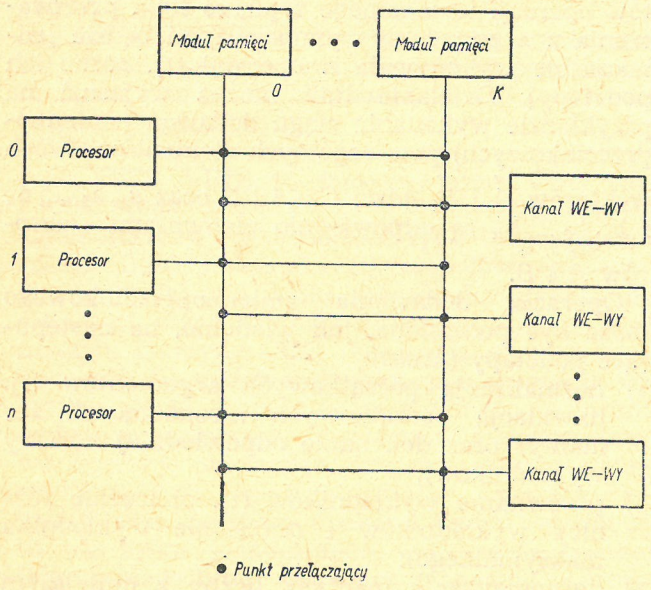
Rys. 9.4. Struktura systemu wieloprocesorowego oparta o szynę dzieloną

Powyższa struktura jest bardzo prosta i tania w realizacji, lecz nie pozwala na jednoczesne przesyłanie danych pomiędzy różnymi zespołami systemu (np. procesor I — pamięć II, procesor II — pamięć I). Szybkość działania takiego systemu można zwiększyć przez zastosowanie drugiej dzielonej szyny oznaczonej na rys. 9.4 linią przerywaną, lecz w tym przypadku każdy z zespołów musi mieć dwa punkty dostępu (lub dwa porty).

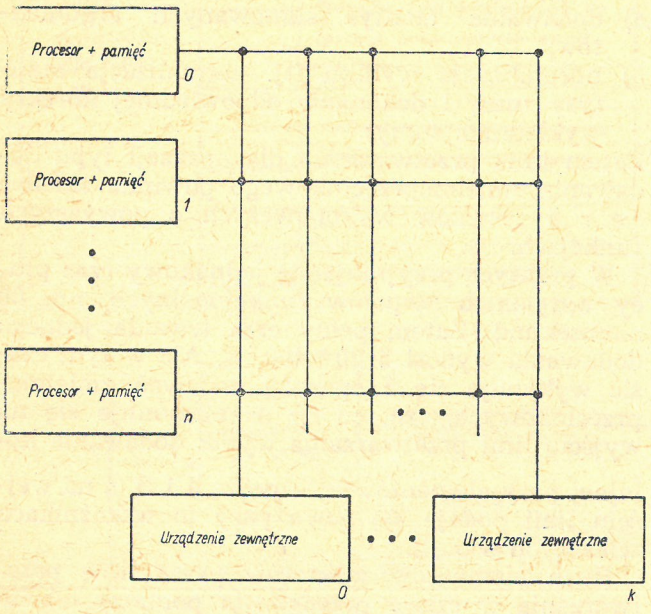
Struktury z szyną dzieloną występują na ogół w małych (np. minikomputerowych) lub w średnich systemach wieloprocesorowych, gdzie koszt systemu jest rzeczą ważniejszą od szybkości działania.

9.2.2. Systemy wieloprocesorowe z przełącznicą krzyżową

Organizacją, w której każdy z zespołów systemu ma tylko jedno miejsce dostępu (jeden port), jest organizacja oparta o pewną macierz punktów przełączających (rys. 9.5). W tym przypadku każdy zespół (moduł) pamięci może być przełączony do dowolnego procesora lub do dowolnego zespołu urządzeń wejścia—wyjścia. W tej strukturze każde dwa wolne zespoły mogą współpracować ze sobą i to równocześnie z innymi przesyłaniami.



Rys. 9.5. System wieloprocesorowy oparty o przełącznicę krzyżową

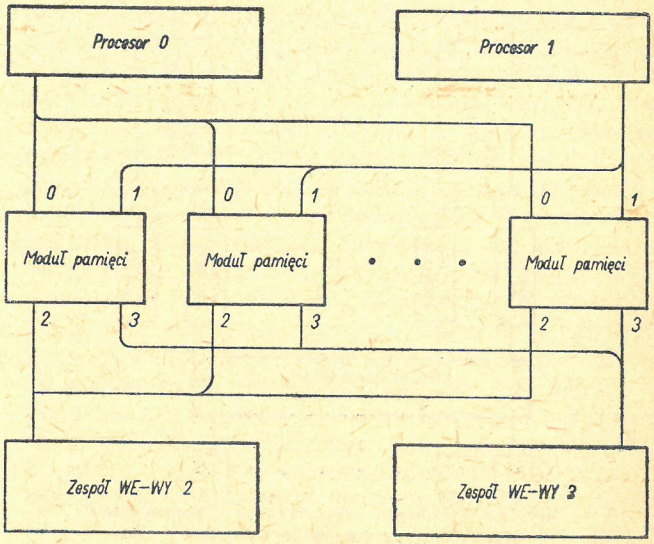


Rys. 9.6. System wielomaszynowy oparty o przełącznicę krzyżową

Jak widać z rys. 9.5 struktura krzyżowej przełącznicy jest modularna i może być dowolnie rozszerzona, gdy rośnie ilość zespołów systemu. Struktura krzyżowa może również wystąpić w systemach wielomaszynowych (rys. 9.6), gdzie elementem jest kompletna jednostka centralna (tj. procesor wraz z własną pamięcią operacyjną i kanałami).

9.2.3. Systemy z wieloszynową i wieloportową strukturą

Jest to struktura systemu, w którym moduły pamięci mają więcej punktów dostępu niż jeden (na rys. 9.7 podano przykład z czterema portami). Do różnych portów doprowadza się połączenia (szynę) z różnych procesorów lub z różnych zespołów wejścia—wyjścia, dlatego zbędna jest tu osobna

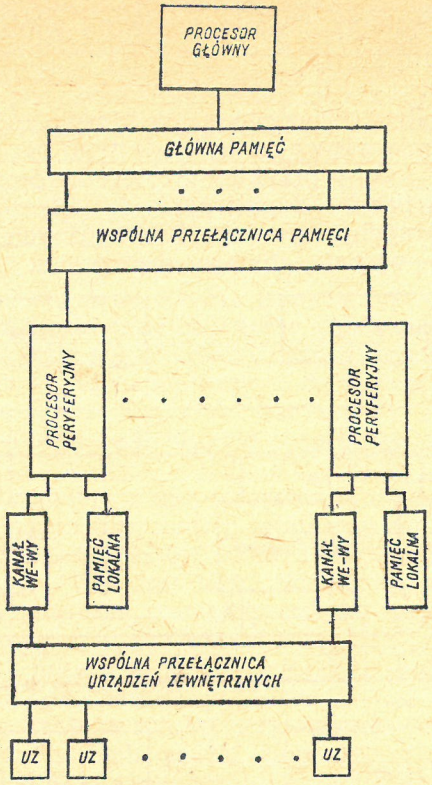


Rys. 9.7. System wieloprocesorowy o strukturze wieloszynowej i wieloportowej (przykład z dwoma procesorami i dwoma zespołami urządzeń zewnętrznych)

przełącznica pamięci i tym samym nie występują zewnętrzne konflikty dostępu do pamięci. Oczywiście różne procesory i różne zespoły urządzeń zewnętrznych mogą jednocześnie zgłaszać odwoływania do tego samego modułu pamięci; wówczas powstają tzw. lokalne konflikty, którymi zajmuje się lokalne sterowanie danego modułu pamięci. Im większa ilość niezależnych modułów pamięci występuje w systemie, tym mniejsze jest prawdopodobieństwo powstania konfliktu, ale z kolei im większa ilość modułów pamięci, tym większa ogólna liczba portów dostępu (powodująca wzrost kosztu modułu). Warto również zauważyć, że dla ustalonej konstrukcji modułu pamięci ilość procesorów (lub zespołów wejścia—wyjścia) jest ograniczona i nie może wzrastać w sposób modułowy.

9.2.4. Asymetryczne systemy wieloprocesorowe

Asymetryczne (lub niejednorodne) systemy wieloprocesorowe składają się z niejednakowych procesorów. W takich systemach występuje przeważnie jeden bardzo szybki i uniwersalny procesor (tzw. procesor główny), przeznaczony głównie do realizacji głównego procesu przetwarzania — oraz kilka (lub kilkanaście) niedużych procesorów specjalistycznych, przeznaczonych do wykonywania procedur pomocniczych (np. obsługa systemu wprowadzania i wyprowadzania informacji). Procesory specjalistyczne mają przeważnie własną małą pamięć operacyjną i mogą wykonywać niezależne programy. Ogólnym sterowaniem pracą procesorów specjalistycznych zajmuje się wspólny system operacyjny, mieszczący się w pamięci procesora głównego. Zatem wspólny system opera-

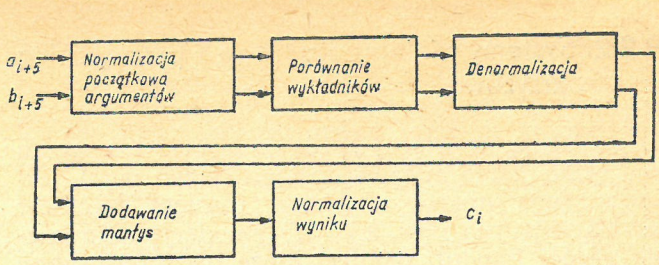


Rys. 9.8. Struktura asymetryczna systemu wieloprocesorowego (przykład CDC 6600 i 6700)

cyjny rozdziela pracę pomiędzy poszczególnymi procesorami na poziomie zadań. Najbardziej typowymi i rozpowszechnionymi systemami asymetrycznymi są systemy CDC 6600 i 7600 (rys. 9.8), w których procesorami specjalistycznymi (zw. peryferyjnymi) są małe maszyny typu CDC 160A (przeważnie do 10 procesorów). Procesory peryferyjne mają 12-bitowe mechanizmy działań oraz prostą (lecz uniwersalną) listę rozkazów. Mają również małą własną pamięć o pojemności 4096 słów 12-bitowych. Procesory peryferyjne (PP) zajmują się głównie wykonywaniem funkcji wejścia—wyjścia (w tym funkcji realizowanych przez tradycyjne jednostki sterujące urządzeniami zewnętrznymi). Ponadto jeden z procesorów PP zajmuje się wykonywaniem funkcji pulpitu sterowania systemem, zaś inny zorganizowaniem pracy całego systemu (w tym i podziałem zadań pomiędzy procesorami). Przyjęte rozwiązania współpracy międzyprocesorowej zakładają, że współpraca ta nie odbywa się za pośrednictwem przerwań międzyprocesorowych, lecz za pośrednictwem „skrzynki pytań i odpowiedzi" zorganizowanej we wspólnej pamięci.

9.2.5. Systemy z linią przetwarzania

Dotychczas była mowa o jednoprocesorowych maszynach, w których pojedynczy rozkaz steruje pojedynczym strumieniem danych. Do tej klasy maszyn należą również maszyny z tzw. linią przetwarzania (*Pipe Line*), w których występuje wysoki stopień jednoczesności wykonania operacji. Maszyny z linią przetwarzania mają odrębne zespoły wykonawcze do realizacji poszczególnych (logicznych) podetapów procesu obliczeń złożonych operacji. Zespoły te tworzą jakby „taśmę", na wejściu której realizuje się pierwszy podetap operacji, zaś na końcu ostatni podetap danej operacji. Przy tym, jeśli na wejściu linii przetwarzania dostarcza się, np. co określony przedział czasu (krok), nowy zestaw argumentów operacji tego samego typu, to linia przetwarzania z n zespołami wykonawczymi będzie jednocześnie wykonywała n operacji (tj. różne ich podetapy). Działanie linii można wyjaśnić na przykładzie wykonania ciągu dodawań (zmiennoprzecinkowych) poszczególnych elementów dwóch wektorów $\vec{A}$ i $\vec{B}$, gdzie $\vec{C}=\vec{A}+\vec{B}$ oraz $a_1, a_2, ..., a_n$ i $b_1, b_2 ..., b_n$ są odpowiednio elementami wektorów $\vec{A}$ i $\vec{B}$.



Rys. 9.9. Przykład linii przetwarzania do dodawania zmiennoprzecinkowego

Operacja dodawania zmiennoprzecinkowego może być podzielona, jak wiadomo, na następujące podetapy (kroki):

1) normalizacja początkowych argumentów (tj. likwidacja początkowych nieznaczących zer mantys oraz dokonanie odpowiedniej korekty wykładników);
2) porównanie wykładników (tj. obliczenie różnicy wykładników i określenie wykładnika maksymalnego);
3) dernomalizacja mantysy liczby z mniejszym wykładnikiem (w celu wyrównania przecinka dzielącego część całkowitą od części ułamkowej);
4) dodawanie mantys sumowanych argumentów;
5) normalizacja wyniku (tj. normalizacja mantysy sumy i dokonanie odpowiedniej korekty wykładnika sumy).

Zatem linia przetwarzania dla operacji typu dodawania zmiennoprzecinkowego powinna składać się z 5 zespołów wykonawczych o powyższych funkcjach.

W praktyce przyjmuje się jednakowy czas pracy wszystkich zespołów funkcjonalnych, np. 80 nanosekund. Zatem pełny czas trwania jednego dodawania wynosi $5 \cdot 80 = 400$ ns. Ale w tym czasie wykonuje się 5 operacji dodawań zmiennoprzecinkowych, tzn. co 80 ns otrzymuje się na wyjściu linii przetwarzania wynik dodawania kolejnej pary elementów wektorów $\vec{A}$ i $\vec{B}$, a na wejściu linii podaje się nowe pary (o wskaźnikach większych o pięć).

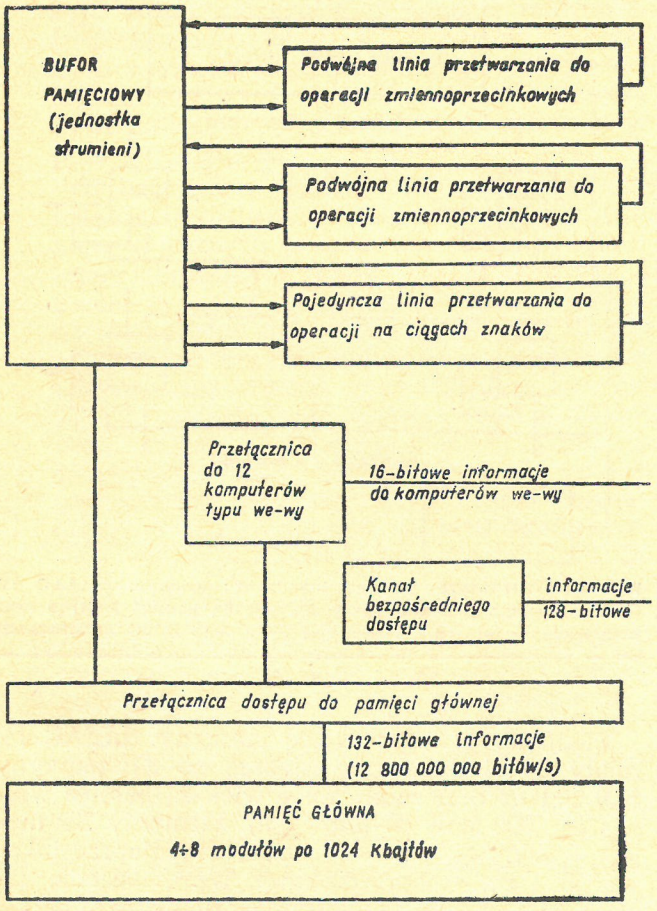
Ruch elementów wektorów przez linię przetwarzania w czasie przedstawia poniższa tabela:

Czas	t_1	t_2	t_3	t_4	t_5	...	t_{i+4}
Na wejściu zespołu 1	a_1, b_1	a_2, b_2	a_3, b_3	a_4, b_4	a_5, b_5	...	a_{i+4}, b_{i+4}
2		a_1, b_1	a_2, b_2	a_3, b_3	a_4, b_4	...	a_{i+3}, b_{i+3}
3			a_1, b_1	a_2, b_2	a_3, b_3	...	a_{i+2}, b_{i+2}
4				a_1, b_1	a_2, b_2	...	a_{i+1}, b_{i+1}
5					a_1, b_1	...	a_i, b_i

Pełne wykorzystanie maszyny z linią przetwarzania zależy zarówno od typu zadań (np. algebraicznego) do rozwiązania, jak i od struktury samego programu, który będzie się starał „nieprzerwanie" dostarczać na wejściu linii przetwarzania kolejne argumenty operacji tego samego typu. Na przykład pętla dodawania elementów dwóch wektorów nie powinna posiadać tylko jednej instrukcji maszynowej dodawania, lecz więcej (w naszym przypadku pięć):

DODAJ (a_i, b_i), DODAJ (a_{i+1}, b_{i+1}), DODAJ (a_{i+2}, b_{i+2}), DODAJ (a_{i+3}, b_{i+3}), DODAJ (a_{i+4}, b_{i+4}), POWIĘKSZ WSKAŹNIKI (I) O PIĘĆ, SKOCZ DO POCZĄTKU PĘTLI DODAWANIA JEŚLI NIE PODANO JESZCZE WSZYSTKICH ELEMENTÓW WEKTORÓW (inaczej wyjdź z pętli). W celu uproszczenia założono w przykładzie, że liczba elementów wektorów A i B jest wielokrotnością piątki.

Najbardziej znanymi maszynami o organizacji typu PIPE LINE są maszyny typu CDC 6600 i CDC 7600, gdzie każdy z dziewięciu różnych arytmometrów ma strukturę typu linia przetwarzania. Ilość podetapów (tj. zespołów wykonawczych) wykonania danej operacji lub grupy operacji jest różna w różnych arytmometrach tych maszyn. Maszyny CDC 7600 mają odrębne arytmometry dla takich instrukcji, jak: mnożenia stałoprzecinkowe i zmiennoprzecinkowe, dodawania zmiennoprzecinkowe, dodawania stałoprzecinkowe, operacje przesunięć, dzielenia stało- i zmiennoprzecinkowe, operacje logiczne, operacje konwersji oraz operacje skoków. Zatem występuje tu nie tylko jednoczesność wykonania operacji (tego samego typu) wewnątrz danej linii przetwarzania, lecz także jednoczesność pracy poszczególnych arytmometrów.



Rys. 9.9a. Ogólna struktura systemu CDC STAR-100

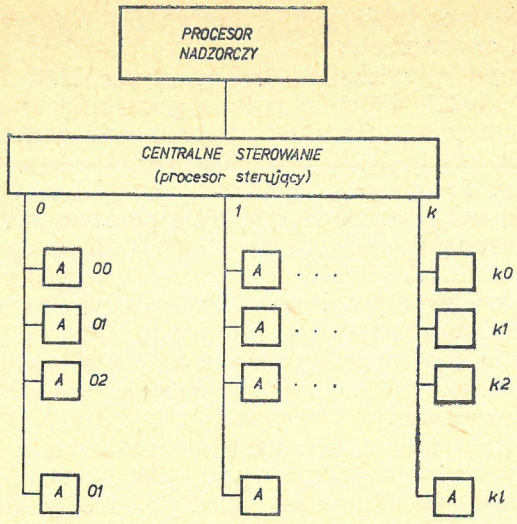
Innym przykładem jest maszyna CDC STAR-100 (1971), która ma dwie podwójne zmiennoprzecinkowe linie przetwarzania z 30 podetapami każda, czyli każda linia może jednocześnie wykonywać 30 operacji. System CDC STAR-100 (rys. 9.9. a) ma również trzecią linię przetwarzania dostosowaną do działań na ciągach znakowych.

System STAR-100 jest w rzeczywistości układem wielokomputerowym, składającym się z dziewięciu komputerów, w którym to układzie jednocześnie wykonuje się różne zadania systemu operacyjnego, przetwarzania zbiorów danych oraz wprowadzanie i wyprowadzanie danych. Pamięć główna systemu, która może składać się z 8 modułów jednomegabajtowych, jest bardzo szybka i przekazuje dane z szybkością 12,8 Gigabitów na sekundę. Ciągi danych napływają do linii przetwarzania z szybkością 100 milionów 32-bitowych operandów na sekundę. Dla zapewnienia takiej szybkości napływu danych należałoby przyjąć cykl wewnętrzny linii przetwarzania równy 10 nanosekund (lub częstotliwości 100 MHz). W związku z trudnościami technicznymi w realizacji tak wysokiej częstotliwości, przyjęto częstotliwość 25 MHz (czyli 40-nanosekundowy cykl) dla jednej linii i wbudowano dwie podwójne linie przetwarzania danych zmiennoprzecinkowych. Zatem jednocześnie przetwarza się $4 \times n$ operacji, w czterech n-etapowych liniach przetwarzania (liczba etapów lub podetapów wykonania operacji zależy od rodzaju danej operacji i wynosi średnio 30).

9.2.6. Systemy o organizacji macierzowej

Dotychczas omówione klasyczne systemy wieloprocesorowe składały się z odrębnych procesorów, z których każdy wykonywał niezależny ciąg instrukcji (tzn. miał własny licznik instrukcji) i na niezależnym strumieniu danych. Zatem maszyny wieloprocesorowe osiągają wysokie szybkości dzięki powielaniu kompletnych procesorów (tj. łącznie z własnym sterowaniem pobrania i wykonania ciągu instrukcji). Natomiast maszyny o organizacji typu PIPE LINE osiągają dużą szybkość wykonania operacji dzięki wysokiemu zrównoleglaniu wykonania poszczególnych instrukcji tego samego ciągu instrukcji.

Innym typem maszyn osiągających wysokie wydajności przetwarzania są tzw. maszyny o organizacji m a c i e r z o w e j (lub tablicowej). Istotną cechą tych maszyn jest to, że wykonują ten sam ciąg operacji na różnych ciągach (strumieniach) danych. Strumienie danych są oczywiście tego samego typu, lecz ich konkretne wartości opisują różne jednocześnie badane (lub przetwarzane) obiekty. Zatem tę samą operację wykonuje się jednocześnie na n różnych parach (lub zestawach) danych, przy czym liczba n może np. przyjmować wartości: 64, 256, 1024. Oznacza to wbudowanie w jednej maszynie 1024 procesorów (lub raczej arytmometrów), które sterowane są jednocześnie z tego samego centralnego sterowania (czyli według jednego licznika instrukcji). Arytmometry (A) — maszyny o takiej organizacji, zwane też elementami przetwarzającymi, ustawiane są w formie macierzy (np. o $k+1$ kolumn i $l+1$ wier-



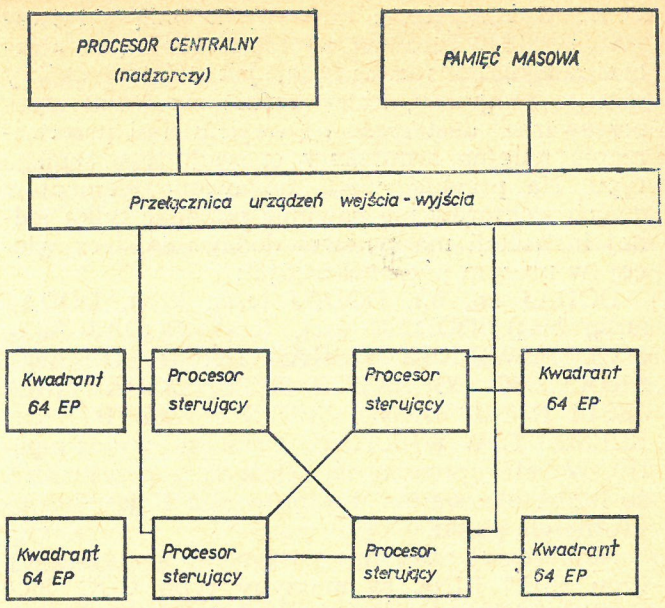
Rys. 9.10. Ogólna struktura systemu o organizacji macierzowej
(k+1 — kolumn i l+1 — wierszy)

szy). Przykładową strukturę maszyny o organizacji macierzowej przedstawia rys. 9.10.

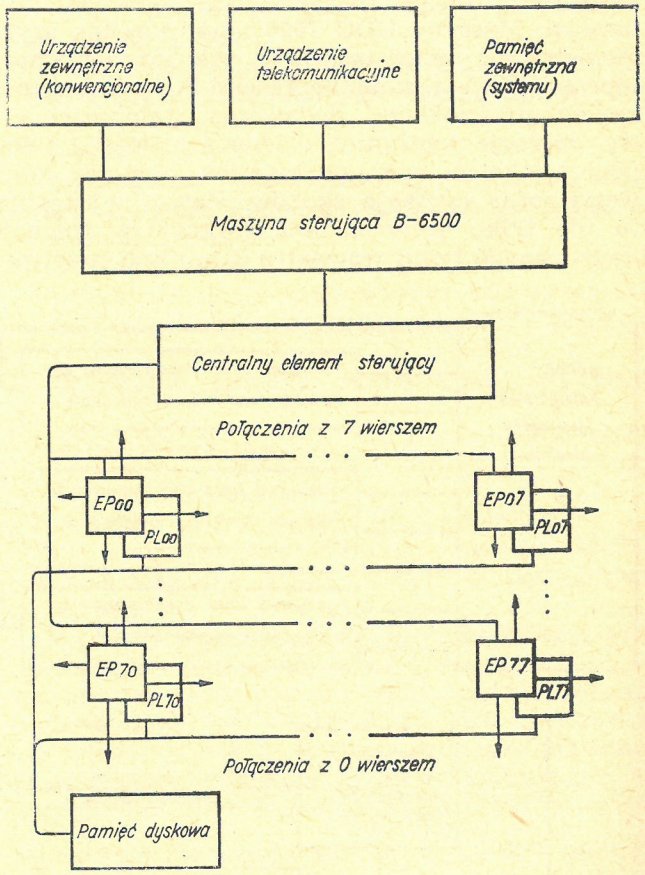
Centralne sterowanie (lub procesor sterujący) systemu o macierzowej organizacji przekazuje do poszczególnych arytmometrów sygnały sterujące, które określają typ operacji do wykonania w danej chwili. Poszczególne arytmometry można również włączyć lub wyłączyć z procesu wykonania operacji.

Pierwszym projektem maszyny o macierzowej organizacji był SOLOMON (1965), który zakładał istnienie 1024 (=32×32) elementów przetwarzających (EP). Każdy EP miał mieć własną pamięć operacyjną o pojemności 4096 bitów. Procesor sterujący miał własną pamięć, w której przechowywany był program przetwarzający. Procesor sterujący pobierał i interpretował kolejne instrukcje programu, po czym wysyłał odpowiednie sygnały do poszczególnych EP (w celu wykonania tej samej operacji przez poszczególne EP, lecz na różnych danych przechowywanych w komórkach — pamięci „prywatnych" zespołów EP — o tych samych adresach). Procesor centralny mógł również przesyłać tę samą liczbę (np. wybrane stałe wielkości) jednocześnie do wszystkich elementów EP. Pomiędzy sąsiednimi EP istniała komunikacja, tzn. możliwość równoległego przekazywania danych. Ta ostatnia możliwość pozwala wprowadzać dane z urządzeń zewnętrznych w sposób równoległy (np. wierszami), gdzie tylko „brzegowe" EP miały połączenia z urządzeniami zewnętrznymi.

Kolejnym projektem maszyny o organizacji macierzowej, który częściowo doczekał się realizacji technicznej, był projekt maszyny ILLIAC IV (1970). Pełny projekt maszyny ILLIAC IV (rys. 9.11), którego głównym twórcą był prof. D. L. Slotnick, przewidywał cztery zespoły elementów przetwarzających (EP), zwane kwadrantami, każdy po 64 elementów EP występujących w formie matrycy 8×8 (rys. 9.12). Każdy kwadrant miał mieć własne i niezależne sterowanie (tj. własny procesor sterujący). Zatem poszczególne kwadranty mogły rozwiązywać cztery niezależne główne zadania lub mogły być skupione do rozwiązania jednego zadania z 256 niezależnymi strumieniami danych.



Rys. 9.11. Ogólna struktura systemu ILLIAC IV (zawierająca cztery kwadranty po 64 elementy przetwarzające z własną pamięcią po 2000 słów 64-bitowych)



Rys. 9.12. Struktura jednego kwadrantu systemu ILLIAC IV (każdy element przetwarzający ma własną pamięć lokalną oraz połączenia — poziome i pionowe — z sąsiednimi elementami)
EP — częstotliwość 4 Mh, argumenty 32-, lub 64-bitowe, PL — cykl 240 ns, dostęp 120 ns/64 bity)

Elementy przetwarzające maszyny ILLIAC IV miały być kilkakrotnie szybsze od EP maszyny SOLOMON oraz miały mieć większą pamięć własną (=2000 słów 64-bitowych). Elementy EP mogły wykonywać zarówno działania stałoprzecinkowe jak i zmiennoprzecinkowe. Wprowadzenie danych od urządzeń zewnętrznych do elementów EP zostało również przyśpieszone przez bezpośrednie

połączenia elementów EP z urządzeniami wejścia-wyjścia (a także przez szybszą komunikację międzyarytmometrową). W roli procesora sterującego przewidziano bardzo szybką maszynę typu B-6500 firmy Burroughs (pierwsza instalacja — 1970 r.). Łączna wydajność systemu ILLIAC IV miała sięgać do 200 milionów operacji na sekundę.

Jeden kwadrant maszyny ILLIAC IV, to jest jedna maszyna B-6500 oraz macierz z 64 EP, został zmontowany i uruchomiony na Uniwersytecie w Illinois (przy pomocy firmy Burroughs) w roku 1973.

Maszyna ILLIAC IV, jako maszyna specjalna, jest przeznaczona do rozwiązywania zadań, których działania mogą być wykonywane równolegle. Do takich można zaliczyć różne zadania z dziedziny programowania liniowego, meteorologii i inne. Autorzy projektu tej maszyny twierdzą, że zadania z równoległymi strumieniami danych można rozwiązywać 200 razy szybciej w maszynie macierzowej niż w maszynie o konwencjonalnej organizacji (jeden strumień instrukcji — jeden strumień danych).

Aczkolwiek elementy przetwarzające EP są sterowane z jednego centralnego sterowania, tzn. ustawiane są przez to sterowanie do wykonania tej samej sekwencji instrukcji, jednak istnieje również możliwość uzależnienia sposobu pracy poszczególnych EP od ich stanu lokalnego oraz od typu ich danych lokalnych. Istnieje w szczególności w każdym EP specjalny rejestr modalny, w którym ustawia się tryb przetwarzania danych w danym elemencie EP. Stany rejestrów modalnych mogą być różne w różnych elementach przetwarzających. Stan trybu przetwarzania może wpływać na sposób wykonania sekwencji instrukcji; można zatrzymać wykonanie sekwencji instrukcji w określonych EP stosownie do napotkanych wielkości argumentów (np. przy zero, przy nadmiarze itp.). Powyższa możliwość oznacza indywidualizację (w ograniczonym zakresie) pracy poszczególnych EP.

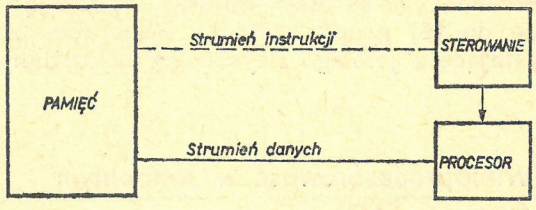
W systemie ILLIAC IV założono, ogólnie mówiąc, możliwość wykonania różnych programów w różnych procesorach EP (oczywiście kosztem obniżenia szybkości całego systemu). Możliwość ta została osiągnięta dzięki wprowadzeniu do centralnego sterowania specjalnego 64-bitowego rejestru (zwanego rejestrem maski), którego pozycje odpowiadają poszczególnym procesorom EP. Za pomocą tego rejestru można maskować, czyli blokować wykonanie kolejnej operacji przez te procesory EP, którym odpowiada pozycja ze stanem 0 w rejestrze maski. Zatem można wyłączyć z pracy dowolną kombinację procesorów EP, wstawiając zero w odpowiadających im pozycjach rejestru maski. Powyższa możliwość oznacza, że można określoną grupę procesorów EP włączyć do wykonania jednej sekwencji instrukcji, zaś inną grupę procesorów do wykonania drugiej sekwencji instrukcji, aczkolwiek wykonanie tych dwóch różnych sekwencji nie odbywa się już równolegle, lecz szeregowo. Zatem kolejno wstawiając jedynkę w kolejnych pozycjach rejestru maski otrzyma się wykonanie różnych programów przez różne procesory EP. Powyższa możliwość jest wykorzystywana w takich przypadkach, kiedy zostaje naruszona regularność obliczeń w określonym przedziale danych (wówczas może zachodzić potrzeba zmiany algorytmu liczenia). Ponadto każdy procesor EP może generować do 8 różnych cech opisujących wyniki działań w danym EP. Cechy te przechowywane są w rejestrze modalnym danego EP i charakteryzują proces obliczeń w każdej chwili. Stany rejestrów modalnych mogą być przeczytane i testowane przez centralne sterowanie. Zatem centralne sterowanie może testując cechy z poszczególnych EP, podejmować różne decyzje dotyczące dalszego biegu obliczeń.

9.2.7. Ciągi instrukcji i ciągi danych

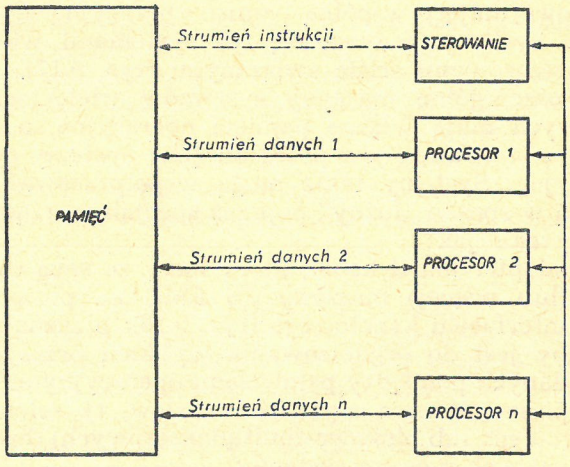
W dotychczasowych opisach różnych systemów wieloprocesorowych rozpatrywano je głównie pod względem struktury. Często w literaturze podaje się również inny podział systemów komputerowych — zależnie od liczby niezależnych strumieni (ciągów) instrukcji i niezależnych strumieni danych występujących w danym systemie komputerowym. W tej klasyfikacji wyróżnia się systemy mające:

— pojedynczy strumień instrukcji i pojedynczy strumień danych (PSI—PSD); są to komputery jednoprocesorowe o konwencjonalnej strukturze, w których występuje jeden ciąg instrukcji (jeden licznik instrukcji) i jeden ciąg danych (argumentów) związanych z tym ciągiem instrukcji (rys. 9.13);

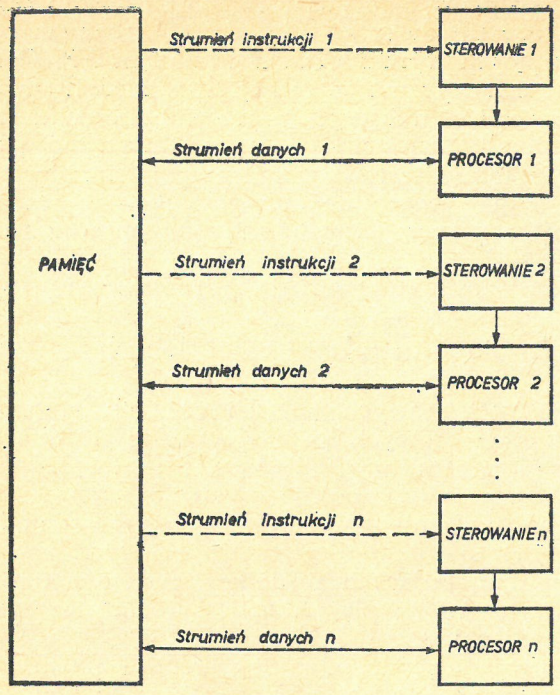
— pojedynczy strumień instrukcji i wielokrotne strumienie danych (PSI-WSD); są to np. procesory macierzowe, asocjacyjne i inne niekonwencjonalne rozwiązania (rys. 9.14);



Rys. 9.13. System z jednym strumieniem instrukcji (pobieranych do sterowania) i z jednym strumieniem danych (do lub z właściwego procesora)



Rys. 9.14. System z jednym strumieniem instrukcji i n strumieniami danych



Rys. 9.15. System z n strumieniami instrukcji i z n strumieniami danych

— wielokrotne strumienie instrukcji i wielokrotne strumienie danych (WSI—WSD); jest to normalna sytuacja w typowym systemie wieloprocesorowym, gdzie każdy procesor wykonuje swój niezależny program na danych związanych z własnym programem (rys. 9.15).

System komputerowy, w którym występuje kilka niezależnych strumieni instrukcji i jeden strumień danych nie znajduje na ogół praktycznego zastosowania (niektórzy autorzy zaliczają formalnie do tej grupy systemy wieloprocesorowe korzystające z jednego fizycznego bloku danych).

9.3. Wieloprocesorowość w Jednolitym Systemie EMC

Maszyny cyfrowe Jednolitego Systemu mogą wchodzić w konfiguracje wielomaszynowe (zwane też systemami wieloprocesorowymi luźnie związanymi) lub w typowe konfiguracje wieloprocesorowe mające wspólną pamięć operacyjną. Ostatnie systemy nazywa się też systemami wieloprocesorowymi ściśle związanymi (rys. 9.17).

Poszczególne maszyny systemów wielomaszynowych mają własne pamięci operacyjne, a ich praca sterowana jest przez odrębne systemy operacyjne. Systemy takie mogą współpracować ze sobą w sposób „luźny" poprzez specjalne urządzenia, takie jak:

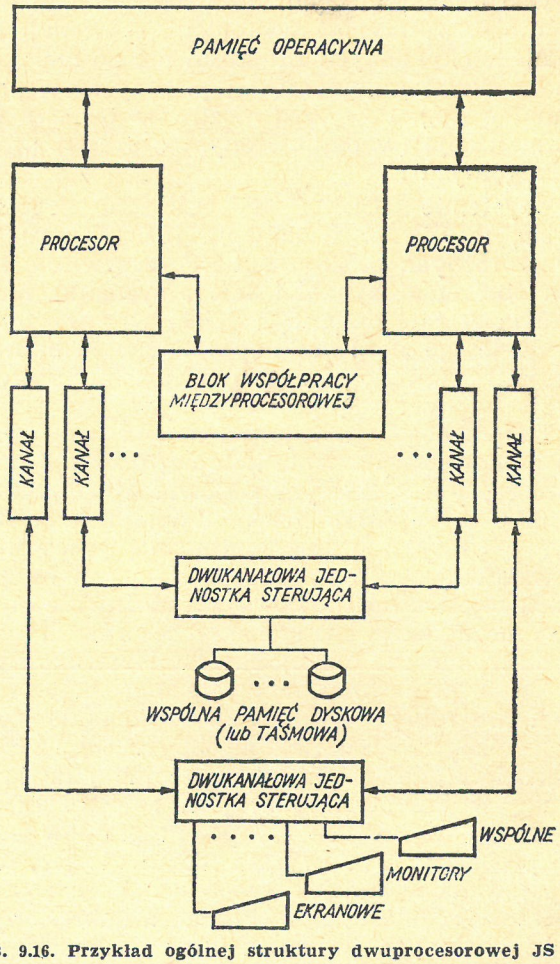
— adapter kanał-kanał, który łączy ze sobą dwie (lub więcej) maszyny JS EMC na poziomie interface'u kanałowego (rys. 9.18); przeznaczony jest do organizowania szybkich przesyłań danych pomiędzy pamięciami operacyjnymi (za pośrednictwem szybkich kanałów, tj. selektorowych lub blokowo-multiplekserowych) dwóch różnych maszyn systemu;

— jednostka bezpośredniego sterowania, która służy do szybkiej wymiany informacji pomiędzy dwiema maszynami; tą drogą przekazuje się krótkie informacje (np. pytania, odpowiedzi, przerwania awaryjne itp.).

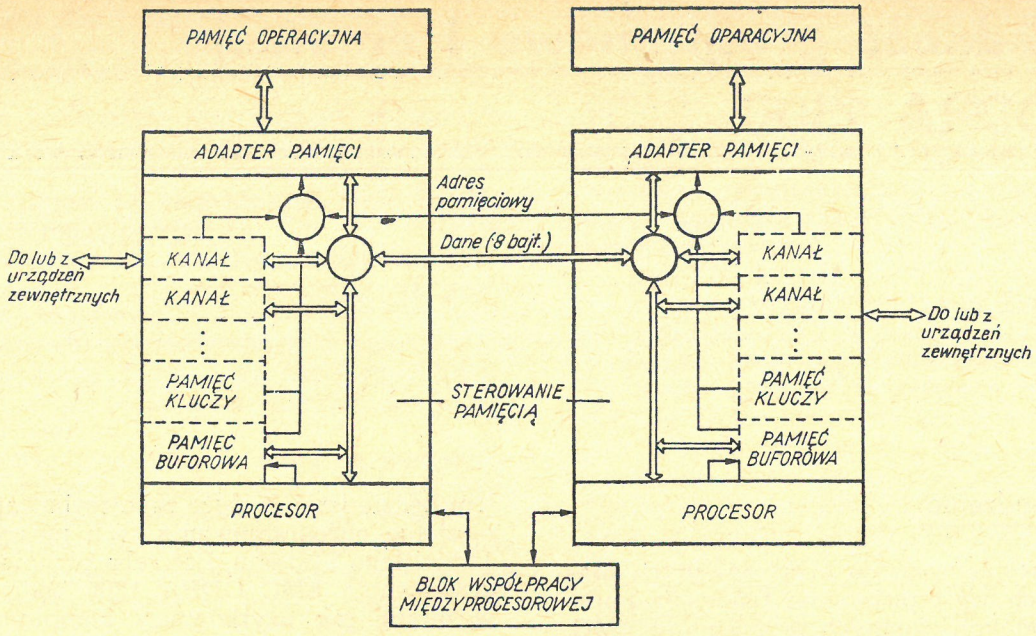
Systemy wieloprocesorowe Jednolitego Systemu mają nie tylko wspólną pamięć operacyjną (rys. 9.16), tj. pamięć jednolicie adresowaną przez wszystkie procesory, lecz także wspólny system operacyjny, który organizuje przebieg prac w poszczególnych procesorach systemu oraz organizuje wszystkie operacje wejścia-wyjścia. Współpraca pomiędzy procesorami odbywa się za pośrednictwem specjalnego bloku, przez który jeden procesor przekazuje do drugiego procesora różne polecenia i sygnały (sygnały o awarii danego procesora) informacyjne. Każdy system wieloprocesorowy wyposażony jest również w specjalny pulpit, na którym zadaje się aktualną konfigurację systemu (np. numeracja i przydział bloków pamięci, przydział urządzeń zewnętrznych itp.).

Możliwość tworzenia konfiguracji wieloprocesorowych została zapewniona dzięki przyjęciu następujących środków:

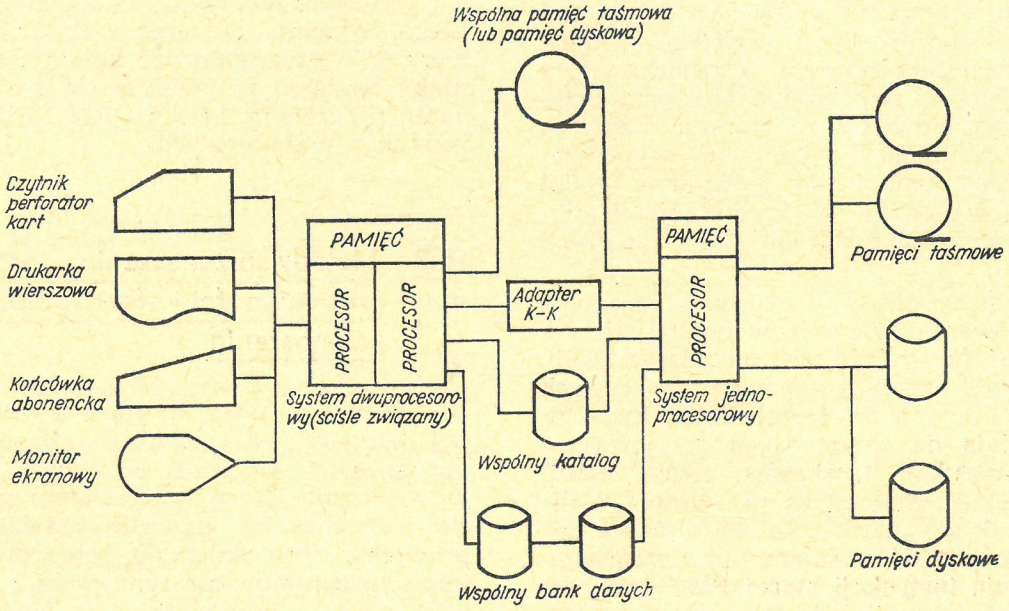
— środki p r e f i k s o w a n i a, które pozwalają przypisać początkowemu obszarowi (o adresach 0÷4095) danego procesora dowolny blok pamięci operacyjnej;
jeśli np. początkowym obszarom dwóch różnych procesorów przyporządkuje się adresy ALFA I BETA (gdzie ALFA ≠ BETA), to odwoływania tych procesorów do początkowego obszaru (o realnych adresach 0÷4095) nie będą odnosiły się do tego samego bloku fizycznego o pojemności 4 Kb, lecz do dwóch różnych bloków fizycznych — zatem każdy z procesorów zachowa swój „prywatny" obszar po-



Rys. 9.16. Przykład ogólnej struktury dwuprocesorowej JS EMC



Rys. 9.17. Przykład systemu dwuprocesorowego JS EMC (ściśle związanego)



Rys. 9.18. Przykład wieloprocesorowej organizacji (do zrealizowania na bazie sprzętu JS EMC)

czątkowy, pomimo że pozostała pamięć operacyjna jest wspólna dla wszystkich procesorów systemu; jest to sprawa istotna, gdyż obszar początkowy pamięci jest ściśle związany z organizacją logiczną EMC Jednolitego Systemu (np. z systemem przerwań, systemem wejścia-wyjścia itp.);

— środki synchronizacji zegarów różnych procesorów systemu wieloprocesorowego; dzięki nim zapewnione jest jednoczesne zapoczątkowanie pracy zegarów (dobowych) oraz kontrola współbieżnego ich biegu; zatem w dowolnej chwili stan zegarów różnych procesorów jest jednakowy i dzięki temu instrukcja czytania stanu zegara może pojawiać się w dowolnym procesorze;

— środki współpracy międzyprocesorowej, które pozwalają wykonać specjalną instrukcję o nazwie PRZEKAŻ SYGNAŁ DO PROCESORA o określonym adresie.

Za pomocą ostatniej instrukcji procesor, w którym ona się pojawiła, może przekazać określone polecenie do adresowanego procesora i otrzymać stosowną odpowiedź. Mogą to być takie polecenia, jak: przerwanie zewnętrzne, sygnał awaryjny, start procesora, stop procesora, zerwanie stanu procesora, testowanie procesora, pierwotne załadowanie programu itp. Procesor, do którego dane polecenie jest skierowane, informuje o możliwości przyjęcia i wykonania danego polecenia w formie odpowiedzi bezpośredniej (ładowanej do określonego rejestru ogólnego i do kodu warunków KW).

Urządzenia zewnętrzne (np. pamięć dyskowa) mogą być dzielone przez oba procesory systemu dwuprocesorowego (rys. 9.16). Oznacza to, że dostęp do nich jest możliwy ze strony każdego z procesorów. Jeśli zatem nastąpi uszkodzenie w danym procesorze, to przerwane przesyłania danych — które odbywały się za pośrednictwem procesora uszkodzonego — będą mogły być wznowione przez drugi procesor (tj. przez procesor pracujący).

10. Zgodność programowa i emulacja EMC ODRA 1300

10.1. Wprowadzenie

W obecnym etapie rozwoju elektronicznych maszyn cyfrowych występuje silna tendencja opracowania i produkcji zgodnych programowo EMC. Z drugiej strony opracowane EMC Jednolitego Systemu będą stopniowo zastępowały park istniejących maszyn, przeważnie należących do poprzednich generacji (zarówno pod względem strukturalnym jak i techniki realizacji). W związku z tym zachodzi konieczność rozwiązania problemu przeniesienia oprogramowania ze starych EMC na nowe EMC Jednolitego Systemu, tzn. wyposażenie nowych EMC Jednolitego Systemu w środki umożliwiające wykonanie większej części istniejącego oprogramowania starych EMC (np. serii ODRA 1300).

Podstawowym środkiem rozwiązania tego problemu, zastosowanym głównie w maszynach systemu 360 i 370 firmy IBM, jest metoda emulacji. Metoda emulacji zakłada istnienie specjalnych środków technicznych w nowej EMC, które to środki pozwalają na sprzętowo-programową interpretację instrukcji (rozkazów) starej EMC. Środki techniczne emulacji są przeważnie realizowane na poziomie mikrostruktury i zbioru mikrooperacji nowej EMC. Zatem mikroprogramowa interpretacja instrukcji starej EMC jest bardzo efektywna w stosunku do zwykłej programowej interpretacji oraz stosunkowo tania w realizacji (gdyż korzysta się w tej metodzie głównie ze środków przewidzianych do realizacji funkcji nowej EMC). Ponadto mikroprogramowa interpretacja pozwala na emulację kilku starszych EMC o różnej organizacji logicznej w jednej nowej EMC i to przy wykorzystaniu tych samych środków technicznych nowej EMC. Na ogół efektywność emulacji można podwyższyć przez zastosowanie w nowej EMC dodatkowych środków technicznych, specjalnie dostosowanych do specyficznej struktury informacji starej EMC. Mogą to być dodatkowe rejestry i wskaźniki wewnętrzne, dodatkowe układy lub dodatkowe mikrooperacje usprawniające działania określone przez instrukcje starej EMC.

EMC serii ODRA 1300 (tj. ODRA 1304, 1325, 1305), najbardziej rozpowszechnione w Polsce maszyny, posiadają duże oprogramowanie użytkowe z wielu dziedzin nauki i gospodarki narodowej. Szacuje się, że do roku 1979 będzie w eksploatacji w Polsce i w innych krajach socjalistycznych ponad 500 EMC serii ODRA 1300. Zatem będzie istniało duże zapotrzebowanie na przeniesienie oprogramowania tych maszyn na EMC Jednolitego Systemu. Potrzeba przeniesienia oprogramowania EMC ODRA 1300 będzie wzrastała w miarę, jak użytkownicy komputerów ODRA 1304 (zaś w latach następnych także i komputerów ODRA 1325 i 1305) będą instalowali w swoich ośrodkach obliczeniowych nowe EMC Jednolitego Systemu. Dlatego w rozwiązaniach technicznych i programowych kolejnych EMC Jednolitego Systemu przewiduje się środki do wykonania programów serii ODRA 1300 na poziomie instrukcji maszynowych.

10.2. Metody przenoszenia oprogramowania oraz trudności i efektywność przenoszenia

Jak wiadomo, program napisany w języku danej maszyny jest ciągiem przeważnie wzajemnie powiązanych ze sobą instrukcji określających jednoznacznie pracę EMC. Zatem program określa jednoznacznie algorytm rozwiązania danego problemu. Pracochłonność opracowania algorytmów (programów maszynowych) z różnych dziedzin nauki i gospodarki jest ogromna, dlatego efektywne wykorzystanie istniejącego oprogramowania ma duże znaczenie gospodarcze.

Przenoszenie programu (algorytmu) jednej EMC, zwanej maszyną źródłową lub „matką", na drugą EMC, zwaną „gościnną", można zrealizować różnymi sposobami. Należą do nich zarówno różne metody t r a n s l a c j i, programu, jak i metody m o d e l o w a n i a źródłowej EMC w gościnnej EMC. Celem translacji programu maszyny źródłowej jest automatyczne wygenerowanie nowego programu (na ogół różnego od źródłowego) dla maszyny gościnnej, który realizuje ten sam maszynowy algorytm liczenia, czyli daje identyczne wyniki obliczeń.

Translacja powoduje zamianę każdej instrukcji programu źródłowego na jedną lub więcej instrukcji maszny gościnnej, bardziej lub mniej niezależnie od pozostałych części programu. Natomiast w przypadku modelowania maszyna gościnna zachowuje się w czasie wykonywania programu źródłowego tak jak maszyna — matka. Zatem nie zachodzi potrzeba wcześniejszego przetłumaczenia programu maszyny źródłowej.

10.2.1. Trudności i efektywność metod translacji

W procesie automatycznego tłumaczenia programu źródłowego na odpowiedni program maszyny gościnnej spotyka się różne istotne trudności, które często czynią to tłumaczenie niemożliwym. Trudności te mogą polegać na:
— istnieniu programu lub części programu, który w pełni nie jest przetłumaczalny na język gościnnej maszyny;
— występowaniu idiomatycznych wyrażeń (które mają sens np. tylko przy określonych wartościach argumentów i określonych mechanizmach działania);
— występowaniu instrukcji, które są modyfikowane dopiero w procesie obliczeń (zatem struktura programu zmienia się dynamicznie);
— istnieniu specyficznych mechanizmów w maszynie źródłowej, których funkcji nie da się opisać jednoznacznie w języku maszyny gościnnej.

Metody translacji uwzględniają tylko początkową (statyczną) strukturę programu i na tej podstawie montują nowy program w języku maszyny gościnnej. Zatem montaż ten odbywa się bez uwzględnienia struktury dynamicznej programu źródłowego. Z tego względu wykonanie tak tłumaczonego programu może przebiegać inaczej w maszynie gościnnej i tym samym może prowadzić do różnych wyników obliczeń. Oprócz wyżej wymienionych trudności, metoda translacji jest mało efektywna, gdyż w praktyce każdą instrukcję maszyny źródłowej zamienia się przeważnie na kilka lub kilkanaście instrukcji maszyny gościnnej.

10.2.2. Metody modelowania i ich efektywność

Modelowanie jednej EMC w drugiej na poziomie instrukcji maszynowych może być dokonane trzema metodami:
a) metodą czystej symulacji,
b) metodą czystej emulacji,
c) metodą zwykłej emulacji, tj. zarówno środkami sprzętowymi jak i programowymi.

Metody modelowania różnią się od metod translacji tym, że każda instrukcja programu źródłowego jest kolejno interpretowana dopiero w czasie wykonywania programu. Zatem interpretacja instrukcji odbywa się nie raz (jak w metodzie translacji), lecz każdorazowo w czasie jej wykonania. Powoduje to oczywiście pewne obniżenie szybkości obliczeń, ale za to nie występują na ogół w metodzie modelowania te trudności, które często występują w metodzie translacji. Sprawa efektywności modelowania zależy od stopnia usprzętowienia, tzn. od ilości funkcji zrealizowanych środkami technicznymi.

Metoda czystej symulacji zakłada, że zarówno interpretacja — tj. pobranie, modyfikacja adresów i rozszyfrowanie typu instrukcji — jak i wykonanie podstawowej funkcji dowolnej instrukcji odbywa się wyłącznie za pomocą odpowiednich procedur programowych (czyli tzw. programów interpretacyjnych). Metoda ta jest łatwa do realizacji w dowolnej EMC, lecz daje ogromne (np. o cały rząd) zwolnienie szybkości wykonania programu symulowanej EMC (nawet w stosunku do szybkości źródłowej EMC, która na ogół jest zbudowana na wolniejszej technice).

Natomiast metoda czystej emulacji zakłada, że interpretacja i wykonanie wszystkich instrukcji oraz innych funkcji maszyny źródłowej są zrealizowane w maszynie gościnnej wyłącznie środkami sprzętowymi. W tej metodzie realizuje się na ogół dwa różne tryby pracy określone np. przez operatora:
— tryb EMULACJI, w którym wykonuje się wyłącznie programy źródłowej EMC;
— tryb RODZIMY, w którym wykonuje się wyłącznie programy danej gościnnej EMC.

W trybie EMULACJI gościnna EMC wykonuje dość sprawnie programy źródłowej (starej) EMC, lecz w rzeczywistości zamienia się w starą EMC i wówczas przez długi czas nie są wykorzystywane nowoczesne cechy i możliwości nowej EMC. Duże są również straty czasowe związane z nastawieniem odpowiedniego trybu pracy i załadowaniem odpowiedniego systemu operacyjnego.

Metoda trzecia, tj. zwykłej emulacji, zakłada, że wykonanie instrukcji i innych funkcji maszyny źródłowej odbywa się zarówno środkami sprzętowymi, jak i programowymi. Metoda zwykłej emulacji, w której można wyróżnić różne warianty, daje nie tylko zadowalające szybkości wykonania programu źródłowej EMC w nowej EMC (np. 2 do 5 razy szybciej niż w starej EMC), lecz także istotnie obniża ogólne koszty przenoszenia oprogramowania starej EMC na nową EMC oraz w pełni wykorzystuje wszystkie możliwości nowej EMC. Dlatego w rozwiązaniach EMC Jednolitego Systemu przewiduje się zastosowanie metody zwykłej emulacji (zwanej dalej po prostu metodą emulacji).

10.2.3. Metody emulacji

Poniżej rozpatrzono dwa warianty zwykłej emulacji, zależnie od sposobu organizacji systemu operacyjnego maszyny gościnnej.

A. **Sprzęt — oprogramowanie — własny system operacyjny**

W tym wariancie pewna ilość funkcji maszyny źródłowej (przeważnie funkcji najczęściej występujących w programach i nieskomplikowanych w realizacji) jest realizowana sprzętowo w gościnnej EMC, natomiast wszystkie pozostałe funkcje są realizowane programowo (przeważnie należą do nich rozkazy skomplikowane w realizacji technicznej, a rzadko występujące w programach, oraz rozkazy nielegalne, głównie rozkazy wprowadzania i wyprowadzania informacji). Zatem dzięki programowej realizacji funkcji wprowadzania i wyprowadzania informacji istnieje duża elastyczność w symulacji urządzeń zewnętrznych źródłowej EMC.

W wariancie tym zakłada się, że istnieją dwa tryby pracy: tryb EMULACJI i tryb RODZIMY. W trybie EMULACJI wykonuje się wyłącznie programy użytkowe emulowanej EMC, lecz pod kontrolą własnego systemu operacyjnego (w którym symuluje się programowo nie tylko pracę urządzeń zewnętrznych i nieemulowanych funkcji jednostki centralnej, lecz także i funkcje systemu operacyjnego źródłowej EMC). Natomiast w trybie RODZIMY wykonuje się wyłącznie pro-

gramy nowej EMC, oczywiście pod kontrolą „rodzimego" systemu operacyjnego. Zachodzi zatem potrzeba przeładowywania systemu operacyjnego — co jest związane z określonymi stratami czasowymi — przy przechodzeniu z jednego trybu na drugi. Oczywiście w przypadku trybu emulacji również nie wykorzystuje się wielu nowych urządzeń zewnętrznych, np. urządzenia teletransmisji, których odpowiedniki nie występowały w starej EMC przeważnie wcześniejsze konfiguracje były uboższe od nowych.

B. **Sprzęt — oprogramowanie — wspólny system operacyjny**

Również w tym wariancie pewne funkcje źródłowej EMC są realizowane sprzętowo, a inne programowo. Natomiast wspólny system operacyjny organizuje jednoczesne (na zasadzie wieloprogramowości) wykonanie programów użytkowych, zarówno emulowanej jak i emulującej EMC.

Firma IBM wykonała i przeanalizowała różne emulatory (jeszcze na systemie 360). W wyniku tych prac sformułowano pewne generalne wymagania co do emulatorów:

1) pełne zintegrowanie z systemami operacyjnymi, a praca taka jak w przypadku programów problemowych (użytkowych);
2) pełna wieloprogramowość, w tym również praca wieloprogramowa emulatorów;
3) pełna niezależność w przydzielaniu urządzeń zewnętrznych przez system operacyjny;
4) pełna zgodność struktur danych z punktu widzenia systemu operacyjnego;
5) praca w systemie „pojedynczy strumień zadań";
6) wspólna i modularna architektura emulatorów (dla ułatwienia konserwacji programów);
7) brak ograniczeń w wykonywaniu programów emulowanych (w tym możliwość przerwania wszystkich operacji).

W przypadku emulatorów typu A (tzw. emulatory autonomiczne) można dowolnie stosować nielegalne rozkazy danej EMC w programie emulatora. Rozkazów tych nie można natomiast stosować w przypadku, gdy dany emulator ma funkcjonować jako program problemowy pod kontrolą systemu operacyjnego. Dlatego te fragmenty emulatora, w których muszą być zastosowane rozkazy nielegalne (związane np. z wprowadzaniem danych), są przenoszone do systemu operacyjnego.

Ogólne wymagania sprzętowe. W celu realizacji wyżej wymienionych generalnych wymagań dotyczących emulatorów zintegrowanych, tj. emulatorów typu B, można sformułować pewne wymagania techniczne:

1) system operacyjny nie powinien „widzieć" programu emulator, lecz program problemowy; zatem eliminuje się wyróżnianie trybu pracy (emulacji);
2) emulator, tj. program interpretacyjny wraz ze wszystkimi jego tablicami oraz pełny obraz emulowanego programu źródłowego, zajmuje dowolny obszar pamięci;
3) instrukcje (operacje) emulatora muszą być standaryzowane;
4) instrukcje (operacje) emulatora muszą być tak zorganizowane, aby można było je przerywać i powtarzać w przypadku błędu;
5) komunikacja pomiędzy sprzętem emulacji a programem emulator powinna odbywać się wyłącznie za pośrednictwem rejestrów programowych, np. rejestrów ogólnych i zmiennoprzecinkowych (wynika to z potrzeby wieloprogramowej pracy emulatorów).

10.3. Ogólna koncepcja wykonania programu EMC serii ODRA 1300 w polskich EMC JS

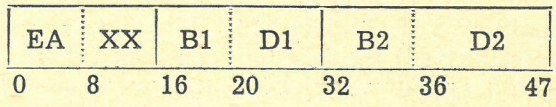
Opracowywane środki emulacji zakładają, że:

1) istnieje szereg interpretujących programów w emulującej EMC JS;
2) w celu zwiększenia efektywności wykonania programu źródłowego, tj. programu maszyny ODRA 1300, wprowadza się w emulującej EMC JS pewne nowe instrukcje dwóch typów:
   a) instrukcje dla emulacji najczęściej występujących instrukcji w programach serii ODRA 1300,
   b) instrukcje dla wywoływania odpowiednich mikroprogramów realizujących pewne operacje często występujące w programach interpretujących emulatora (mikroprogramy te zastępują odpowiednie podprogramy emulatora).

Zakłada się, że emulujące EMC JS posiadać będą mikroprogramowe sterowanie, przy czym pamięć mikroprogramów tych maszyn składać się będzie z części stałej i zapisywalnej. Część zapisywalna jest przeznaczona między innymi do przechowywania mikroprogramów emulacji ODRA 1300. W gościnnej maszynie JS EMC sprzęt emulacji składać się będzie z pewnej liczby specjalnych instrukcji dodanych do podstawowego repertuaru instrukcji EMC. Instrukcje te, tzw. instrukcje mulatora, będą użyte przez program mulator w celu zainicjowania określonych mikroprogramów realizujących funkcje różnych operacji maszyn serii ODRA 1300. Wybór operacji maszynowych EMC serii ODRA 1300, których wykonanie zapoczątkowują te specjalne instrukcje, jest wynikiem analizy złożoności i częstotliwości występowania tych operacji w programach serii ODRA 1300. W przypadku emulacji EMC serii ODRA 1300 (która to seria składa się z trzech różnych jednostek centralnych o szyfrach ODRA 1304, ODRA 1325 i ODRA 1305) przyjmuje się w emulującej EMC JS grupę instrukcji typu SS o kodzie podstawowym EA (zwanych instrukcjami emulatora). Drugi bajt, tj. bity 8÷15, tej instrukcji określa dalsze rozszerzenie kodu operacji.

Instrukcja EA powoduje wywołanie odpowiedniego mikroprogramu realizującego określoną instrukcję lub funkcję EMC ODRA 1304, 1325 lub 1305. Kody operacyjne EA 00, EA 01 i EA 02 powodują uruchomienie cyklu rozkazowego odpowiednio EMC ODRA 1304, 1305 i 1325. Natomiast kody operacyjne EA 03÷OF powodują uruchomienie odpowiedniego mikroprogramu realizującego funkcję przypisaną odpowiednim makroinstrukcjom emulatora.

Instrukcja EA o kodach dodatkowych (XX) od 00 do 02 ma postać:



gdzie adres efektywny E1=(B1)+D1 określa początek tzw. tablicy kodów operacyjnych (TKO). Tablica kodów operacyjnych składa się ogólnie z 128 słów; przechowuje się w niej adresy początkowe programów interpretacyjnych poszczególnych instrukcji EMC ODRA 1300 oraz różne dodatkowe informacje o danej instrukcji. Natomiast adres efektywny E2=(B2)+D2 określa początek wspólnego pola (WP), w którym przechowuje się różne informacje opisujące stan programu emulowanego oraz tablice konwersji kodów (przy czym z informacji tych korzystają zarówno programy interpretujące emulatora jak i mikroprogramy).

Instrukcja EA o kodach dodatkowych (XX) od EA 03 do OF ma postać:

EA	XX	00	R1 R2	B2	D2
0	8	16	20	32	36 47

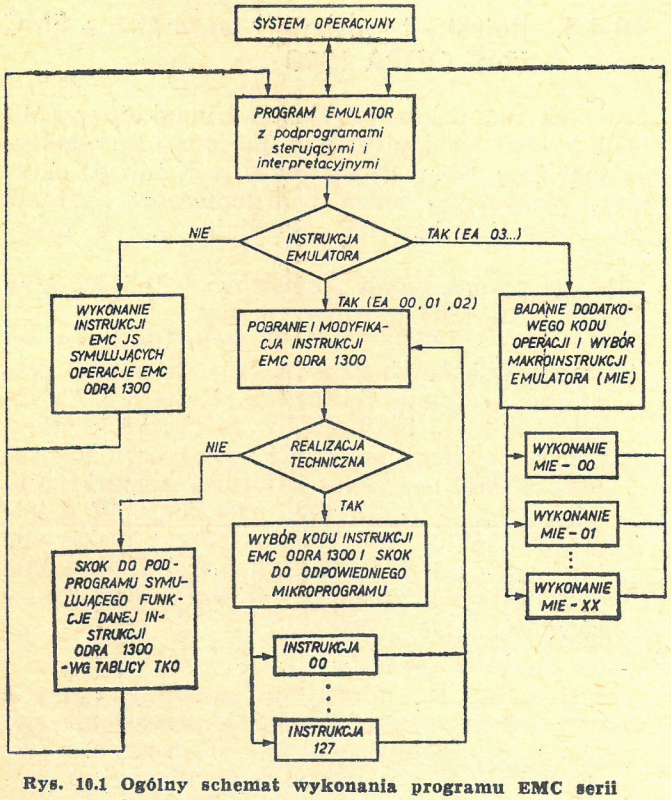
gdzie rejestry ogólne R1, R2 oraz adres E2=(B2)+D2 wykorzystuje się przy określeniu funkcji odpowiednich mikroprogramów makroinstrukcji emulatora.

Poniżej opisano dla przykładu funkcje instrukcji EA 1, która powoduje pobranie z pamięci operacyjnej i interpretację instrukcji EMC ODRA 1305:

a) badanie warunków przerwania; jeśli istnieje jakikolwiek warunek przerwania, to przechodzi się do procedury przerwania, jeśli nie — przechodzi się do pktu b;
b) pobranie zawartości symulowanego licznika rozkazów (LR) maszyny ODRA 1305;
c) pobranie instrukcji programu maszyny ODRA 1305 według wskazań LR;
d) rozszyfrowanie typu instrukcji maszyny ODRA 1305 i wykonanie występujących w tej instrukcji modyfikacji:
   — indeksowej (gdy część M instrukcji jest różna od zera),
   — premodyfikacji (gdy występuje instrukcja o kodzie 117),
   — relatywnej,
   — pośredniej,
   — datumowania,
   w tej samej kolejności jak w EMC ODRA 1305 oraz z tymi samymi ograniczeniami;
e) umieszczenie efektywnych adresów w odpowiednich ogólnych rejestrach emulującej maszyny EMC JS;
f) zaktualizowanie licznika rozkazów (maszyny ODRA 1305);
g) zinterpretowanie kodu operacji pobranej instrukcji i zorganizowanie skoku do:
   — odpowiedniego mikroprogramu emulowanej instrukcji ODRA 1305 lub
   — odpowiedniej programowej procedury interpretacyjnej (której adres określa tablica TKO) w przypadku napotkania instrukcji nie posiadającej technicznej realizacji.

Zbliżoną funkcję mają również instrukcje EA 00 EA 02 (związane odpowiednio z maszynami ODRA 1304 i 1325).

Rysunek 10.1 przedstawia ogólny schemat wykonania programu EMC ODRA 1300 w emulującej EMC Jednolitego Systemu.



Rys. 10.1 Ogólny schemat wykonania programu EMC serii ODRA 1300 w EMC Jednolitego Systemu

10.4. Ogólna charakterystyka struktur rozkazów i danych EMC serii ODRA 1300

Poniżej podano krótki opis cech i formatów informacji EMC ODRA 1300, które występują na poziomie emulacji.

Maszyny typu ODRA 1300 są maszynami uniwersalnymi o słowowej strukturze informacji, tzn. podstawową adresowalną informacją jest słowo 24-bitowe (plus jeden bit kontrolny). Słowo może przedstawiać: rozkaz, krótką liczbę stałoprzecinkową, parametr logiczny lub 4 znaki alfanumeryczne (kody wewnętrzne znaków alfanumerycznych mają po 6 bitów).

Maszyny serii ODRA 1300 pracują tylko w dwójkowym systemie liczenia (arytmetyka uzupełnieniowa). W maszynach ODRA 1300 wykonuje się działania zarówno na 24-bitowych liczbach stałoprzecinkowych jak i zmiennoprzecinkowych (48-bitowych).

10.4.1. Struktura rozkazów

Struktura rozkazu EMC ODRA 1300 zależy od stanu programowego, w którym znajduje się jednostka centralna. Wyróżnia się następujące stany programowe:

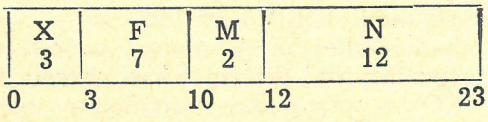
a) pamięć kompaktna (*Compact Store Mode*, skrót CSM); w tym stanie pamięć operacyjna jest traktowana jako blok o pojemności do 32 K słów (dla jednego programu), zaś adresy danego programu jako 15-bitowe;
b) pamięć rozszerzona (*Extended Store Mode*, skrót ESM); w tym stanie nie ma ograniczenia pamięci do 32 K słów, a adresy programu mają formalnie długość do 22 bitów;
c) skoki normalne (*Normal Jump Mode*, skrót NJM), dla których adres rozkazu ma 15 bitów;
d) skoki rozszerzone (*Extended Jump Mode*, skrót

EJM), dla których adres skuteczny może być rozszerzony do 22 bitów.

Stany CSM i ESM są określane przez wartość specjalnego jednobitowego rejestru ESM (ESM=0 oznacza CSM, ESM=1 oznacza ESM), natomiast stany NJM i EJM — przez inny jednobitowy rejestr EJM (EJM=0 oznacza NJM, EJM=1 oznacza EJM).

10.4.1.1. Struktura rozkazów arytmetycznych

Postać rozkazów arytmetycznych (normalnych) jest, niezależnie od stanu wskaźników ESM i EJM, następująca:



gdzie 24-bitowe słowo (bity ponumerowane są od 0—23) podzielone jest na części:

X — adres akumulatora (od 0 do 7),

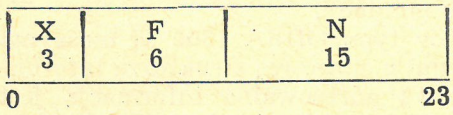
F — kod rozkazu,

M — adres modyfikatora (tj. akumulatora od X1 do X3; wielkość M=0 oznacza brak modyfikacji indeksowej);

N — adres pierwotny rozkazu (gdzie jeśli N⩽7, to N oznacza adres akumulatora).

10.4.1.2. Struktura rozkazów skokowych

A. W przypadku stanów CSM i NJM oraz ESM i NJM (skoki norm.):

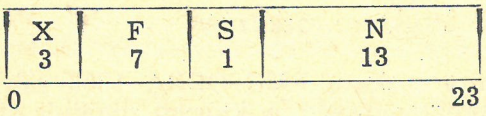


X — określa testowany akumulator lub warunek,

F — kod rozkazu (zawsze parzysty),

N — adres skoku (czyli adres komórki, w której znajduje się następny rozkaz do wykonania, w przypadku skoku efektywnego). Ponieważ N ma 15 bitów, można organizować skoki w ramach obszaru do 32 K słów. Adres ten może być rozszerzony do 22 bitów w przypadku ESM i premodyfikacji.

B. W przypadku stanu CSM i EJM oraz ESM i EJM:



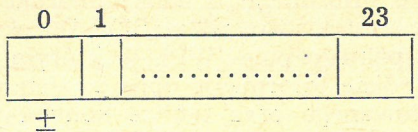
X — określa testowany akumulator lub warunek,

F — kod rozkazu (może być zarówno parzysty, jak i nieparzysty); jeśli F jest parzysty, oznacza skok RELATYWNY, jeśli zaś nieparzysty — POŚREDNI.

W przypadku skoku relatywnego adres skoku efektywnego wynosi LR+N (dla S=0) lub LR−N (dla S=1), gdzie skrót LR oznacza stan licznika rozkazów. W przypadku skoku pośredniego adres skuteczny skoku efektywnego określa 22-bitowa zawartość komórki o adresie SN (14-bitowym), co zapewnia skoki w obszarze powyżej 32 K (do 4 M słów).

10.4.2. Struktura liczb stałoprzecinkowych

Liczba stałoprzecinkowa zajmuje jedno słowo 24-bitowe. W pozycji 0 tego słowa przechowuje się znak liczby, zaś w pozycjach 1÷23 cyfry znaczące (w arytmetyce uzupełnieniowej):

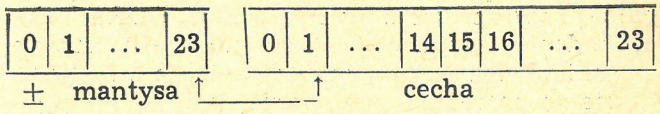


10.4.3. Struktura liczb zmiennoprzecinkowych

W EMC ODRA 1300 istnieje również drugi sposób przedstawienia liczby, zawierający informację o miejscu przecinka w liczbie. Sposób ten polega na przedstawieniu liczby x w postaci pary (m, c), gdzie $x=m\cdot 2^c$ oraz c to liczby całkowite z przedziału od -256 do $+255$; zaś m — liczba spełniająca warunek:

$$-1 \leqslant m \leqslant 1-2^{-37}$$

Liczby c i m są nazwane odpowiednio cechą i mantysą liczby x:



Bit zerowy określa znak mantysy (0 — plus, 1 — minus), a bity od 1 do 23 słowa pierwszego i od 1 do 14 słowa drugiego określają cyfry mantysy po przecinku. Bit 15 określa znak cechy, zaś bity od 16 do 23 słowa drugiego określają kolejne cyfry znaczące cechy z przesunięciem +256).

Z przyjętej struktury liczby zmiennoprzecinkowej x wynika, że należy ona do przedziału: -2^{255}; $+2^{255}$.

10.4.4. Rejestry i indykatory wewnętrzne EMC EMC ODRA 1300

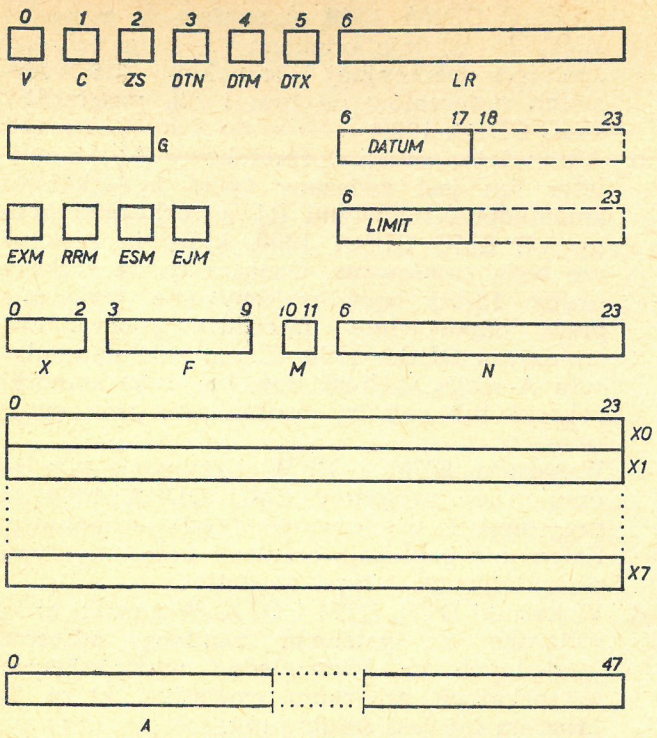
Rysunek 10.2 pokazuje rejestry i indykatory EMC ODRA 1305 występujące na poziomie architektury logicznej. Stany rejestrów (np. *X* lub *A*) określają przeważnie jeden z argumentów operacji, zaś stany indykatorów określają sposób pracy EMC.

Główne przeznaczenie rejestrów i indykatorów EMC ODRA 1300 jest następujące:

a. Rejestry DATUM i LIMIT są to 18-bitowe rejestry, w których przechowuje się adresy graniczne programu bieżącego. Rejestr DATUM zawiera adres bezwzględny pierwszej komórki pamięci operacyjnej danego programu, zaś rejestr LIMIT — adres ostatniej komórki (powiększony o jeden) tego programu. W czasie pracy wszelkie bezwzględne adresy efektywne (rozkazów lub argumentów operacji) aktualnie pracującego programu powinny spełniać nierówność:

$$\text{DATUM} \leqslant \text{ADRES} < \text{LIMIT}$$

Pojawienie się adresu nie spełniającego powyższej nierówności powoduje przerwanie bieżącego programu i przejście do programu EXECUTIVE. W ten sposób zapewniona jest protekcja programów.



Rys. 10.2. Rejestry i indykatory wewnętrzne maszyny ODRA 1305

b. Rejestr licznika rozkazów LR. Rejestr ten zawiera 18 pozycji; przechowuje się w nim adres bezwzględny kolejnego rozkazu do wykonania.
c. Rejestr rozkazu składa się z części:
   X — do pamiętania adresu akumulatora ogólnego (0÷7),
   F — do pamiętania funkcji rozkazu (000÷177),
   M — do pamiętania adresu modyfikatora (1÷3),
   N — do pamiętania adresu argumentu operacji (0÷4095) bez modyfikacji lub 0÷262144 po modyfikacjach).
d. Akumulator zmiennoprzecinkowy A (48 bitów).
e. Indykator nadmiaru V i indykator przeniesienia C.
f. Indykatory EXM i PRM opisujące stan programowy, w którym znajduje się w danej chwili procesor:

EXM	PRM	Stan procesora
0	0	stan programu normalnego
0	1	stan programu priorytetowego
1	0	stan programu executive
1	1	stan programu priorytetowego

   Zatem procesor może znajdować się w jednym z trzech stanów. Przechodzi on w stan programu priorytetowego, gdy otrzyma specjalne przerwanie od tzw. kanału priorytetowego.
g. Wskaźnik ESM i EJM; uruchamiają one dodatkowe mechanizmy w procesorze, w przypadku gdy pojemność danego programu przekracza 32 K słów maszynowych.
h. Rejestr G (4-bitowy), w którym pamięta się numer binarny aktualnie pracującego programu głównego.
i. Wskaźnik ZS, DTN, DTM i DTX. Wskaźnik ZS służy do likwidacji zer początkowych liczb dziesiętnych otrzymanych za pomocą specjalnych rozkazów konwersji z układu binarnego na układ dziesiętny. Natomiast wskaźnik DTN, DTM i DTX (gdy ich stan równa się 1) powodują, że adresy N, M i X występujące w rozkazach programu executive są automatycznie powiększane o zawartość rejestru DT. Czyli w tych przypadkach executive kontaktuje się z komórkami programu użytkowego (o adresach: N+DT, M+DT, X+DT).

10.5. Rozmieszczenie informacji EMC serii ODRA 1300 w EMC JS

Każde 24-bitowe słowo EMC ODRA 1300 zostaje rozmieszczone w jednym słowie (32-bitowym) EMC JS, tj. w bitach od 0 do 23, przy czym bit 0 słowa EMC ODRA 1300 pokrywa się z bitem 0 słowa EMC JS. Zatem nie wykorzystuje się w emulacji bitów od 24 do 31 słowa EMC JS (wykorzystanie tych bitów prowadziłoby do znacznych strat czasowych w wykonywaniu programu EMC ODRA 1300). Bity 24÷31 zawierają zawsze zera. Informacja będąca obrazem pamięci EMC ODRA 1300 jest rozmieszczana w pamięci operacyjnej i rejestrach ogólnych EMC JS w sposób jednakowy zarówno dla wielkości liczbowych jak i rozkazów.

10.5.1. Rozmieszczenie akumulatorów X0÷X7

Obszar dowolnego programu EMC ODRA 1300 jest zawsze adresowany od 0 z punktu widzenia programisty (czyli obowiązuje adresacja relatywna). Proces zamiany adresów relatywnych na rzeczywiste odbywa się przez dodawanie zawartości rejestru DATUM do wszystkich relatywnych adresów — tj. adresów N, M, X oraz LR dla skoków — przy zwrotach do pamięci operacyjnej. Początkowych 8 komórek każdego programu, tj. komórki o relatywnych adresach od 0 do 7, są wykorzystywane jako akumulatory i adresowane są również przez część X lub M rozkazu ODRA 1300.

W rozwiązaniach emulacji przyjmuje się, że w tych komórkach — tj. komórkach o adresach 0÷7 — przechowuje się stan akumulatorów X0÷X7 tylko wtedy, gdy program EMC ODRA 1300 jest nieaktywny. Natomiast w czasie pracy programu EMC ODRA 1300 rolę akumulatorów X0÷X7 przyjmują na siebie ogólne rejestry EMC JS o adresach od 0 do 7. Zatem w czasie działania programu EMC ODRA 1300 zwroty do akumulatorów X0÷X7 (tj. przy X, M lub $N < 8$) są kierowane do rejestrów ogólnych o numerach 0÷7. Program emulator ODRA 1300 przepisuje zawartość początkowych komórek obszaru pamięci, będącego obrazem pamięci EMC ODRA 1300, do odpowiednich rejestrów ogólnych EMC JS przed pierwszym uruchomieniem danego programu emulowanego.

10.5.2. Rozmieszczenie licznika rozkazów

Licznik rozkazów EMC ODRA 1300 może przedstawiać wielkości do 18 bitów i jest przechowywany w czasie emulacji w ogólnym rejestrze

o numerze 8, tj. w bitach 6÷23, zaś w bitach 0÷5 oraz 24÷31 przechowuje się zera.

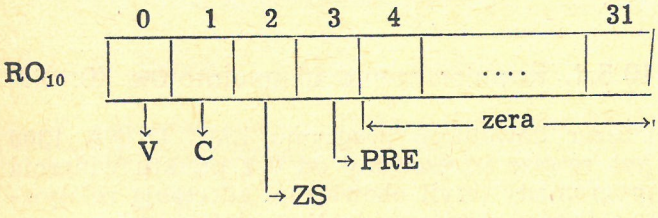
Stan licznika rozkazów zawiera zawsze adres rzeczywisty (tj. po uwzględnieniu stanu rejestru DATUM).

10.5.3. Rozmieszczenie rejestru DATUM

Rejestr DATUM, tj. rejestr dolnego granicznego adresu programu użytkowego EMC ODRA 1300, jest przechowywany w czasie emulacji w ogólnym rejestrze o numerze 9 (w bitach 6÷17), zaś w bitach 0÷5 i 18÷31 tego rejestru ogólnego wpisuje się zera. W czasie działania programu emulowanego EMC ODRA 1300, adresy względne N są zamieniane na bezwzględne przez dodanie do nich stanu rejestru DATUM.

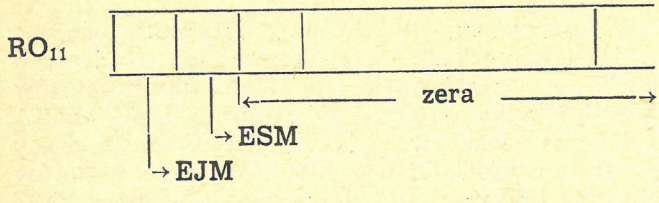
10.5.4. Wskaźniki V, C, ZS, PRE

Wskaźniki programowe nadmiaru (V), przeniesienia (C) i likwidacji zer (ZS) oraz wskaźnik premodyfikacji (PRE) są przechowywane w czasie emulacji kolejno w bitach 0÷3 rejestru ogólnego o numerze 10 (RO_{10}):



10.5.5. Wskaźniki EJM i ESM

Wskaźnik trybu „pamięć rozszerzona" (ESM) oraz wskaźnik trybu „skoki rozszerzone" (EJM) są przechowywane w czasie emulacji odpowiednio w bicie 1 i 0 rejestru ogólnego o numerze 11 (RO_{11}):



10.5.6. Wskaźniki i rejestry niesymulowane

W poprzednich punktach tego opisu podano rejestr i wskaźniki EMC ODRA 1300, które są symulowane w rejestrach EMC JS w czasie emulacji.

Niżej wymienione wskaźniki i rejestry wewnętrzne EMC ODRA 1300, występujące na poziomie architektury logicznej tej maszyny, nie zostały „zanurzone" w rejestrach technicznych EMC JS:

a. Rejestr LIMIT. Rejestr ten jest związany z mechanizmem ochrony pamięci operacyjnej w EMC ODRA 1300, natomiast w maszynie EMC JS istnieje odmienny system ochrony „pamięci operacyjnej i jednakowy dla wszystkich programów, w tym i dla programów EMC ODRA 1300.
b. Akumulator zmiennoprzecinkowy A. Akumulator ten jest związany tylko z rozkazami zmiennoprzecinkowymi (tj. z rozkazami grupy 13) EMC ODRA 1300, które to rozkazy nie będą emulowane. Oznacza to, że rozkazy grupy 13-tej będą wykonywane wyłącznie przez odpowiednie procedury programowe emulatora ODRA 1300. Zatem rolę akumulatora A mogą spełniać dowolne dwie komórki robocze lub rejestry ogólne programu emulator.
c. Wskaźniki EXM i PRM. Przyjmuje się, że emulowane programy EMC ODRA 1300 są programami użytkowymi typu normalnego. Zatem nie podlegają emulacji programy typu pirorytetowego i typu executive.
d. Wskaźniki DTN, DTM i DTX. Wskaźniki te są związane z systemem zamiany adresów względnych na bezwzględne występujących w rozkazach programu executive, który to program nie jest emulowany.

10.5.7. Rozmieszczenie elementów rozkazu EMC ODRA 1300

W rozkazie EMC ODRA 1300 występują następujące części:
— część X (adres akumulatora),
— część M (adres modyfikatora),
— część F (kod rozkazu),
— adres pierwotny N (bez modyfikacji),
— adres efektywny Ne (po ew. modyfikacjach).

W czasie działania emulacji mikroprogram pobrania i interpretacji rozkazu (MIR) EMC ODRA 1300 rozpakowuje ten rozkaz i wyznacza efektywny adres tego rozkazu (patrz też rys. 10.1), po czym przechodzi się do wykonania funkcji podstawowej danego rozkazu za pomocą środków technicznych (mikroprogramów) lub programowych. W wyniku działania mikroprogramu MIR elementy rozpatrywanego rozkazu zostają rozmieszczone w rejestrach technicznych EMC JS (występujących na poziomie mikrostruktury), zaś adres efektywny Ne w rejestrze ogólnym nr 12.

Jeśli rozpatrywany rozkaz ma realizację techniczną (mikroprogramową), to wielkości zapisane w rejestrach technicznych są wyjściowymi dla mikroprogramu realizującego funkcję podstawową danego rozkazu. Jeśli natomiast dany rozkaz ma realizację programową, to można wykorzystać tylko wielkości zapisane w rejestrach ogólnych 8 i 12. Oznacza to, że jeśli w danym podprogramie interpretacyjnym emulatora ODRA 1300 potrzebne są początkowe elementy rozkazu X, F, M i N, to można je uzyskać według stanu rejestru ogólnego nr 8 (tj. licznika rozkazu). Natomiast adres efektywny tego rozkazu, który znajduje się w rejestrze ogólnym 12, jest łatwo dostępny (np. przez modyfikację indeksową względem tego rejestru).

Lista instrukcji JS EMC RIAD-1

Poniżej podano wykaz podstawowych instrukcji maszynowych Jednolitego Systemu EMC, uporządkowanych według numerycznego kodu operacji. Wykaz ten zawiera angielski skrót literowy danej instrukcji — występujący w językach programowania — oraz typ instrukcji, polską nazwę instrukcji i symboliczny zapis funkcji wykonywanej przez rozpatrywaną instrukcję. W wykazie zastosowano niektóre skróty, których znaczenie opisano na końcu dodatku.

Kod	Skrót	Typ	Nazwa	Funkcja
1	2	3	4	5
04	SPM	RR	USTAW MASKĘ PROGRAMU	$(R1)_{2-7} \rightarrow SSP$
05	BALR	RR	SKOCZ ZE ŚLADEM	$(LI) \rightarrow R1$, $(R2) \rightarrow LI$
06	BCTR	RR	SKOCZ WG LICZNIKA	$(R1)-1 \rightarrow R1$, $(R2) \rightarrow LI$, gdy $(R1) \neq 0$
07	BCR	RR	SKOCZ WG WARUNKU	$(R2) \rightarrow LI$, wg maski R1
08	SSK	RR	USTAW KLUCZ PAMIĘCI	$(R1) \rightarrow$ KLUCZ wg (R2)
09	ISK	RR	CZYTAJ KLUCZ PAMIĘCI	KLUCZ wg $(R2) \rightarrow R1$
0A	SVC	RR	WYWOŁAJ SUPERVISOR	$R1\ R2 \rightarrow SSP_{24-31}$; stare $SSP \rightarrow 32-39$, $(96—103) \rightarrow SSP$
10	LPR	RR	ŁADUJ DODATNIO STP	$\lvert(R2)\rvert \rightarrow R1$
11	LNR	RR	ŁADUJ UJEMNIE STP	$-\lvert(R2)\rvert \rightarrow R1$
12	LTR	RR	ŁADUJ I TESTUJ STP	$(R2) \rightarrow R1$
13	LCR	RR	ŁADUJ UZUPEŁNIENIE STP	$-(R2) \rightarrow R1$
14	NR	RR	MNÓŻ LOGICZNIE	$(R1) \wedge (R2) \rightarrow R1$
15	CLR	RR	PORÓWNAJ LOGICZNIE	$(R1)-(R2) \gtrless 0$
16	OR	RR	SUMUJ LOGICZNIE	$(R1) \vee (R2) \rightarrow R1$
17	XR	RR	RÓŻNICA SYMETRYCZNA	$(R1) \div (R2) \rightarrow R1$
18	LR	RR	ŁADUJ STP	$(R2) \rightarrow R1$
19	CR	RR	PORÓWNAJ ALGEBR. STP	$(R1)—(R2) \gtrless 0$
1A	AR	RR	DODAJ STP	$(R1)+(R2) \rightarrow R1$
1B	SR	RR	ODEJMIJ STP	$(R1)-(R2) \rightarrow R1$
1C	MR	RR	MNÓŻ STP	$(R1) \cdot (R2) \rightarrow R1$ *
1D	DR	RR	DZIEL STP	$(R1)\ *:(R2) \rightarrow R1$ *
1E	ALR	RR	DODAJ WIELKOŚCI LOG.	$(R1)+(R2) \rightarrow R1$
1F	SLR	RR	ODEJMIJ WIELKOŚCI LOG.	$(R1)-(R2) \rightarrow R1$
20	LPDR	RR	ŁADUJ DODATNIO ZMP D	$\lvert(R2)\rvert \rightarrow R1$
21	LNDR	RR	ŁADUJ UJEMNIE ZMP D	$-\lvert(R2)\rvert \rightarrow R1$
22	LTDR	RR	ŁADUJ I TESTUJ ZMP D	$(R2) \rightarrow R1$
23	LCDR	RR	ŁADUJ UZUPEŁN. ZMP D	$-(R2) \rightarrow R1$
24	HDR	RR	DZIEL PRZEZ 2 ZMP D	$(R2):2 \rightarrow R1$
28	LDR	RR	ŁADUJ ZMP D	$(R2) \rightarrow R1$
29	CDR	RR	PORÓWNAJ ZMP D	$(R1)-(R2) \gtrless 0$
2A	ADR	RR	DODAJ I NORM. ZMP D	$(R1)+(R2) \rightarrow R1$
2B	SDR	RR	ODEJMIJ I NORM. ZMP D	$(R1)-(R2) \rightarrow R1$
2C	MDR	RR	MNÓŻ ZMP D	$(R1) \cdot (R2) \rightarrow R1$
2D	DDR	RR	DZIEL ZMP D	$(R1):(R2) \rightarrow R1$
2E	AWR	RR	DODAJ BEZ NORM. ZMP D	$(R1)+(R2) \rightarrow R1$
2F	SWR	RR	ODEJMIJ BEZ NORM. ZMP D	$(R1)-(R2) \rightarrow R1$
30	LPER	RR	ŁADUJ DODATNIO ZMP K	$\lvert(R2)\rvert \rightarrow R1$
31	LNER	RR	ŁADUJ UJEMNIE ZMP K	$-\lvert(R2)\rvert \rightarrow R1$
32	LTER	RR	ŁADUJ I TESTUJ ZMP K	$(R2) \rightarrow R1$
33	LCER	RR	ŁADUJ UZUPEŁN. ZMP K	$(R2) \rightarrow R1$
34	HER	RR	DZIEL PRZEZ 2 ZMP K	$(R2):2 \rightarrow R1$

1	2	3	4	5
38	LER	RR	ŁADUJ ZMP K	(R2) → R1
39	CER	RR	PORÓWNAJ ZMP K	(R1) − (R2) ≷ 0
3A	AER	RR	DODAJ I NORM. ZMP K	(R1) + (R2) → R1
3B	SER	RR	ODEJMIJ I NORM. ZMP K	(R1) − (R2) → R1
3C	MER	RR	MNÓŻ ZMP K	(R1) · (R2) → R1
3D	DER	RR	DZIEL ZMP K	(R1) : (R2) → R1
3E	AUR	RR	DODAJ BEZ NORM. ZMP K	(R1) + (R2) → R1
3F	SUR	RR	ODEJMIJ BEZ NORM. ZMP K	(R1) − (R2) → R1
40	STH	RX	PAMIĘTAJ PÓŁSŁOWO STP	$(R1)_{16-31} \rightarrow E2$
41	LA	RX	ŁADUJ ADRES	E2 → R1
42	STC	RX	PAMIĘTAJ ZNAK STP	$(R1)_{24-31} \rightarrow E2$
43	JC	RX	PRZECZYTAJ ZNAK STP	$(E2)_{24-31} \rightarrow R1$
44	EX	RX	WYKONAJ INSTRUKCJĘ ZAPISANĄ W POLU E2	
45	BAL	RX	SKOCZ ZA ŚLADEM	(LI) → R1, E2 → LI
46	BCT	RX	SKOCZ WG LICZNIKA	(R1) − 1 → R1, E2 → LI, gdy (R1) ≠ 0
47	BC	RX	SKOCZ WG WARUNKU	E2 → LI wg maski R1
48	LH	RX	ŁADUJ PÓŁSŁOWO STP	$(E2)_{16-31} \rightarrow R1$
49	CH	RX	PORÓWNAJ PÓŁSŁOWO STP	(R1) − (E2) ≷ 0
4A	AH	RX	DODAJ PÓŁSŁOWO STP	(R1) + (E2) → R1
4B	SH	RX	ODEJMIJ PÓŁSŁOWO STP	(R1) − (E2) → R1
4C	MH	RX	MNÓŻ PÓŁSŁOWO STP	(R1) · (E2) → R1
4E	CVD	RX	ZAMIEŃ NA LICZBĘ DZIES.	$(R1)_{bin} \rightarrow E2_{dzies.}$
4F	CVB	RX	ZAMIEŃ NA LICZBĘ BINAR.	$(E2)_{dzies.} \rightarrow R1_{bin}$
50	ST	RX	PAMIĘTAJ STP	(R1) → E2
54	N	RX	MNÓŻ LOGICZNIE	(R1) ∧ (E2) → R1
55	CL	RX	PORÓWNAJ LOGICZNIE	(R1) − (E2) ≷ 0
56	O	RX	SUMUJ LOGICZNIE	(R1) ∨ (E2) → R1
57	X	RX	RÓŻNICA SYMETRYCZNA	(R1) ÷ (E2) → R1
58	L	RX	ŁADUJ STP	(E2) → R1
59	C	RX	PORÓWNAJ ALGEB. STP	(R1) − (E2) ≷ 0
5A	A	RX	DODAJ STP	(R1) + (E2) → R1
5B	S	RX	ODEJMIJ STP	(R1) − (E2) → R1
5C	M	RX	MNÓŻ STP	(R1) · (E2) → R1 *
5D	D	RX	DZIEL STP	(R1) *: (E2) → R1 *
5E	AL	RX	DODAJ WIELKOŚCI LOGICZNE	(R1) + (E2) → R1
5F	SL	RX	ODEJMIJ WIELKOŚCI LOGICZNE	(R1) − (E2) → R1
60	STD	RX	PAMIĘTAJ ZMP D	(R1) → E2
68	LD	RX	ŁADUJ ZMP D	(E2) → R1
69	CD	RX	PORÓWNAJ ZMP D	(R1) − (E2) ≷ 0
6A	AD	RX	DODAJ I NORM. ZMP D	(R1) + (E2) → R1
6B	SD	RX	ODEJMIJ I NORM. ZMP D	(R1) − (E2) → R1
6C	MD	RX	MNÓŻ ZMP D	(R1) · (E2) → R1
6D	DD	RX	DZIEL ZMP D	(R1) : (E2) → R1
6E	AW	RX	DODAJ BEZ NORM. ZMP D	(R1) + (E2) → R1
6F	SW	RX	ODEJMIJ BEZ NORM. ZMP D	(R1) − (E2) → R1
70	STE	RX	PAMIĘTAJ ZMP K	(R1) → E2
78	LE	RX	PAMIĘTAJ ZMP K	(E2) → R1
79	CE	RX	PORÓWNAJ ZMP K	(R1) − (E2) ≷ 0
7A	AE	RX	DODAJ I NORM. ZMP K	(R1) + (E2) → R1
7B	SE	RX	ODEJMIJ I NORM. ZMP K	(R1) − (E2) → R1
7C	ME	RX	MNÓŻ ZMP K	(R1) · (E2) → R1 *
7D	DE	RX	DZIEL ZMP K	(R1) : (E2) → R1
7E	AU	RX	DODAJ BEZ NORM. ZMP K	(R1) + (E2) → R1
7F	SU	RX	ODEJMIJ BEZ NORM. ZMP K	(R1) − (E2) → R1
80	SSM	SI	USTAW MASKĘ SYSTEMU	$(E1) \rightarrow SSP_{0-7}$
82	LPSW	SI	ŁADUJ SSP	(E1) * → SSP
83	DIG	SI	DIAGNOZA (FUNKCJA ZALEŻY OD MODELU)	
84	WRD	SI	PISZ BEZPOŚREDNIO	
85	RDD	SI	CZYTAJ BEZPOŚREDNIO	
86	BXH	RS	SKOCZ PRZY WIĘKSZYM INDEKSIE	(R1) + (R3) → R1 (R1) − (R3) ≦ 0 oraz E2 → LI, gdy (R1) > (R3)

1	2	3	4	5
87	BXLE	RS	SKOCZ PRZY MNIEJSZYM LUB RÓWNYM INDEKSIE	$(R1)+(R3) \rightarrow R1$, $(R1)-(R3) \leqslant 0$ oraz $E2 \rightarrow LI$, gdy $(R1) \nwarrow (R3)$
88	SRL	RS	PRZESUŃ W PRAWO LOG. K	$(R1):2^{E2} \rightarrow R1$
89	SLL	RS	PRZESUŃ W LEWO LOG. K	$(R1)\cdot 2^{E2} \rightarrow R1$
8A	SRA	RS	PRZESUŃ W PRAWO ALG. K	$(R1):2^{E2} \rightarrow R1$
8B	SIA	RS	PRZESUŃ W LEWO ALG. K	$(R1)\cdot 2^{E2} \rightarrow R1$
8C	SRDL	RS	PRZESUŃ W PRAWO LOG. D	$(R1)*:2^{E2} \rightarrow R1*$
8D	SIDL	RS	PRZESUŃ W LEWO LOG. D	$(R1)*\cdot 2^{E2} \rightarrow R1*$
8E	SRDA	RS	PRZESUŃ W PRAWO ALG. D	$(R1)*:2^{E2} \rightarrow R1*$
8F	SLDA	RS	PRZESUŃ W LEWO ALG. D	$(R1)*\cdot 2^{E2} \rightarrow R1*$
90	STM	RS	PAMIĘTAJ WIELOKROTNIE	$(R1), \ldots, (R3) \rightarrow E2$
91	TM	SI	TESTUJ WG MASKI	$(E1) \wedge I2 \rightarrow KW$
92	MVI	SI	PRZENIEŚ PARAMETR	$I2 \rightarrow E1$
93	TS	SI	TESTUJ I USTAW	$(E2)_0 \rightarrow KW$
94	NI	SI	MNÓŻ LOGICZNIE ZNAKI	$(E1) \wedge I2 \rightarrow E1$
95	CLI	SI	PORÓWNAJ LOGICZ. ZNAKI	$(E1)-I2 \gtrless 0$
96	OI	SI	SUMUJ LOGICZNIE ZNAKI	$(E1) \vee I2 \rightarrow E1$
97	XI	SI	RÓŻNICA SYMETR. ZNAKÓW	$(E1) \div I2 \rightarrow E1$
98	LM	RS	ŁADUJ WIELOKROTNIE	$(E2) \rightarrow R1, \ldots, R3$
9C	SIO	SI	START WE-WY	
9D	TIO	SI	TESTUJ WE-WY	
9E	HIO	SI	STOP WE-WY	
9F	TCH	SI	TESTUJ KANAŁ	
D1	MVN	SS	PRZENIEŚ POLE CYFR	$(E2+j)_{4-7} \rightarrow E1+j$, dla $j = 0,1, \ldots, L-1$
D2	MVC	SS	PRZENIEŚ POLE ZNAKÓW	$(E2+j) \rightarrow E1+j$
D3	MVZ	SS	PRZENIEŚ POLE STREF	$(E2+j)_{0-3} \rightarrow E1+j$
D4	NC	SS	MNÓŻ LOGICZNIE POLA	$(E2+j) \wedge (E1+j) \rightarrow E1+j$
D5	CLC	SS	PORÓWNAJ LOGICZNIE POLA	$(E2+j)-(E1+j) \leqslant 0$
D6	OC	SS	SUMUJ LOGICZNIE POLA	$(E2+j) \vee (E1+j) \rightarrow E1+j$
D7	XC	SS	RÓŻNICA SYMETR. PÓL	$(E2+j) \div (E1+j) \rightarrow E1+j$
DC	TR	SS	TŁUMACZ	
DD	TRT	SS	TŁUMACZ I TESTUJ	
DE	ED	SS	REDAGUJ	
DF	EDNK	SS	REDAGUJ I TESTUJ	
F1	MVO	SS	PRZESUŃ I PRZENIEŚ	$(E2, L2) \rightarrow E1, L1$
F2	PACK	SS	PAKUJ	$(E2, L2) \rightarrow E1, L1$
F3	UNPK	SS	ROZPAKUJ	$(E2, L2) \rightarrow E1, L1$
F8	ZAP	SS	ZERUJ I DODAJ	$(E2, L2) \rightarrow E1, L1$
F9	CP	SS	PORÓWNAJ DZIESIĘTNIE	$(E1, L1)-(E2, L2) \gtrless 0$
FA	AP	SS	DODAJ DZIESIĘTNIE	$(E1, L1)+(E2, L2) \rightarrow E1, L1$
FB	SP	SS	ODEJMIJ DZIESIĘTNIE	$(E1, L1)-(E2, L2) \rightarrow E1, L1$
FC	MP	SS	MNÓŻ DZIESIĘTNIE	$(E1, L1)\cdot(E2, L2) \rightarrow E1, L1$
FD	DP	SS	DZIEL DZIESIĘTNIE	$(E1, L1):(E2, L2) \rightarrow E1, L1$

Skróty i oznaczenia:

STP — stałoprzecinkowo, tzn. argumentami działania są liczby stałoprzecinkowe (całkowite).

ZMP — zmiennoprzecinkowo, tzn. argumentami działania są liczby zmiennoprzecinkowe (K — krótkie, D — długie).

* — oznacza, że dana wielkość jest długa (64-bitowa) i że adres pierwszego rejestru ogólnego jest parzysty; jeśli w nazwie instrukcji występuje skrót D (liczba długa), to w zapisie symbolicznym funkcji opuszcza się tę gwiazdkę.

LI — licznik instrukcji, tj. bity 40÷63 słowa podwójnego SSP.

NORM — normalizacja wyniku operacji.

(R1), (R2) — oznaczają odpowiednio zawartość rejestru ogólnego, którego adres określa pole R1 (odpowiednio R2).

$(R1)_{2-7} \rightarrow SSP$ — oznacza przesłanie do rejestru SSP wielkość zapisaną w pozycjach od 2 do 7 rejestru ogólnego, którego adres podaje pole R1 rozpatrywanej instrukcji.

(E2) — oznacza zawartość pola pamięci o stałej długości i o początkowym adresie efektywnym E2.

Metoda obliczania wydajności procesorów JS EMC (interpretacja mieszanki Gibson I)

Niniejszy dodatek podaje interpretację metody GIBSON I, przyjętą w Jednolitym Systemie EMC. Metoda GIBSON I [11] jest mieszanką statystyczną i opisuje procentowy (wagowy) udział poszczególnych operacji maszynowych w programach zastosowań naukowo-technicznych. Oryginalna postać tej mieszanki jest bardzo ogólnie sformułowana i nie daje podstaw do precyzyjnego porównywania komputerów różnych firm, w których na ogół stosuje się różne formaty informacji, a także przyporządkowuje się instrukcjom maszynowym — o tej samej nazwie ogólnej — różne funkcje szczegółowe. Sprawa natomiast upraszcza się, gdy mamy do czynienia z rodziną komputerów, w której obowiązuje jednolita struktura formatów informacji i jednolita lista instrukcji użytkowych.

W przypadku EMC Jednolitego Systemu przyjęto interpretację metody GIBSON I, gdzie pod ogólne nazwy typów operacji podstawiono konkretne instrukcje JS EMC (patrz tabela B.1).

Tabela B.1. INTERPRETACJA MIESZANKI GIBSON I

Lp.	Typ operacji (kod instrukcji)	Waga w_i	Prawdopodobieństwo p_i
1	2	3	4
1	Dzielenia zmiennoprzecinkowe:	1,6	—
1.1	DE	0,3	0,8
	DER		0,2
1.2	DD	1,3	0,8
	DDR		0,2
2	Mnożenia zmiennoprzecinkowe:	4,0	—
2.1	ME	0,8	0,8
	MER		0,2
2.2	MD	3,2	0,8
	MDR		0,2
3	Dodawania zmiennoprzecinkowe:	7,3	—
3.1	AE	1,5	0,8
	AER		0,2
3.2	AD	5,8	0,8
	ADR		0,2
4	Pobranie argumentu (64-bitowego):	7,2	—
	LD		0,8
	LDR		0,2
5	Pamiętanie argumentu 64-bitowego	7,2	—
	STD		1,0
6	Przesunięcie logiczne (o 8 pozycji)	4,6	—
	SIL		1,0
7	Dodawania stałoprzecinkowe:	35,1	—
	A		0,6
	AR		0,4
8	Dzielenia stałoprzecinkowe:	0,2	—
	D		0,6
	DR		0,4
9	Mnożenia stałoprzecinkowe:	0,6	—
	M		0,6
	MR		0,4
10	Pamiętanie argumentu 32-bajtowego:	8,2	—
	ST		1,0
11	Skoki bezwarunkowe:	17,5	—
	BC		0,5
	BCR		0,5
12	Skoki warunkowe efektywne:	2,6	—
	BC		0,5
	BCR		0,5
13	Skoki warunkowe nieefektywne:	3,9	—
	BC		0,5
	BCR		0,5

Iloraz otrzymany przez podzielenie sumy wag przez sumę iloczynów postaci $(t_i^{rx} p_i^{rx} + t_i^{rr} p_i^{rr}) \cdot w_i$ określa średnią szybkość procesora według metody GIBSON I, gdzie ilość składników sumowania równa się 16 (gdyż przyjęto 16 różnych instrukcji), tzn. $i = 1,2,\ldots,16$. Wielkości t_i oraz p_i oznaczają:

t_i^{rx} — czas wykonania i-tej instrukcji formatu RX,

t_i^{rr} — czas wykonania i-tej instrukcji formatu RR,

p_i^{rx} — prawdopodobieństwo występowania formatu RX w i-tej instrukcji,

p_i^{rr} — prawdopodobieństwo występowania formatu RR w i-tej instrukcji,

w_i — waga i-tej instrukcji.

Literatura

1. *Dryzek H.:* Organizacja pamięci w systemach liczących. PWN, Warszawa 1970.
2. Encyklopedia — Przyroda i Technika (maszyny rachunkowe). „Wiedza Powszechna" 1967.
3. *Flores I.:* Organizacija Vyczislitelnych maszin. Moskwa 1972.
4. *Flores I.:* Arytmetyka maszyn cyfrowych. WNT, Warszawa 1970.
5. *Hellwig Z. (pod redakcją):* Automatyczne przetwarzanie informacji. PWE, Warszawa 1974.
6. IBM System 360. Podstawy działania. ZETO ZOWAR, Warszawa 1972.
7. *Jakacki W.:* System kontroli i diagnostyki EMC R-32. Biuletyn MERA-ELWRO, Wrocław 1973.
8. Jednolity System EMC, Środki Techniczne, Katalog Informacyjny i Katalog Propozycji.
9. *Kasierski B.:* Pierwsza krajowa maszyna cyfrowa JS-R-32. Rozwiązania konstrukcyjne. Prace OBR MERA-ELWRO nr 1/74, Wrocław 1974.
10. *Łarionow A.:* Jedinnaja Sistiema EWM. „Statystyka", Moskwa 1974.
11. *Turski W.:* Podstawy użytkowania maszyn cyfrowych w ośrodkach naukowo-technicznych. PWN, Warszawa 1973.
12. *Turski W.:* Propedeutyka informatyki. PWN, Warszawa 1975.
13. *Zapendowski W.:* Środki techniczne mechanizacji i automatyzacji przetwarzania danych. OBRI, Warszawa 1972 (materiały szkoleniowe).

Spis treści

CZĘŚĆ PIERWSZA

1. WSTĘP 3
   1.1. Przeznaczenie skryptu 3
   1.2. Ogólne wiadomości o EMC Jednolitego Systemu 3
2. PODSTAWOWE WIADOMOŚCI O MASZYNACH CYFROWYCH 6
   2.1. Wprowadzenie 6
   2.2. Maszyny cyfrowe a maszyny analogowe . 7
   2.3. Historia maszyn liczących w świecie . . 7
   2.4. Rozwój maszyn cyfrowych w Polsce . . 8
   2.5. Klasyfikacja techniczna środków liczących 9
   2.6. Arytmetyka maszyn cyfrowych 11
   2.7. Podstawowe zasady działania maszyny cyfrowej 22

CZĘŚĆ DRUGA

3. PODSTAWOWE CECHY ARCHITEKTURY LOGICZNEJ 29
   3.1. Podstawowa struktura systemu 29
   3.2. Sterowanie wykonaniem programu . . 34
   3.3. Charakterystyka repertuaru instrukcji maszynowych 38
4. ORGANIZACJA KANAŁÓW WEJŚCIA-WYJŚCIA 45
   4.1. Wprowadzenie 45
   4.2. Operacje wprowadzania - wyprowadzania informacji 47
   4.3. Zakończenie przesyłania danych . . . 51

CZĘŚĆ TRZECIA

5. CHARAKTERYSTYKA JEDNOSTEK CENTRALNYCH JS EMC 52
   5.1. Model JS-1010 52
   5.2. Model JS-1021 53
   5.3. Model JS-1020 53
   5.4. Model JS-1030 54
   5.5. Model JS-1040 54
   5.6. Model JS-1050 55
   5.7. Model JS-1032 56

CZĘŚĆ CZWARTA

6. PAMIĘĆ BUFOROWA 61
   6.1. Wprowadzenie 61
   6.2. Organizacja pamięci buforowej 62
   6.3. Algorytmy przydzielania bloków pamięci buforowej 63
7. PAMIĘĆ WIRTUALNA 65
   7.1. Wprowadzenie 65
   7.2. Organizacja pamięci wirtualnej EMC JS . 73
8. URZĄDZENIA ZEWNĘTRZNE 79
   8.1. Pamięci zewnętrzne o dostępie bezpośrednim 79
   8.2. Pamięci taśmowe magnetyczne 83
   8.3. Czytniki i dziurkarki kart 85
   8.4. Drukarki wierszowe 85
   8.5. Czytniki i dziurkarki taśmy 86
   8.6. Moduły bezpośredniej komunikacji operatora z maszyną 87
   8.7. Urządzenia wprowadzania i wyprowadzania informacji w postaci graficznej . . 87
   8.8. Urządzenia teleprzetwarzania danych . . 90
9. SYSTEMY WIELOPROCESOROWE I WIELOMASZYNOWE 92
   9.1. Wprowadzenie 92
   9.2. Przykłady struktur wieloprocesowych . . 94
   9.3. Wieloprocesorowość w Jednolitym Systemie EMC 100
10. ZGODNOŚĆ PROGRAMOWA I EMULACJA EMC ODRA 1300 102
   10.1. Wprowadzenie 102
   10.2. Metody przenoszenia oprogramowania oraz trudności i efektywność przenoszenia 102
   10.3. Ogólna koncepcja wykonania programu EMC serii ODRA 1300 w polskich EMC JS 104
   10.4. Ogólna charakterystyka struktur rozkazów i danych EMC serii ODRA 1300 . . 105
   10.5. Rozmieszczenie informacji EMC serii ODRA 1300 w EMC JS 107

DODATEK A. Lista instrukcji JS EMC RIAD-1 . 109

DODATEK B. Metoda obliczania wydajności procesorów JS EMC 112

Literatura 113