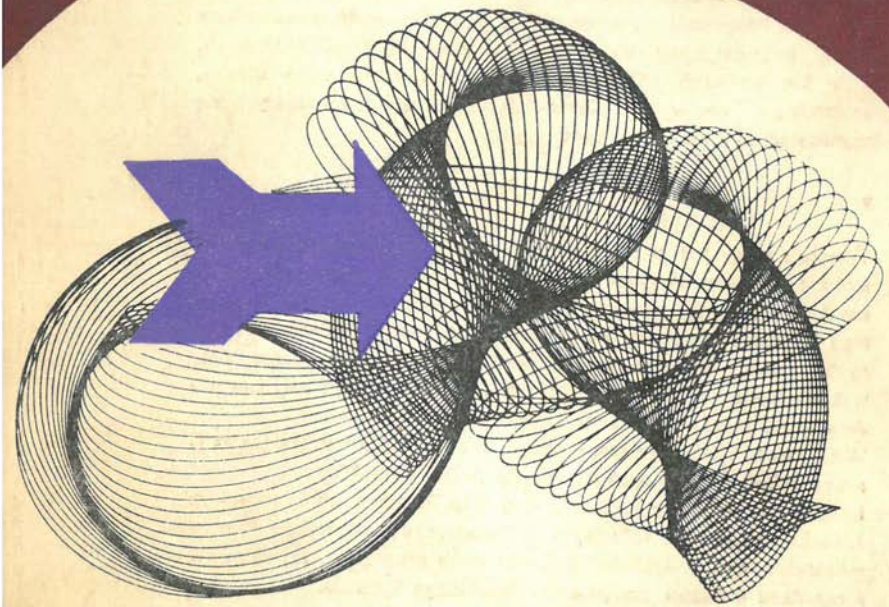
INFORMATYKA
W PRAKTYCE



Romuald Jagielski

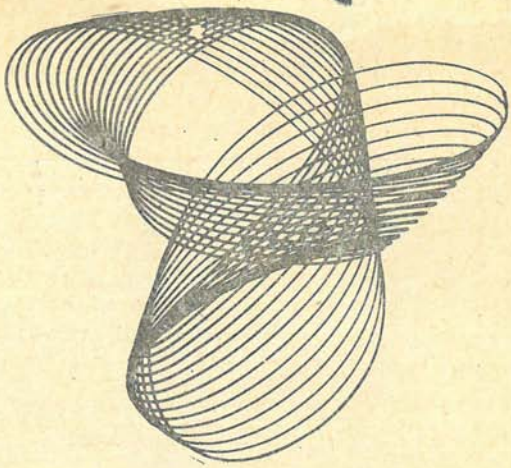
Komputery Jednolitego Systemu

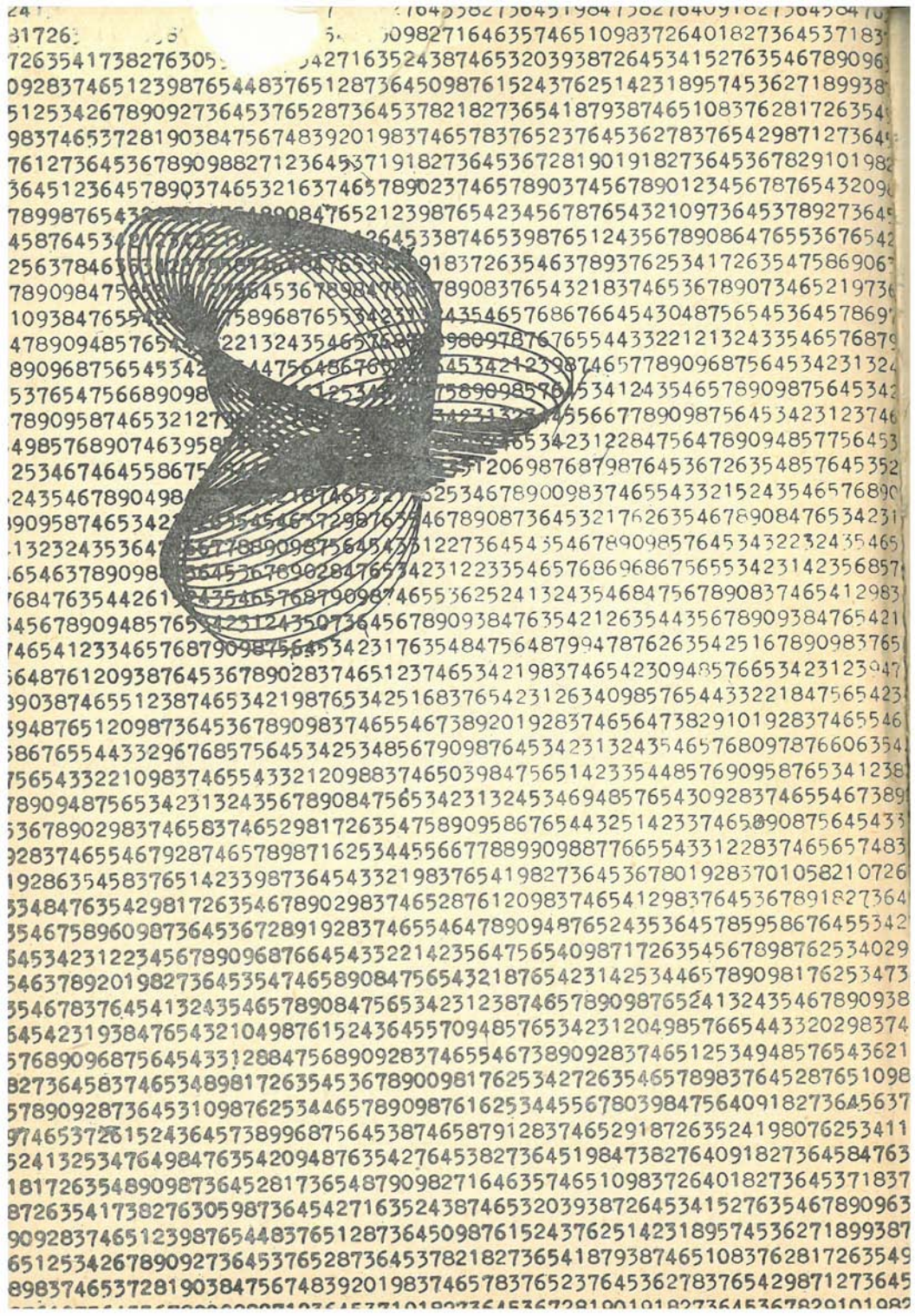
PWE

Seria wydawnicza „Informatyka w praktyce" zaspokaja potrzeby poznania literatury fachowej, poświęconej popularyzowaniu i wykorzystywaniu rozwiązań i zastosowań informatyki, a zwłaszcza budowania i funkcjonowania systemów informatycznych w jednostkach gospodarczych oraz doskonalenia sposobów uzyskiwania informacji w zarządzaniu. Poszczególne pozycje serii są przeznaczone dla pracowników służby informatycznej we wszystkich jednostkach gospodarczych, kadr kierowniczych tych jednostek, studentów wyższych uczelni, uczestników kursów podyplomowych oraz osób interesujących się zagadnieniami i rozwojem informatyki.

*

Tematem pracy jest przedstawienie ogólnych zasad działania komputerów Jednolitego Systemu wraz z charakterystyką sprzętu. Czytelnik uzyskuje informacje dotyczące struktury maszyn cyfrowych Jednolitego Systemu, organizacji wejścia-wyjścia danych, urządzeń zewnętrznych, metod działania Dyskowego Systemu Operacyjnego oraz wykorzystania języków ASSEMBLER i PL/I. W dalszej części są podane m.in. zasady organizacji i przetwarzania kartotek.
W aneksie zamieszczono listę rozkazów maszyn cyfrowych (załącznik I) oraz listę kodów używanych w tym systemie (załącznik II).
Książka jest przeznaczona przede wszystkim dla użytkowników maszyn Jednolitego Systemu, informatyków, studentów, uczestników kursów szkoleniowych i podyplomowych oraz osób pragnących zapoznać się z zasadami działania komputerów Jednolitego Systemu.





INFORMATYKA
W PRAKTYCE

Komputery Jednolitego Systemu

Romuald Jagielski

Państwowe Wydawnictwo Ekonomiczne
Warszawa 1980

Okładkę projektował
FRANCISZEK WINIARSKI

Redaktor
WANDA ROWICKA

Spis treści

Od Autora

Ogromny rozwój nauki, techniki i nowych metod zarządzania, jaki obserwujemy w ostatnich latach, spowodował wzrost zapotrzebowania na środki automatycznego przetwarzania informacji. Mniej więcej co pięć lat pojawia się nowa generacja komputerów, coraz to doskonalszych i wydajniejszych. W pierwszej połowie lat sześćdziesiątych pojawiła się trzecia generacja maszyn cyfrowych, zapoczątkowana przez firmę IBM seryjną produkcją komputerów systemu 360. Również w krajach socjalistycznych były prowadzone prace w zakresie techniki obliczeniowej, chociaż z pewnym opóźnieniem w stosunku do przodujących w tej dziedzinie krajów. W tej sytuacji pożądane było kupienie środków technicznych i naukowych w celu wyprodukowania rodziny komputerów wysokiej klasy.

W III kwartale 1968 r. Bułgaria, Czechosłowacja, NRD, Polska, Węgry i ZSRR utworzyły Międzyrządową Komisję Współpracy Krajów Socjalistycznych w zakresie Techniki Obliczeniowej, której zadaniem jest koordynacja prac związanych z produkcją rodziny komputerów i nazwaną Jednolitym Systemem Elektronicznych Maszyn Cyfrowych. Rezultatem 43 posiedzenia Komitetu Wykonawczego RWPG było podpisanie w grudniu 1969 r. „Międzynarodowego porozumienia o współpracy w zakresie opracowania, produkcji i zastosowań środków techniki obliczeniowej"; później do współpracy przyłączyła się Kuba i Rumunia.

Od tego czasu do dnia dzisiejszego opracowano konstrukcję i uruchomiono produkcję kilku typów maszyn cyfrowych i około kilkuset rodzajów urządzeń zewnętrznych. Wszystkie urządzenia odpowiadają normom przyjętym przez współpracujące kraje, co przyczynia się do zachowania zgodności i wymienności urządzeń technicznych i oprogramowania. Produkowane modele, oznaczone symbolami JS-1010 (WRL), JS-1020 (LRB i ZSRR), JS-1021 (CSRR), JS-1022, JS-1030 (ZSRR), JS-1032 (PRL), JS-1040 (NRD), JS-1050, JS-1060 (ZSRR) lub popularnie R-10, R-20, R-21, R-22, R-30, R-32, R-40, R-50, R-60 (R — pierwsza litera słowa *Riad* — szereg), to maszyny uniwersalne, o mocy obliczeniowej od kilku tysięcy

do kilkuset tysięcy operacji na sekundę. Są one przystosowane do rozwiązywania różnorodnych zadań o charakterze naukowo-technicznym i ekonomicznym, umożliwiają sterowanie procesami produkcyjnymi, jak również zarządzaniem gospodarką narodową.

Komputery Jednolitego Systemu charakteryzuje kompatybilność, obejmująca nie tylko urządzenia techniczne, lecz również programy (z pewnymi ograniczeniami w wypadku R-10 i R-21). Program napisany w jednym z języków używanych w Jednolitym Systemie (ASSEMBLER, PL/1, COBOL, FORTRAN, RPG, ALGOL) uruchomiony na maszynie mniejszej może również być odtworzony poprawnie na maszynie większej. Przy zachowaniu pewnych warunków wymienność programowa może zachodzić również w przeciwnym kierunku.

Każdy model systemu ma określoną minimalną konfigurację, która może być zwiększana w dowolny sposób. Rozszerzanie konfiguracji odbywa się przez zwiększenie zestawu urządzeń zewnętrznych, rozszerzenie pojemności pamięci lub przez tworzenie konfiguracji wielomaszynowych.

Tematem pierwszych czterech rozdziałów są ogólne zasady działania komputerów Jednolitego Systemu oraz charakterystyka produkowanego sprzętu. Następne rozdziały dotyczą wyłącznie oprogramowania i opracowane są pod kątem realizacji programów z Dyskowym Systemem Operacyjnym. Udzielenie tak wielkiej uwagi temu systemowi bierze się stąd, że jest on znacznie prostszy od Systemu Operacyjnego OS JS, a więc łatwiejszy dla pierwszego zapoznania się z tego rodzaju systemami*.

Podane opisy języków ASSEMBLER i PL/1, chociaż skrócone, zawierają dostateczną ilość informacji i przykładów, aby Czytelnik mógł pisać proste programy.

Książka jest w zasadzie adresowana do użytkowników maszyn cyfrowych Jednolitego Systemu, informatyków pragnących uzyskać podstawowe wiadomości o tego typu komputerach oraz studentów. Jest rzeczą wręcz niemożliwą zawarcie w jednej niewielkiej książce wyczerpujących informacji o zasadach działania systemu, sprzęcie, językach programowania i systemach operacyjnych. Trudność tę starałem się przezwyciężyć włączając do książki tylko niezbędne informacje, lecz tak dobrane, aby Czytelnik mógł posługiwać się w swojej pracy komputerami Jednolitego Systemu.

* W toku przygotowywania pracy o Jednolitym Systemie EMC stosowano głównie system DOS. Obecnie jednak używa się przede wszystkim systemu OS.

I. Architektura maszyn cyfrowych Jednolitego Systemu

Od kilku lat prowadzi się w krajach RWPG intensywne prace mające na celu stworzenie uniwersalnych maszyn cyfrowych, spełniających wymagania większości potencjalnych użytkowników. W rezultacie powstała rodzina maszyn cyfrowych składająca się z komputerów o zróżnicowanej mocy obliczeniowej i cenie, charakteryzujących się jednolitą architekturą.

Pojęcie architektury rodziny maszyn cyfrowych obejmuje istotę systemu, jego budowę logiczną i współdziałanie środków technicznych i programowych. Rozważając architekturę maszyn cyfrowych Jednolitego Systemu będziemy w mniejszym stopniu interesować się konkretnymi funkcjami i parametrami poszczególnych urządzeń, a uwagę naszą skupimy przede wszystkim na tym, co łączy wszystkie modele rodziny, a więc formaty danych i instrukcji, lista rozkazów, kody itp. Różne modele rodziny RIAD pod względem realizacji technicznej mogą bardzo od siebie odbiegać, lecz zachowują przy tym jednakową architekturę.

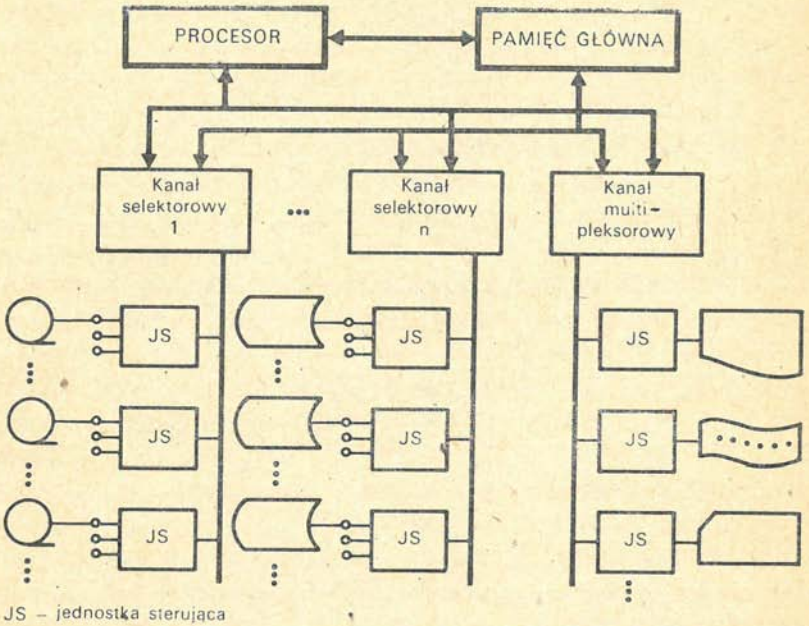
1. Struktura systemu

Komputer składa się z szeregu skomplikowanych urządzeń, spełniających rozmaite funkcje. Często też nazywa się maszynę cyfrową systemem liczącym lub też po prostu systemem.

W skład każdej maszyny cyfrowej Jednolitego Systemu wchodzi:

— pamięć główna,
— procesor,
— kanały: selektorowe i multipleksorowe,
— urządzenia zewnętrzne,
— jednostki sterujące urządzeniami zewnętrznymi.

Strukturę logiczną maszyn JS[1] przedstawia rysunek 1. Pamięć główna (operacyjna) przechowuje przetwarzaną informację. Do procesora należy wykonanie rozkazów pobieranych w pamięci głównej oraz sterowanie pracą urządzeń zewnętrznych. Procesor i pamięć operacyjna noszą ogólną nazwę jednostki centralnej. Do wyprowadzania i wprowadzania danych



Rys. 1. Struktura logiczna maszyn Jednolitego Systemu

z urządzeń zewnętrznych do pamięci lub w kierunku przeciwnym służą kanały. Urządzenia zewnętrzne podłączone są przez standardowe układy sprzęgające (ang. *Standard interface*) do kanałów. Dzięki standardowym układom sprzęgającym możliwe jest wyposażenie wszystkich modeli w urządzenia zewnętrzne tego samego typu.

[1] Por. *IBM System 360. Principles of Operations. IBM Systems Reference Library.* IBM Comparation Poughkeepsie, New York; T. Kamburelis, *Architektura logiczna EMC JS.* Problemy informatyki, Ośrodek Badawczo-Rozwojowy Informatyki, Warszawa 1976.

a. Pamięć główna

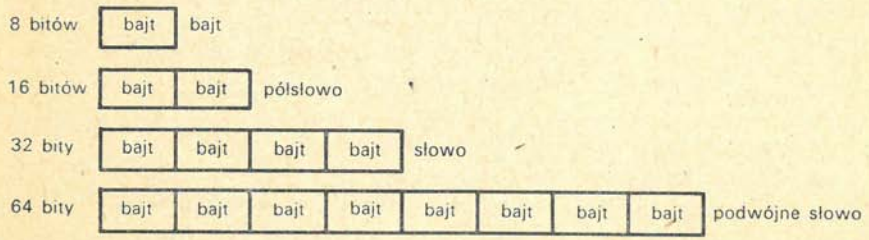
Pamięć główna z punktu widzenia konstrukcji składa się z bloków, które mogą być samodzielnym urządzeniem lub tworzyć jedną całość z procesorem. W obecnie produkowanych modelach pamięć główna jest pamięcią ferrytową. Procesor i kanały mają niezależny dostęp do pamięci.

Podstawową jednostką informacji w komputerach Jednolitego Systemu jest *bajt* (ang. *byte*). Bajt składa się z ośmiu bitów. Większe jednostki informacji są zawsze całkowitą wielokrotnością bajtu. Z każdym bajtem przesyłany jest dodatkowo dziewiąty bit kontrolny (bit parzystości). Wartość tego bitu nie może być programowo zmieniona. Pojemność pamięci podaje się w bajtach lub kilobajtach (1 Kbajt = 1024 bajty); waha się ona w granicach od 64 Kbajtów w małych maszynach do 1024 Kbajtów w dużych.

Grupę kolejnych bajtów pamięci nazywamy *polem*. Szczególne znaczenie mają pola pamięci o długości 2,4 i 8 bajtów, które noszą nazwy:

— półsłowo (2 bajty = 16 bitów),
— słowo (4 bajty = 32 bity),
— podwójne słowo (8 bajtów = 64 bity).

Oprócz tego w systemie używa się pól o zmiennej długości. Rysunek 2 przedstawia jednostki danych w pamięci.



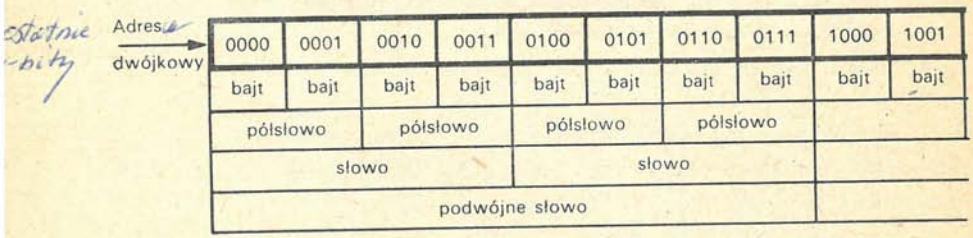
Rys. 2. Podstawowe jednostki informacji

Długość pola może być określona niejawnie w kodzie rozkazu (może ona w takim wypadku wynosić jeden, dwa, cztery lub osiem bajtów), lub w sposób jawny (mamy wtedy do czynienia z argumentami zmiennej długości).

Wszystkie bajty pamięci są ponumerowane kolejnymi liczbami całkowitymi poczynając od zera. Numery te są adresami pamięci. Adresem dowolnego pola jest adres pierwszego z lewej strony bajtu pola. Ponieważ w sy-

stemie używa się dla adresowania pamięci pola 24-bitowego, największy adres może wynosić 16 777 216.

Pola o stałej długości, takie jak półsłowo, słowo i podwójne słowo muszą być tak adresowane, aby leżały w określonych przedziałach pamięci formatu. Tak więc półsłowo musi mieć adres podzielony przez dwa, słowo — przez cztery, podwójne słowo przez osiem. Jeśli adresy zapisać jako liczby binarne, wówczas warunek ten sprowadza się do tego, aby adres półsłowa był liczbą kończącą się jednym zerem, adres słowa — dwoma zerami, a adres słowa podwójnego — trzema zerami (por. rys. 3).



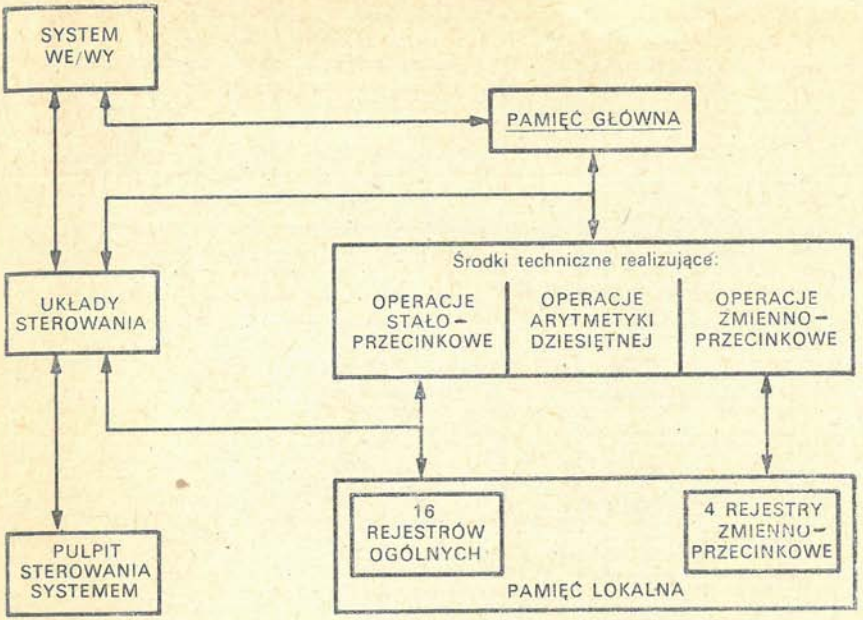
Rys. 3. Adresacja pamięci

Procedurę ustawiania danych w pamięci w żądany sposób nazywa się potocznie wyrównywaniem granic. Wyrównywanie granic ma na celu zwiększenie szybkości wykonywania operacji, ale utrudnia programowanie.

b. Procesor

Procesor komputerów Jednolitego Systemu zawiera układy elektroniczne służące do adresowania pamięci głównej, realizacji operacji arytmetycznych i logicznych w określonej kolejności, inicjowania komunikacji między pamięcią a urządzeniami zewnętrznymi oraz do automatycznej kontroli poprawności działania. Na rysunku 4 przedstawiono strukturę logiczną jednostki centralnej.

Procesory komputerów Jednolitego Systemu (z wyjątkiem R-10) zawierają 16 rejestrów uniwersalnych (ogólnych) i 4 rejestry zmiennoprzecinkowe. Fizycznie mogą to być niezależne układy elektroniczne lub część pamięci operacyjnej (w małych modelach). Rejestry uniwersalne mają długość jednego słowa (32 bity). Ponumerowane są one od 0 do 15, a numery te są jednocześnie adresami rejestrów. Dostęp do każdego z rejestrów możliwy jest dzięki czterobitowemu polu, które zawiera w rozkazach adres rejestru.



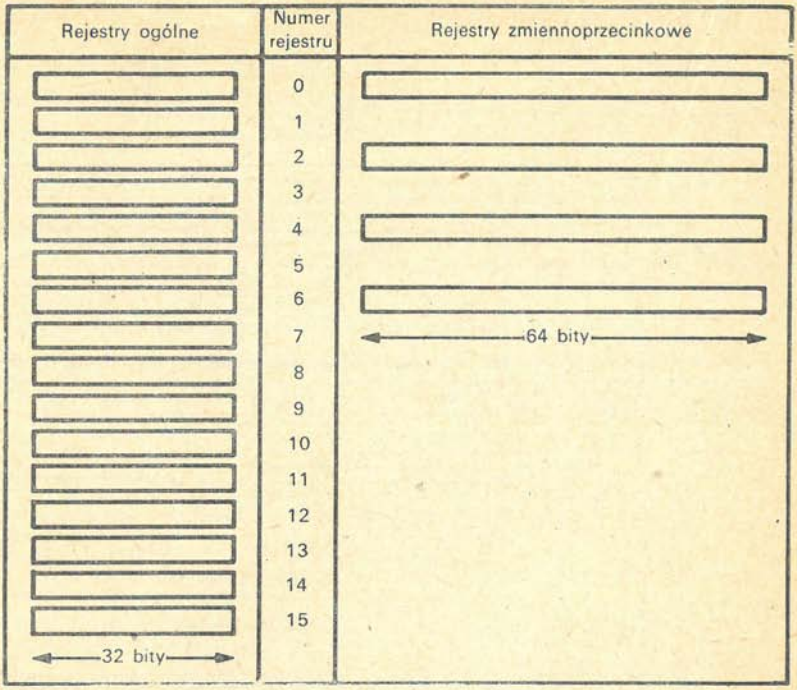
Rys. 4. Struktura logiczna centralnej jednostki przetwarzania

W skrócie będziemy oznaczać rejestry literą *R* z odpowiednim numerem. Rejestry uniwersalne są używane do przechowywania danych w toku wykonywania operacji stałoprzecinkowych, a także w operacjach na adresach i podczas indeksowania (modyfikacji) adresów. W pewnych sytuacjach, np. w operacjach mnożenia i dzielenia, dwa sąsiednie rejestry traktowane są jako jeden o długości 64 bitów. Adresowany rejestr zawiera wtedy bardziej znaczące pozycje liczby i musi mieć parzysty adres, natomiast dodatkowy rejestr zawiera mniej znaczące pozycje i ma adres nieparzysty.

Rejestry zmiennoprzecinkowe mają numery 0, 2, 4, 6. Każdy z nich ma długość 64 bity, ale może zawierać krótki (jedno słowo) lub długi (dwa słowa) argument zmiennoprzecinkowy. W wypadku używania krótkich argumentów druga połowa rejestru jest ignorowana i nie ulega zmianie.

Na rysunku 5 przedstawiono rejestry stało- i zmiennoprzecinkowe.

Jednostka centralna może przetwarzać liczby binarne stałej długości, liczby dziesiętne zmiennej długości oraz dane alfanumeryczne zmiennej długości. Arytmetyczne i logiczne operacje wykonywane w jednostce centralnej można podzielić na cztery klasy operacji, a mianowicie: operacje



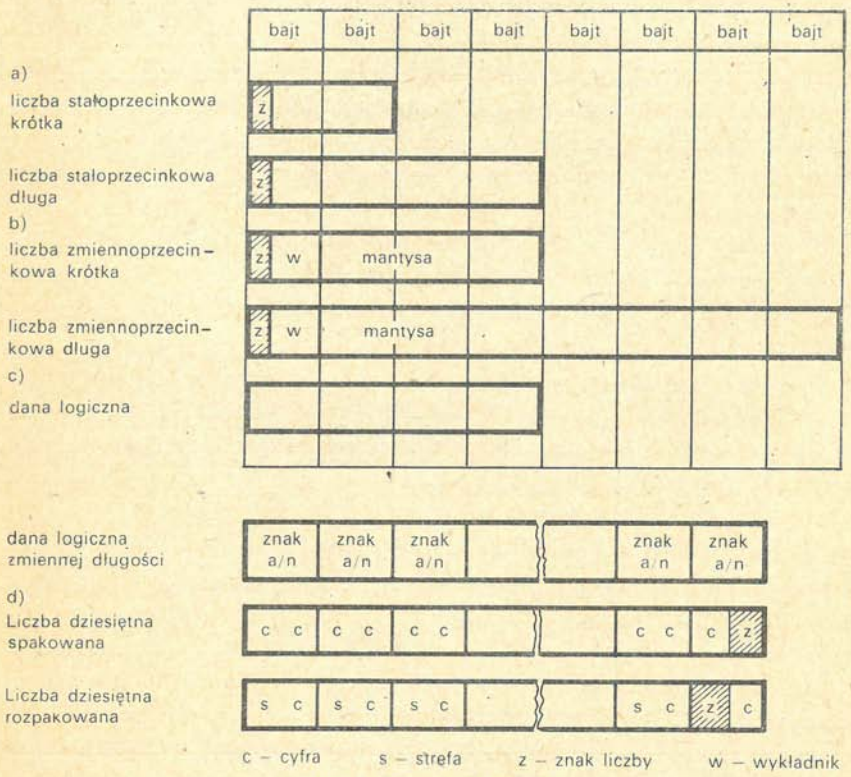
Rys. 5. Pamięć lokalna : rejestry ogólne i zmiennoprzecinkowe

stałoprzecinkowe, operacje zmiennoprzecinkowe, operacje logiczne oraz operacje dziesiętne.

Operacje stałoprzecinkowe. Podstawowym formatem argumentów w operacjach arytmetycznych jest czterobajtowe słowo ze stałym przecinkiem (por. rys. 6a). Liczba stałoprzecinkowa zapisywana jest w rejestrze jako 31 bitowa całkowita liczba dwójkowa (można przyjąć umownie, że przecinek znajduje się poza rejestrem, z prawej jego strony) ze znakiem. Znak zakodowany jest w lewej skrajnej pozycji rejestru: „0" oznacza plus, „1" oznacza minus. Liczby ujemne przedstawione są w kodzie uzupełniającym (jako tzw. uzupełnienie do 2). Oprócz tego można wykonywać operacje stałoprzecinkowe na krótkich liczbach (półsłowach), dzięki czemu można niejednokrotnie zwiększyć szybkość przetwarzania i lepiej wykorzystać pamięć operacyjną.

W operacjach dodawania, odejmowania, mnożenia i dzielenia jeden z argumentów znajduje się w rejestrze, a drugi argument — również w rejestrze — lub w pamięci głównej.

Operacje zmiennoprzecinkowe. Liczby zmiennoprzecinkowe mogą być przedstawione w dwóch formatach jako krótkie lub jako długie (rys. 6b). Formaty te różnią się tylko długością mantysy. Mantysa zawiera 24 lub 56 bitów. Przyjęto, że przecinek znajduje się z lewej strony od najbardziej znaczącej cyfry mantysy. Dla otrzymania prawdziwej wartości liczby należy mantysę pomnożyć przez liczbę 16 podniesioną do odpowiedniej potęgi, którą określa wykładnik (bity 1—7). Wykładnik zapisany jest w rejestrze w postaci liczby równej sumie wykładnika rzeczywistego i liczby 64. W ten sposób można zapisać liczby w zakresie od 10^{-78} do 10^{75}.



Rys. 6. Formaty danych

W operacjach zmiennoprzecinkowych jeden z argumentów biorących udział w operacji znajduje się w rejestrze, natomiast drugi — w innym rejestrze zmiennoprzecinkowym — lub w pamięci głównej.

Operacje logiczne. Operacje logiczne mogą być wykonywane na liczbach stałej długości (słowo), jak i na polach zmiennej długości. W pierwszym wypadku korzysta się z rejestrów uniwersalnych. Drugi sposób realizowania operacji logicznych polega na użyciu pól zmiennej długości; oba argumenty zapisane są w pamięci głównej, przy czym nie mogą to być pola dłuższe od 256 bajtów. Operacje wykonywane są bajt po bajcie, poczynając od lewej strony (por. rys. 6c).

Arytmetyka dziesiętna. W wypadku gdy w zadaniu występuje niewiele obliczeń (np. w systemach przetwarzania danych), a częste konwersje z dziesiętnego systemu liczenia na dwójkowy i odwrotnie są nieuzasadnione, wygodne jest stosowanie rozkazów arytmetyki dziesiętnej. W operacjach tego typu oba argumenty znajdują się w pamięci. Liczby mogą wystąpić w dwóch formatach (por.rys. 6d) — spakowanym i rozpakowanym. Cyfry dziesiętne od 0 do 9 zapisane są w postaci dziesiętnej kodowanej binarnie czterobitowymi kodami, odpowiednio od 0000 do 1001. Kombinacje bitów: 1010, 1100 i 1110 oznaczają znak plus, natomiast 1011 i 1101 oznaczają znak minus.

W postaci spakowanej w każdym bajcie znajdują się dwie cyfry dziesiętne; jedynie w ostatnim bajcie znajduje się jedna cyfra i znak liczby (por. rys. 6d). Format spakowany pozwala oszczędnie gospodarować pamięcią.

W sposób bardziej rozwlekły zapisuje się liczby w *postaci rozpakowanej*, zwanej też postacią strefową. W każdym bajcie znajduje się strefa (1111) i jedna cyfra. W prawym skrajnym bajcie zamiast strefy znajduje się znak liczby. Format rozpakowany liczb jest stosowany przede wszystkim podczas wprowadzania i wyprowadzania danych.

Liczby dziesiętne zajmują w pamięci pola zmiennej długości nie większe jednak od 16 bajtów. Istnieją specjalne instrukcje, zgodnie z którymi można zamienić liczby z postaci rozpakowanej na spakowaną i odwrotnie.

Ponieważ posługiwanie się liczbami dwójkowymi jest bardzo uciążliwe, stosuje się krótszy zapis w postaci szesnastkowych odpowiedników każdej czwórki bitów. Cyfry szestnastkowe od 0 do 9 oznacza się tak samo jak dziesiętne, natomiast literami A, B, C, D, D, E, F przyjęto oznaczać cyfry o wartościach dziesiętnych 10, 11, 12, 13, 14, 15 (por. tabl. 1). Tak więc, przykładowo, jeżeli bajt jest liczbą dwójkową 01001100, to znacznie wygodniej jest zapisać ją w postaci 4C. Inny przykład: liczba dziesiętna 425 równa jest liczbie szesnastkowej 1A9, od której łatwo można przejść do dwójkowej 000110101001.

TABLICA 1

Cyfry szestnastkowe

Liczba dwójkowa	Cyfra szestnastkowa	Liczba dwójkowa	Cyfra szestnastkowa
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Niekiedy, w celu uniknięcia nieporozumień, liczby różne od liczb dziesiętnych będziemy zapisywać z indeksem równym podstawie systemu liczenia, np. $11_{16} = 17 = 10001_2$.

2. Kody

We wszystkich maszynach cyfrowych Jednolitego Systemu kodem wewnętrznym jest kod EBCDIC (*ang. Extended Binary Coded Decimal Interchange Code*) lub ASCII (ang. *American Standard Code for Information Interchange*). Rodzaj kodu ustala się programowo. Najczęściej jednak stosuje się kod EBCDIC (por. rys. 7). Na rysunku 8 przedstawiono kod ASCII, który odpowiada kodowi ISO dla taśmy siedmiościeżkowej.

Kod EBCDIC rozszerzony o znaki alfabetu rosyjskiego nosi nazwę DKOI (ros. *dwoicznyj kod dla obmiena informacyjej*), a kod ASCII w podobnym wariancie otrzymał nazwę KOI-8 (ros. *bitowyj kod dla obmiena informacyjej*).

3. Format rozkazów maszynowych

Program jest zbiorem rozkazów kolejno wykonywanych przez maszynę. Rozkazy te, jak dane, przechowywane są w pamięci głównej lub w rejestrach. Rozkaz zawiera kod operacji oraz adresy danych lub też same dane, czyli argumenty biorące udział w operacji. Kod operacji zajmuje zawsze dwie

Pozycje bitów → 0123 →

↓ 4567

4567	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	NUL				SP	&	—									0
0001							/		a	j			A	J		1
0010									b	k	s		B	K	S	2
0011									c	l	t		C	L	T	3
0100	PF	RES	BYP	PN					d	m	u		D	M	U	4
0101	HT	NL	LF	RS					e	n	v		E	N	V	5
0110	LC	BS	ETB	UC					f	o	w		F	O	W	6
0111	DEL	IL	ESC	EOT					g	p	x		G	P	X	7
1000									h	q	y		H	Q	Y	8
1001									i	r	z		I	R	Z	9
1010			SM		¢	!		:								
1011					.	$	,	#								
1100					<	*	%	@								
1101					(	)	—	'								
1110					+	;	>	=								
1111					|	¬	?	"								

PF Wyłączenie drukarki
HT Tabulacja pozioma
LC Małe litery
DLE Kasowanie znaku
RES Powtórne pamiętanie
NL Nowa linia
BS Cofacz
IL Nieistotny
BYP Ominięcie
LF Zmiana wiersza
ETB Koniec bloku
ESC Przedrostek
SM Ustawianie trybu
PN Włączenie dziurkarki
RS Stop czytnika
UC Duże litery
EOT Koniec przysyłania
SP Odstęp

Rys. 7 Kod EBCDIC *(Extended Binary Coded Decimal Interchange Code)*

Pozycje bitów 76x5 / 4321	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	NUL	DLE			SP	0					@	P			/	p
0001	SOH	DCI			!	1					A	Q			a	q
0010	STX	DC2			"	2					B	R			b	r
0011	ETX	DC3			#	3					C	S			c	s
0100	EOT	DC4			$	4					D	T			d	t
0101	ENQ	NAK			%	5					E	U			e	u
0110	ACK	SYN			&	6					F	V			f	v
0111	BEL	ETB			'	7					G	W			g	w
1000	BS	CAN			(	8					H	X			h	x
1001	HT	EM			)	9					I	Y			i	y
1010	LF	SUB			*	:					J	Z			j	z
1011	VT	ESC			+	;					K	[			k	{
1100	FF	FS			,	<					L	\			l	|
1101	CR	GS			—	=					M	]			m	}
1110	SO	RS				>					N	∨			n	~
1111	SI	US			/	?					O	—			o	DEL

NUL Bez informacji
SOH Początek nagłówka
STX Początek tekstu
ETX Koniec tekstu
EOT Koniec przesyłania
ENQ Zapytanie
ACK Potwierdzenie gotowości
BEL Dzwonek
BS Cofacz
HT Tabulacja pozioma
LF Zmiana wiersza
VT Tabulacja pionowa
FF Zmiana formularza
CR Powrót wózka
SO Rejestr górny
SI Rejestr dolny
DLE Zmiana znaczenia dołączonych danych
DC1 Sterowanie urządzenia zewnętrznego
DC2 Sterowanie urządzenia zewnętrznego
DC3 Sterowanie urządzenia zewnętrznego
DC4 Sterowanie urządzenia zewnętrznego
NAK Przekazanie informacji przeczącej
SYN Synchronizacja
ETB Koniec przysyłania bloku
CAN Anulowanie
EM Koniec nośnika pamięci
SUB Podstawienie
ESC Zmiana znaczenia znaku
FS Oddzielenie pliku
GS Oddzielenie grupy
RS Oddzielenie zapisu
US Udzielenie jednostkowej informacji
SP Odstęp (zwykle niedrukowany)
DEL Kasowanie znaku

Rys. 8 Kod ASCII (*American Standard Code for Information Interchange*)

cyfry szesnastkowe, czyli jeden bajt. Standardowa uniwersalna lista rozkazów maszyn cyfrowych Jednolitego Systemu liczy 143 rozkazy. Jest pięć podstawowych typów rozkazów, które symbolicznie oznacza się w następujący sposób:

RR — rejestr — rejestr (ang. *Register-to-Register*),
RX — rejestr — pamięć z indeksacją (ang. *Register and Indexed Storage*),
RS — rejestr — pamięć bez indeksacji (ang. *Register-and-Storage*),
SI — pamięć — argument bezpośredni (ang. *Storage-and-Immediate operand*),
SS — pamięć — pamięć (ang. *Storage-to-Storage*).

Długość rozkazu zaszyfrowana jest w kodzie operacji; pierwsze dwa bity kodu operacji określają tę długość w sposób następujący:

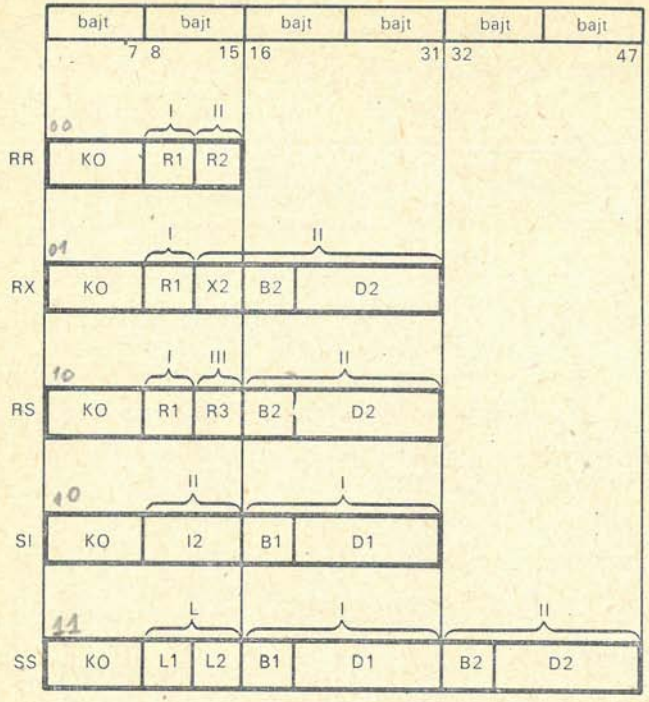
TABLICA

Numer pozycji 0—1	Długość rozkazu	Format
0 0	jedno półsłowo	RR
0 1	dwa półsłowa	RX
1 0	dwa półsłowa	RS lub SI
1 1	trzy półsłowa	SS

Na rysunku 9 przedstawiono wszystkie rodzaje formatów rozkazów. Obowiązują następujące oznaczenia:

R1, R2, R3 — czterobitowe pole, określające numer rejestru uniwersalnego (0 — 15) lub rejestru zmiennoprzecinkowego,
X2 — czterobitowe pole określające numer (0 — 15) rejestru uniwersalnego, zawierającego 24-bitowy indeks,
B1, B2 — czterobitowe pole określające numer rejestru ogólnego, zawierającego adres bazowy,
D1, D2 — 12-bitowy argument zwany przesunięciem,
I2 — 8-bitowy argument bezpośredni,
L1, L2 — czterobitowa długość argumentu, liczona w bajtach.

Cyfry 1, 2 lub 3 oznaczają, że dany element rozkazu odnosi się odpowiednio do pierwszego, drugiego lub trzeciego argumentu. W pewnych rozkazach czterobitowe pole drugiego bajtu lub cały drugi bajt jest ignorowany. Natomiast drugie półsłowo rozkazu ma zawsze jednakowy format.



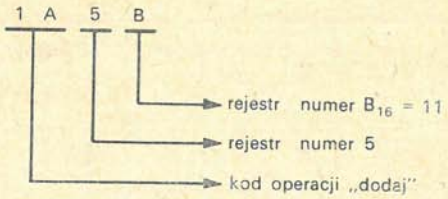
Rys. 9. Formaty rozkazów

a. Operacje typu RR

Rozkaz formatu RR składa się z trzech części: kodu operacji (8 bitów), pierwszego argumentu (czterobitowy numer rejestru), drugiego argumentu (czterobitowy numer rejestru). Rozkaz realizowany jest w ten sposób, że na zawartościach obu rejestrów wskazanych w rozkazie dokonuje się operacji określonej kodem operacji, a wynik tego działania zostaje wpisany do rejestru zawierającego uprzednio pierwszy argument. Oczywiście, początkowa wartość tego rejestru zostanie zniszczona.

PRZYKŁAD 1

Liczba 1A5B jest rozkazem zapisanym szesnastkowo. Poszczególne znaki mają następujące znaczenie:



W wyniku wykonania tego rozkazu do liczby zapisanej w rejestrze numer 5 zostanie dodana zawartość rejestru numer 11, a suma umieszczona zostanie w rejestrze numer 5. Symbolicznie operację tę zapiszemy w następujący sposób:

$$(R5)+(R11) \longrightarrow R5$$

Nawiasy oznaczają, że chodzi tu o zawartość rejestru, a nie o jego adres (numer). ■

b. Operacje typu RX

W operacjach typu RX pierwszy argument znajduje się w rejestrze, natomiast drugi jest słowem lub półsłowem w pamięci głównej. Po wykonaniu operacji, określonej kodem podanym w rozkazie, wynik zostaje zapisany w rejestrze zawierającym początkowo pierwszy argument. Adres argumentu w pamięci głównej określa suma:

$$E = D+(B) + (X),$$

gdzie:

D — przesunięcie,

(B) — zawartość rejestru bazowego,

(X) — zawartość rejestru indeksowego.

Przesunięcie D jest 12-bitową liczbą, której maksymalna wartość może wynosić 4096. B i X to dowolne numery rejestrów, przyjęte według uznania programisty. Rejestr indeksowy występuje wyłącznie w operacjach typu RX, natomiast rejestr bazowy występuje również w innych operacjach. Oba te rejestry umożliwiają adresowanie wewnątrz pamięci głównej, a ich jednoczesna obecność w operacjach typu RX umożliwia podwójne indeksowanie.

PRZYKŁAD 2

Przeanalizujmy rozkaz typu RX:

Adres drugiego argumentu oblicza się w tym wypadku jako sumę:

$$E2 = 005 + (R4) + (R9)$$

Tak więc rozkaz ten oznacza, że zawartość rejestru nr 3 ma być dodana do zawartości pola pamięci o adresie E2 i wynik umieszczony w rejestrze nr 3:

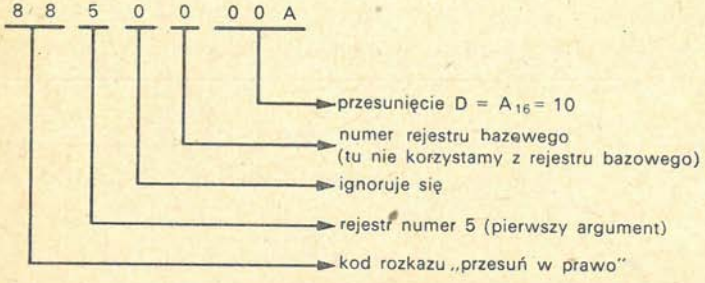
$$(R3) + (E2) \longrightarrow R3$$ ■

c. Operacje typu RS

W operacjach typu RS występują trzy argumenty. Pierwszy i trzeci mieści się w rejestrze, natomiast drugi w pamięci głównej. Formatu tego używa się w dwojaki sposób: w rozkazach przesunięcia trzeci argument jest pomijany, w pozostałych wypadkach spełnia różne zadania. Zilustrujemy to dwoma przykładami.

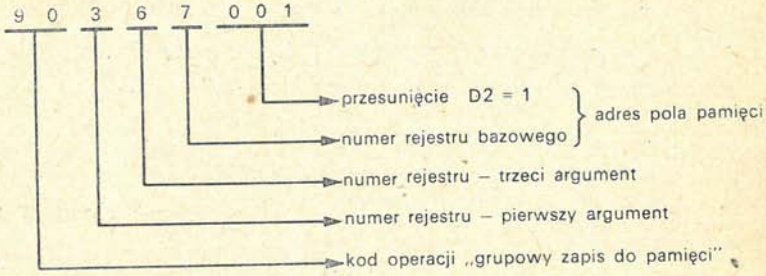
PRZYKŁAD 3

a) rozszyfrujemy rozkaz przesunięcia 8850000A:



W danym wypadku rozkaz zostanie zainterpretowany przez maszynę w następujący sposób: przesunąć zawartość rejestru 5 w prawo o ilość pozycji dwójkowych wskazaną przez ostatnie sześć bitów adresu drugiego argumentu (w rozkazach przesunięcia nie wykorzystuje się drugiego argumentu). Argument trzeci nie występuje. W rezultacie liczba mieszcząca się w rejestrze 5 zostanie przesunięta w prawo o 10 pozycji;

b) przykład zapamiętywania zawartości rejestrów uniwersalnych w pamięci głównej:



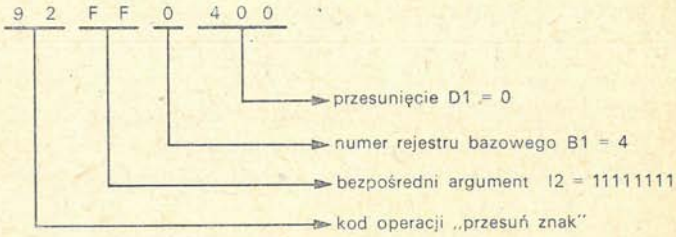
Rozkaz ten spowoduje zapamiętanie w pamięci głównej pod adresem określonym jako $(B2)+D2$ zawartości kolejnych rejestrów od 3 do 6.

d. Operacje typu SI

Operacje typu SI charakteryzują się tym, że pierwszy argument jest adresem pamięci w postaci $(B1)+D1$, a drugi argument jest daną, której wartość podana jest bezpośrednio w rozkazie. Przykładem tej operacji może być przesłanie określonego znaku pod wskazany adres pamięci.

PRZYKŁAD 4

Rozważmy przykład rozkazu, który powoduje przesłanie jednego bajtu do pamięci głównej. Wartość tego bajtu podana jest bezpośrednio w samym rozkazie.



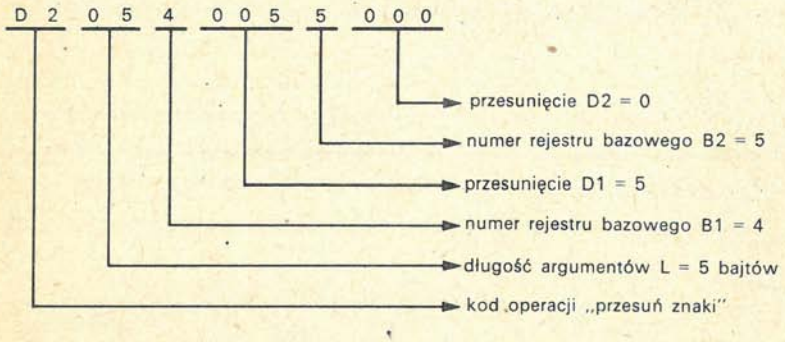
Rozkaz ten należy rozumieć następująco: przesłać bezpośredni argument I2, równy szesnastkowej liczbie FF, do pamięci według adresu wskazanego przez adres $E1$ ($E1 = (B1)+D1$). ■

e. Operacje typu SS

Bardzo często oba argumenty przechowywane są w pamięci operacyjnej i można wykonać na nich pewne działania nie korzystając z pomocy rejestrów uniwersalnych. Możliwość taką stwarzają rozkazy typu SS, które określają adresy obydwu argumentów i ich długości. Rozpatrzmy taki rozkaz na konkretnym przykładzie.

PRZYKŁAD 5

Rozkaz ten służy do przemieszczania danych w pamięci:



Sens tego rozkazu jest następujący: przesunąć zawartość pola w pamięci o długości $L = 5$ bajtów określonego adresem $E2 = (B2)+D2$ do pola o adresie $E1 = (B1)+D1$.

Pełną listę rozkazów maszynowych komputerów Jednolitego Systemu zawiera załącznik I.

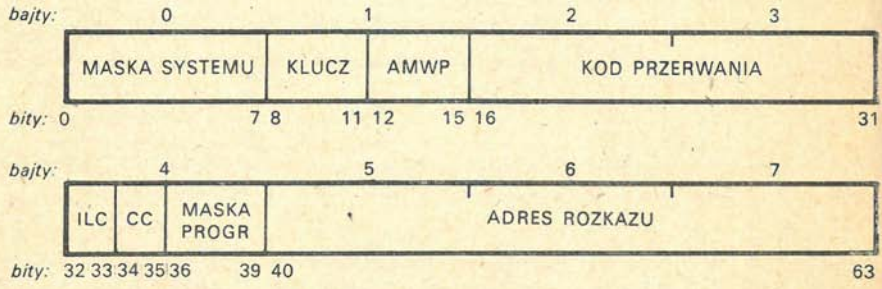
W przykładach tych podaliśmy rozkazy maszynowe, czyli rozkazy w takiej postaci, w jakiej występują one wewnątrz maszyny. Pisanie programów w taki właśnie sposób, chociaż możliwe, jest zbyt uciążliwe, aby mogło być stosowane. Dużym udogodnieniem w tej mierze jest język symboliczny — ASSEMBLER.

4. Realizacja programu

Rozkazy programu umieszczone są w kolejnych komórkach pamięci. Jeśli program jest wykonywany sekwencyjnie (nie występują operacje typu skok warunkowy lub skok bezwarunkowy), to wówczas po wykonaniu bieżącego rozkazu jako następny wykonywany jest rozkaz wskazywany przez licznik adresów, którego wartość zwiększa się o liczbę równą długości ostatnio zrealizowanego rozkazu.

Sekwencyjną kolejność wykonywania instrukcji zmieniają operacje skoków. Skok może być uwarunkowany spełnieniem lub niespełnieniem pewnej relacji w operacjach arytmetycznych lub logicznych. Możliwa jest również bezwarunkowa zmiana kolejności wykonywania instrukcji programu.

Informacje konieczne do właściwej realizacji programu zawiera *Słowo Stanu Programu* (PSW — ang. *Program Status Word*). Jest to podwójne słowo, którego zadaniem jest ciągłe rejestrowanie stanu systemu. PSW ma następujący format:



Poszczególne pola mają następujące znaczenie:

0— 7 Maska systemu. Dotyczy przerwań we/wy i przerwań zewnętrznych. O maskowaniu przerwań we/wy będziemy mówić w rozdziale II.

8—11 Klucz ochrony pamięci. Jeśli nie wykorzystuje się, pole to wypełnione jest zerami.

12 **A.** Bit ten ma wartość 1, gdy w systemie używa się kodu ASCII. Jeśli w tym miejscu zapisane jest 0, system pracuje w kodzie EBCDIC (por. rys. 7 i 8).

13 **M.** Bit zwykle równa się 1.M = 0 oznacza, że przekłamania od układów kontrolnych sprawności maszyny są ignorowane.

14 **W.** Gdy W = 1 oznacza to, że jednostka centralna znajduje się w stanie oczekiwania (rozkazy nie są wykonywane). Bit W = 0 oznacza stan BIEG, w którym rozkazy są wykonywane w zwykły sposób.

15 **P.** P = 1 oznacza, że jednostka centralna znajduje się w stanie PROBLEM. W stanie PROBLEM rozkazy we/wy, bezpośredniego sterowania. Ustaw maskę systemu itp. (tzw. uprzywilejowane rozkazy) są niedozwolone. Rozkazy te mogą być wykonywane tylko w stanie SUPERVISOR (P=0).

16-31 Kod przerwania programu. Wartość tego pola określa przyczynę przerwania programu (por. tabl. 3).

32-33 ILC. Długość w półsłowach ostatnio wykonywanego rozkazu:
01 dla rozkazów typu RR,
10 „ RX, RS i SI,
11 „ SS
00 oznacza, że system nie był w stanie określić długości ostatnio wykonywanego rozkazu.

34-35 Kod warunku. Może być 0, 1, 2 lub 3. Każda z tych wartości ma określone znaczenie dla pewnych rozkazów i jest wykorzystywana przez instrukcje skoków.

36-39 Maska programu. Bity maski programu określają, czy pewne sytuacje wyjątkowe pojawiające się w czasie wykonywania programu będą powodowały przerwanie (odpowiedni bit równy 1), czy też będą ignorowane (wartość maski równa 0). Dotyczy to czterech następujących sytuacji:

bit 36 — nadmiar stałoprzecinkowy (ang. *fixed-point overflow*)
bit 37 — nadmiar dziesiętny (ang. *decimal overflow*)
bit 38 — niedomiar wykładnika (ang. *exponent underflow*)
bit 39 — nieprawidłowa wartość mantysy (ang. *significance*)

40-63 Adres rozkazu. W polu tym znajduje się adres następnego do wykonania rozkazu.

Przy każdym przerwaniu pracy programu konieczne jest zapamiętanie aktualnej wartości PSW, gdyż informacja ta umożliwia późniejsze wznowienie przetwarzania tego programu.

5. System przerwań

Pewne zdarzenia mogą spowodować przerwanie wykonywania aktualnie przetwarzanego programu w maszynie. Okoliczności powodujące przerwanie mogą nosić charakter sytuacji awaryjnej, spowodowanej np. nieprawidłową pracą urządzeń elektronicznych, lub też mogą być świadomie wywołane celem zwiększenia efektywności pracy systemu.

System przerwań pozwala jednostce centralnej zmieniać swój stan przy pojawieniu się określonych warunków na zewnątrz systemu, w urządzeniach we/wy lub w samej jednostce centralnej. Przewidziano pięć klas przerwań:

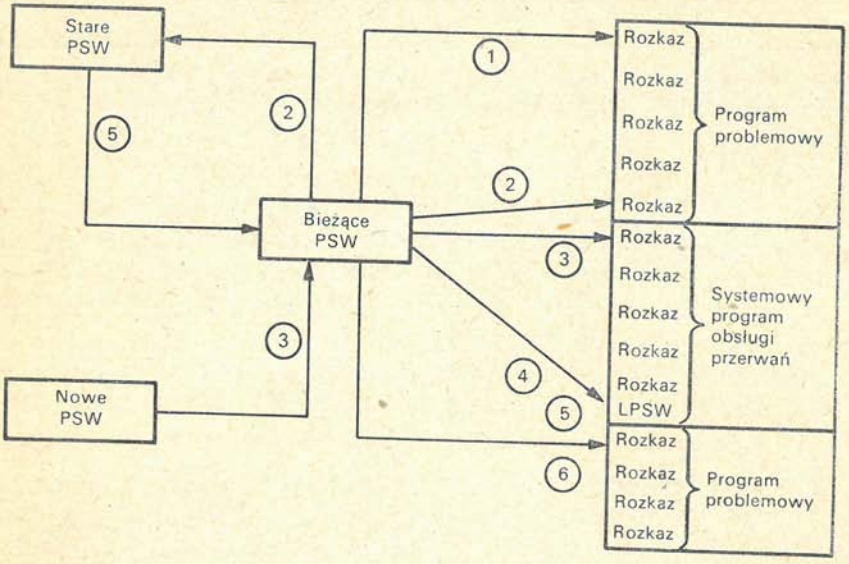
1) przerwania „błąd maszyny",
2) przerwania we/wy,
3) przerwania zewnętrzne,
4) przerwania programowe,
5) przerwania przywołania SUPERVISORA.

Wszystkie sytuacje powodujące przerwania danej klasy są obsługiwane przez pewien podprogram, który jest częścią SUPERVISORA. W ten sposób pięciu klasom przerwań odpowiada pięć programów SUPERVISORA, obsługujących te przerwania.

a. Mechanizm przerwań

Słowo Stanu Programu określające w danym momencie stan wykonywanego programu nazywa się „bieżącym" PSW. Każdej klasie przerwań odpowiadają dwa PSW, noszące nazwy „stare" i „nowe" PSW; przeznaczone są dla nich stałe pola w pamięci. Przerwanie powoduje zapamiętanie bieżącego PSW i umieszczenie go w pamięci głównej w polu przeznaczonym dla „starego" PSW. „Stare" PSW zawiera wszystkie niezbędne informacje o stanie systemu w momencie przerwania. Następnie system pobiera z odpowiedniego pola pamięci „nowe" PSW, które staje się bieżącym. Pozwala na wykonanie programu obsługi odpowiedniego dla danej klasy przerwań. Gdy program ten zostanie wykonany, wówczas ostatni jego rozkaz *Ładuj* PSW **(LPSW)** zamienia bieżące PSW na „stare" i system wraca do stanu, w jakim znajdował się przed przerwaniem. Na rysunku 10 przedstawiono przebieg procesu przerwania. Cyfry oznaczają poszczególne kroki.

„Stare" i „nowe" PSW dla danego typu przerwań zapamiętywane są zawsze w tych samych polach pamięci głównej. Adresy tych pól podane są w tablicy 2.



① wykonywane są rozkazy programu problemowego,
② w toku tego rozkazu nastąpiło przerwanie; po dokończeniu rozkazu bieżące PSW zostaje zapamiętane jako „stare",
③ z pamięci pobierane jest „nowe" PSW jako bieżące. Rozpoczyna się wykonywanie programu obsługi przerwań,
④ koniec pracy przerwań programu obsługi,
⑤ ostatni rozkaz programu obsługi, jest to **LPSW**, który umieszcza z powrotem „stare" PSW jako bieżące,
⑥ program użytkownika jest kontynuowany.

Rys. 10. Mechanizm przerwań

Żądania przerwań przyjmowane są tylko wtedy, jeśli jednostka centralna nie jest zamaskowana odnośnie do danego źródła przerwania. Maskowanie przerwań polega na ustawieniu 0 w odpowiedniej pozycji PSW (maska systemu, maska programu lub bit M). Za pomocą maski systemu można maskować przerwania od we/wy i przerwania zewnętrzne, przy czym zgłoszenia tych przerwań są przechowywane do czasu, dopóki jednostka centralna będzie w stanie je przyjąć. Za pomocą maski programu można

TABLICA 2
Adresy PSW w pamięci głównej

Typy przerwań	Adres pamięci	
	stare PSW	nowe PSW
Zewnętrzne	24	88
Przywołanie SUPERVISORA	32	96
Programowe	40	104
Od układów kontroli	48	112
Od we/wy	56	120

zamaskować cztery spośród 15 przerwań programowych, a używając bitu M można zamaskować przerwania od układów kontrolnych maszyny. Innych typów przerwań nie można zamaskować.

Żądanie przerwania spełniane jest zawsze po zakończeniu wykonania bieżącego rozkazu, przed wykonaniem następnego. Może się jednak zdarzyć, że zostanie zakłócone wykonywanie bieżącego rozkazu. Na podstawie zapamiętanych w PSW informacji można wtedy określić adres ostatniego wykonywanego przed przerwaniem rozkazu.

b. Klasy przerwań

Przerwania we/wy mają miejsce wtedy, gdy urządzenie zewnętrzne kończy operację lub gdy w urządzeniu zdarzyła się sytuacja wymagająca udziału jednostki centralnej. Przerwania tego typu umożliwiają reakcję jednostki centralnej na sygnały z kanałów i urządzeń zewnętrznych.

Przerwanie od we/wy może nastąpić tylko wtedy, gdy odpowiedni kanał nie jest zamaskowany (maskowanie powoduje bit 0 w masce systemu PSW). Adresy kanału i urządzenia we/wy, które wywołało przerwanie zapamiętuje się w starym PSW w pozycjach 21-31. Dodatkowa informacja o we/wy przechowywana jest w *Słowie Stanu Kanału* (CSW—ang. *Channel Status Word*), które zapamiętywane jest w czasie przerwania.

Przerwania programowe pojawiają się w wyniku błędów w programie takich, jak niedopuszczalny kod operacji, nieistniejący adres w pamięci, próba dzielenia przez zero itp. Przyczynę przerwania wskazuje kod przerwania programowego w „starym" PSW. Możliwe są następujące rodzaje przerwań:

1. Operacja niezidentyfikowana. Rozkaz zawiera nieistniejący kod operacji.

2. Operacja uprzywilejowana. Przerwanie wskutek pojawienia się kodu operacji uprzywilejowanej, np. operacji we/wy, gdy system znajduje się w stanie PROBLEM.

3. Operacja EXECUTE. Nieprawidłowo użyty rozkaz **EXECUTE** (*wykonaj instrukcję*).

4. Ochrona pamięci. Jeżeli klucz ochrony pamięci pola, do którego wpisuje się wynik nie jest zgodny z kluczem ochrony w PSW, następuje wówczas przerwanie programowe.

5. Adresacja. Błąd adresacji polega na użyciu adresu nie mieszczącego się w granicach istniejącej w systemie pamięci.

6. Specyfikacja. Przerwanie z powodu błędnej specyfikacji następuje, jeśli:
— adres danej lub instrukcji nie znajduje się na granicy wymaganej dla półsłowa, słowa lub podwójnego słowa,
— w rozkazach wymagających argumentów zajmujących parę rejestrów podany jest nieparzysty numer rejestru,
— w operacjach zmiennoprzecinkowych użyto rejestrów o numerach różnych od 0, 2, 4 lub 6.

7. Dane. Błędne dane powodują przerwanie w następujących wypadkach:
— nieprawidłowa postać liczby dziesiętnej,
— w operacji arytmetyki dziesiętnej nieprawidłowe nakładanie się pól argumentów,
— niedopuszczalnie długa mnożna.

8. Nadmiar stałoprzecinkowy. Nadmiar stałoprzecinkowy pojawia się wówczas, gdy następuje przeniesienie z najstarszej pozycji liczby do pozycji znakowej, lub gdy dana opuszcza granice rejestru.

9. Dzielenie stałoprzecinkowe. Przerwanie tego typu nastąpi w wypadku operacji dzielenia, jeśli całkowita część ilorazu nie mieści się w rejestrze, lub wystąpiło dzielenie przez zero. Odnosi się tu również nieprawidłowość konwersji dwójkowej, gdy wynik zajmuje więcej niż 31 bitów.

10. Nadmiar dziesiętny. Nadmiar dziesiętny występuje wtedy, gdy w operacji arytmetyki dziesiętnej wynik nie mieści się w wyznaczonym dla niego polu.

11. Dzielenie dziesiętne. Przerwanie wystąpi wtedy, gdy w dzieleniu dziesiętnym iloraz nie mieści się w polu dla niego przeznaczonym.

12. Nadmiar wykładnika. Jeżeli w operacjach zmiennoprzecinkowych wykładnik osiągnął wartość większą od 127, wówczas następuje przerwanie.

13. Niedomiar wykładnika. Przerwanie następuje wówczas, gdy w operacjach zmiennoprzecinkowych wykładnik okazał się mniejszy od zera.

14. Nieprawidłowa wartość. Przerwanie tego typu pojawia się w zmiennoprzecinkowych operacjach dodawania lub odejmowania w wypadku, gdy mantysa wyniku równa się zeru.

15. Dzielenie zmiennoprzecinkowe. Błąd w dzieleniu zmiennoprzecinkowym występuje wtedy, gdy ma miejsce próba dzielenia przez zero zmiennoprzecinkowe.

TABLICA 3

Kody przerwań programowych

Kod przerwania	Przyczyna przerwania
0000 0001	Operacja niezidentyfikowana
0000 0010	Operacja uprzywilejowana
0000 0011	Operacja EXECUTE
0000 0100	Ochrona pamięci
0000 0101	Adresacja
0000 0110	Specyfikacja
0000 0111	Dane
0000 1000	Nadmiar stałoprzecinkowy
0000 1001	Dzielenie stałoprzecinkowe
0000 1010	Nadmiar dziesiętny
0000 1011	Dzielenie dziesiętne
0000 1100	Nadmiar wykładnika
0000 1101	Niedomiar wykładnika
0000 1110	Nieprawidłowa wartość mantysy
0000 1111	Dzielenie zmiennoprzecinkowe

W tablicy 3 przedstawiono przyczyny przerwań programowych i ich kody w PSW.

Przerwanie z powodu przywołania SUPERVISORA. Przerwanie tego typu zachodzi w wyniku wykonania rozkazu Przywołaj SUPERVISOR (SVC — ang. *Supervisor Call*). Osiem bitów z pozycji 8—15 rozkazu **SVC** zostaje przeniesionych na miejsce kodu przerwania w „starym" **PSW**, co pozwala tym samym odróżniać przerwania w granicach danej klasy. Podstawowym celem rozkazu **SVC** jest przełączenie jednostki centralnej ze stanu PROBLEM w stan SUPERVISOR (stany jednostki centralnej będą opisane w dalszej części działu).

Przerwania zewnętrzne. Przerwania zewnętrzne pojawiają się w następujących wypadkach:

— gdy zegar maszyny osiągnął odpowiednią wartość,

— gdy operator nacisnął na pulpicie sterującym przycisk PRZERWANIE,

— w wyniku otrzymania sygnału zewnętrznego, np. od innej sprzężonej maszyny.

Przerwania zewnętrzne zachodzą tylko wtedy, gdy 7 bit maski systemu w PSW ustawiony jest w „1". Przyczynę przerwania można odczytać z 24-31 bitów PSW (bity 16-23 równe są zeru):

Pozycje PSW	Przyczyna przerwania
24	Zegar
25	Przycisk PRZERWANIE
26	Sygnał zewnętrzny 6
27	„ 5
28	„ 4
29	„ 3
30	„ 2
31	„ 1

Przerwania „błąd maszyny". Jeśli zostaje wykryte nieprawidłowe działanie maszyny następuje wstrzymanie wykonywania bieżącej operacji, o ile nie jest przerwanie zamaskowane (13 bit PSW). Towarzyszy temu uruchomienie odpowiednich programów diagnostycznych, po czym następuje procedura przerwania. W rezultacie otrzymuje się pełen opis stanu jednostki centralnej, zapamiętany w pamięci głównej od adresu 128. Informacja ta przyczynia się do zlokalizowania przyczyny niesprawności maszyny.

c. Priorytet przerwań

Podczas realizacji programu może wystąpić jednocześnie kilka żądań przerwań wywołanych różnymi przyczynami. Żądania przerwań są przyjmowane według ustalonej kolejności:

— przerwanie „błąd maszyny",

— przerwanie programowe lub przywołanie SUPERVISORA,

— przerwanie zewnętrzne,

— przerwanie od we/wy.

Przerwanie programowe i przerwanie z powodu przywołania SUPERVISORA wzajemnie się wykluczają.

Najwyższy priorytet ma przerwanie wywołane błędem maszyny. Przerwanie takie powoduje natychmiastowe wstrzymanie wykonywania bieżącej operacji. Jeżeli nie ma sygnałów przerwania z powodu błędu maszyny, przyjmowane są przerwania programowe lub przerwania z powodu przywołania SUPERVISORA. W następnej kolejności przyjmuje się przerwania zewnętrzne, a na końcu przerwania od we/wy. W takiej właśnie kolejności system zapamiętuje „stare" PSW w polach pamięci odpowiednich dla danej klasy przerwań, jednak nie wykonuje się programów obsługi wskazanych „nowym" PSW do czasu przyjęcia wszystkich zgłoszeń przerwań. Procesor zaczyna wykonywać rozkazy dopiero po przyjęciu ostatniego zgłoszenia, rozpoczynając od ostatnio zgłoszonego PSW. W ten sposób programy obsługi przerwań wykonywane są w kolejności odwrotnej do tej, w jakiej były przyjmowane zgłoszenia przerwań.

6. Stany jednostki centralnej

Stan jednostki centralnej jest określany czterema parami wzajemnie wykluczających się stanów:

PROBLEM — SUPERVISOR,
CZEKAJ — BIEG,
PRZERWANIE DOZWOLONE — PRZERWANIE ZABRONIONE,
STOP — PRACA.

Przejście od jednego stanu do drugiego dla każdej pary stanów może odbywać się niezależnie. Zmiana stanu nie wpływa na zawartość rejestrów i na wykonanie operacji we/wy. Może natomiast oddziaływać na pracę zegara.

Stan STOP - PRACA. Przełączenie jednostki centralnej ze stanu STOP do stanu PRACA może odbyć się tylko ręcznie. W stanie STOP rozkazy są wykonywane, przerwania są ignorowane, zegar nie pracuje. Na pulpicie operatorskim świeci się lampka STOP. W stanie PRACA jednostka centralna może wykonywać rozkazy i przyjmować przerwania.

Stan CZEKAJ - BIEG. W stanie BIEG rozkazy wykonywane są w zwykły sposób. W stanie CZEKAJ rozkazy nie są wykonywane, ale zegar pracuje. Dopuszczalne są przerwania we/wy i przerwania zewnętrzne, jeżeli nie są zamaskowane. Stan CZEKAJ występuje zwykle wtedy, gdy program oczekuje przerwania we/wy lub interwencji operatora. O stanie

BIEG lub CZEKAJ świadczy wartość 14 bitu PSW: jeśli bit ten równy jest jedynce, wówczas jednostka centralna znajduje się w stanie CZEKAJ, gdy równy jest zeru — w stanie BIEG. Przełączyć system ze stanu BIEG w stan CZEKAJ lub odwrotnie można tylko przez wprowadzenie nowego PSW. System można przełączyć ze stanu CZEKAJ w stan BIEG przez przerwanie we/wy, przerwanie zewnętrzne lub początkowe ładowanie systemu.

Stan PRZERWANIE ZABRONIONE — PRZERWANIE DOZWOLONE. Jednostka centralna może być zamaskowana dla przerwań zewnętrznych, we/wy, błędu maszyny oraz niektórych programowych. Znajduje się wówczas w stanie PRZERWANIA ZABRONIONE. Gdy przerwania we/wy i zewnętrzne są zamaskowane, ich zgłoszenia są przechowywane i oczekują przejścia systemu w stan PRZERWANIA DOZWOLONE. Zamaskowane przerwania programowe i przerwania błędu maszyny są ignorowane.

Przełączenia ze stanu PRZERWANIA ZABRONIONE w stan PRZERWANIA DOZWOLONE i odwrotnie dokonuje się przez zmianę odpowiednich pozycji w PSW (bity 0—7, 13 i 36—39). Gdy bit maski jest równy 1, odpowiadające mu przerwania są przyjmowane, a gdy równy jest 0, przerwania są zabronione.

Stan SUPERVISOR — PROBLEM. W stanie SUPERVISOR dozwolone są wszystkie rozkazy. W stanie PROBLEM niedopuszczalne jest wykonywanie rozkazów uprzywilejowanych, tzn. rozkazów we/wy, bezpośredniego sterowania, ochrony pamięci oraz Ładuj PSW, Ustaw maskę systemu i Diagnoza. Wybór stanu zależy od wartości 15 bitu PSW. Można go zmienić jedynie przez wprowadzenie nowego PSW.

7. Zegar

Wszystkie maszyny Jednolitego Systemu wyposażone są w zegar (ang. *timer*), który służy do odmierzania czasu rzeczywistego. Zegar jest słowem w pamięci głównej o adresie 80. Zawartość tej komórki pamięci traktuje się jako liczbę całkowitą ze znakiem, od której system odejmuje określoną wartość co 1/300 sekundy.

Gdy wartość zegara zmienia się z dodatniej na ujemną następuje przerwanie. Pełny cykl zegara wynosi 15,5 godziny. Aktualna wartość zegara dostępna jest po wykonaniu każdego rozkazu. Zegar nie jest czynny, jeśli jednostka centralna znajduje się w stanie STOP.

Zwykle zegar służy do odmierzania stosunkowo krótkich okresów. Programista może w dowolny sposób ustawić początkową wartość zegara. Dla przykładu, na podstawie zegara maszynowego, można napisać program wskazujący bieżącą porę dnia.

8. Ochrona pamięci

Komputery Jednolitego Systemu mają system ochrony pamięci, który gwarantuje, że przechowywana w pamięci informacja nie zostanie zniszczona przez przypadkowy błędny zapis lub odczytana z niewłaściwego obszaru pamięci. W celu ochrony pamięć podzielona jest na bloki po 2 Kbajty. Z każdym blokiem związany jest specjalny pięciobitowy klucz ochrony pamięci. Klucz ochrony znajduje się również w PSW i w CSW (*Słowo Stanu Kanału*, o którym będzie mowa w następnym rozdziale). Wartość tych kluczy ustalona jest przez system i musi być zgodna dla całego chronionego rozdziału pamięci.

System ochrony działa w ten sposób, że przy każdym odczycie z pamięci lub zapisie do pamięci głównej porównywany jest klucz ochrony w PSW lub w CSW z kluczem ochrony adresowanego miejsca w pamięci głównej. Dokładniej mówiąc, porównuje się cztery pierwsze bity klucza ochrony zawarte w pamięci z bitami 8—11 PSW lub bitami 0—3 CSW. Piąty bit w kluczu pamięci równy 0 oznacza, że ochrona dotyczy tylko zniszczenia przed niewłaściwym zapisem informacji, natomiast 1 oznacza ochronę zarówno przy zapisie, jak i przy odczycie.

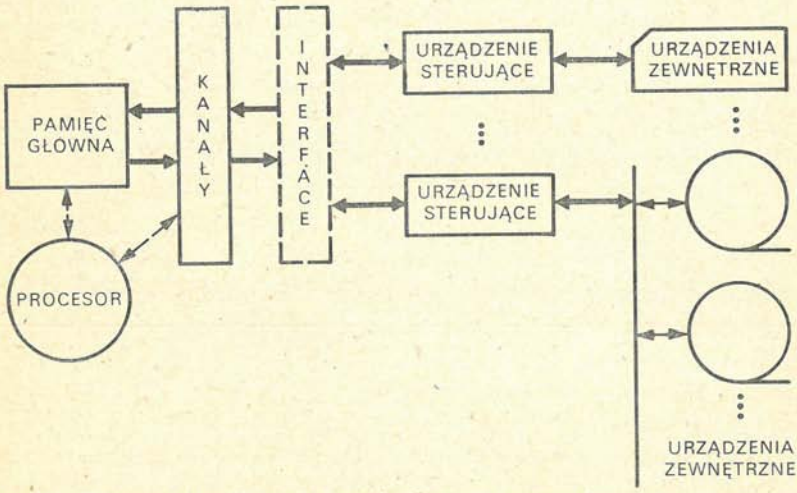
Klucze ochrony w pamięci nie należą do adresowalnej części pamięci. Można je jedynie zmieniać specjalnymi rozkazami — *Ustawić klucz pamięci* (**SSK**) i sprawdzić rozkazem *Przeczytaj klucz pamięci* (**ISK**). W wypadku gdy w toku wykonywania pewnego rozkazu klucze ochrony nie są zgodne, rozkaz nie zostanie wykonany wcale lub nie zostanie wykonany do końca. Następuje wówczas przerwanie, a ochraniany obszar pamięci pozostaje nie zmieniony.

Ochrona pamięci jest nieodzowna przy pracy wieloprogramowej. W wypadku gdy ochrona pamięci jest wyłączona, klucz ochrony w **PSW** lub **CSW** ma wartość zerową i przyjmuje się wówczas, że równy jest wszystkim kluczom ochrony w pamięci.

II. Organizacja wejścia — wyjścia

Współczesne systemy liniowe charakteryzują się wielką ilością różnorodnych urządzeń zewnętrznych, takich jak tradycyjne czytniki i perforatory kart lub taśm papierowych, pamięci taśmowe i dyskowe, drukarki, czy też nowsze urządzenia w rodzaju graficznych urządzeń wyjściowych, monitorów ekranowych, urządzeń do transmisji danych itp. Efektywne wykorzystanie tak wyposażonej maszyny cyfrowej wymaga odpowiednich środków technicznych i programowych, umożliwiających wymianę informacji między jednostką centralną a urządzeniami zewnętrznymi i pozwalających na przeprowadzanie operacji we/wy równolegle z wykonywaniem programu w procesorze.

Podstawowym urządzeniem służącym do przesyłania w systemie informacji jest kanał. Kanał jest pośrednikiem między pamięcią operacyjną



Rys. 11. Przepływ danych między pamięcią główną z urządzeniami zewnętrznymi

a urządzeniami zewnętrznymi (por. rys. 11). Cały strumień danych między urządzeniami zewnętrznymi a pamięcią operacyjną przechodzi przez kanały.

Pracę każdego urządzenia organizują jednostki sterujace, które przyjmują polecenia od kanału i interpretują je wydając sekwencje odpowiednich sygnałów sterujących do podłączonych do nich urządzeń zewnętrznych. Urządzenie sterujące może być wykonane jako samodzielna konstrukcja (pamięci dyskowe i taśmowe) lub stanowi jedną całość z urządzeniami zewnętrznymi (czytnik, drukarka). Niektóre urządzenia sterujące kierują pracą grupy urządzeń zewnętrznych tego samego typu. Na przykład do jednostki sterującej pamięcią dyskową (JS-5551) można przyłączyć do 8 jednostek pamięci dyskowej.

Jednostki sterujące połączone są z kanałami przez układy sprzęgające (ang. *interface*). W systemach liczących wprowadza się standaryzację sygnałów, układów i szyn przesyłania danych oraz zasad sterowania (tzw. *standardowy interface*[1]). Standaryzacja ta zapewnia wymienność urządzeń zewnętrznych pochodzących od różnych producentów i umożliwia tworzenie dowolnych konfiguracji systemu.

1. Kanały

Kanał[1] ma własny specjalizowany procesor, który działa według swojego programu i dysponuje własną pamięcią. Pracę kanału można krótko scharakteryzować następująco. Operacje we/wy są inicjowane przez jeden z rozkazów programu użytkowego wykonywanego w centralnym procesorze, wskutek czego zaczyna się realizacja programu kanałowego. Program kanałowy składa się z instrukcji kanałowych, które określają, jaka operacja ma być wykonana, na jakim urządzeniu zewnętrznym i do (z) jakiego obszaru pamięci mają być przesłane dane. Kanał wyznacza kolejne adresy pól pamięci, poczynając od adresu początkowo określonego przez program użytkowy i procesor centralny. Program kanałowy wykonywany jest niezależnie od pracy procesora centralnego. Gdy operacja we/wy zostaje zakończona, jednostka sterująca urządzeniem zewnętrznym sygnalizuje kanałowi zakończenie operacji we/wy i kanał zgłasza przerwanie procesorowi centralnemu.

---

[1] Por. IBM System/360. Principles of Operations. IBM Systems Reference Library, wyd. cyt.; T. Kamburelis, Architektura logiczna EMC JS. Problemy informatyki, wyd. cyt.; *Processor IBM JS-1020* (pod red. A.M. Łarionowa), Moskwa 1975.

W maszynach cyfrowych Jednolitego Systemu, podobnie jak w innych współczesnych systemach liczących, występują dwa typy kanałów: multipleksorowe i selektorowe. Część kanału, która zapewnia wymianę danych z jednym urządzeniem zewnętrznym nazywa się podkanałem. Każdy podkanał ma własną pamięć w postaci zestawu rejestrów przechowujących adresy, liczniki i informacje sterujące daną operacją we/wy.

Kanał selektorowy

Kanał selektorowy ma tylko jeden podkanał, który może obsługiwać w danej chwili tylko jedno urządzenie zewnętrzne. Jeśli więc odbywa się transmisja danych między pewnym urządzeniem zewnętrznym a pamięcią główną, to wówczas wszystkie środki techniczne kanału podporządkowane są wyłącznie temu urządzeniu (jest to tak zwany monopolowy tryb pracy). Pozostałe urządzenia we/wy przyłączone fizycznie do tego samego kanału mogą wykonywać inne operacje, lecz nie mają w tym czasie kontaktu z jednostką centralną.

Każdy model maszyny Jednolitego Systemu ma przynajmniej dwa kanały selektorowe. Kanały te charakteryzują się dużą szybkością transmisji (od 200K do 1500 Kbajtów na sekundę) i przeznaczone są dla szybkich urządzeń we/wy. W typowej konfiguracji do jednego kanału maszyny przyłącza się pamięci taśmowe, do drugiego — pamięci dyskowe.

Kanał multipleksorowy

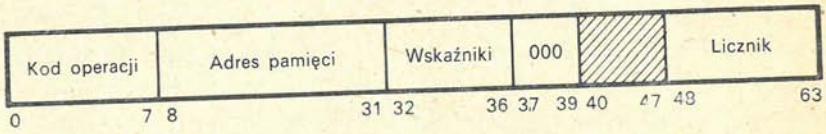
Kanał multipleksorowy ma wiele podkanałów, umożliwiających podłączenie do 256 urządzeń zewnętrznych. Kanał ten może pracować w trybie monopolowym, podobnie jak kanał selektorowy. Podstawowym jednakże sposobem pracy jest tryb multipleksorowy, w którym środki techniczne kanału oddawane są na pewien krótki okres wystarczający do przesłania jednej jednostki informacji (zwykle jednego bajtu). Ponieważ do kanału multipleksorowego przyłącza się tylko wolne urządzenia zewnętrzne, takie jak drukarka lub czytnik kart, wszystkie urządzenia pracują jednocześnie, pod warunkiem że nie nastąpi przeciążenie jego maksymalnej przepustowości. Bajty danych przekazywane do (z) poszczególnych urządzeń we/wy grupowane są w pamięci multipleksorowej i przesyłane blokami do pamięci głównej w operacjach wejścia lub w kierunku przeciwnym — w operacjach wyjścia.

Jeśli kanał nie jest zajęty transmisją, to dokonuje ciągłego przeglądu wszystkich podłączonych do niego urządzeń zewnętrznych w celu zbadania stanu tych urządzeń i sprawdzenia, czy któreś z nich nie wystąpiło z żądaniem przeprowadzenia operacji we/wy.

Szybkość przesyłania danych w zależności od typu kanału i urządzenia waha się w granicach od 16 do 145 Kbajtów/sek.

2. Słowo sterujące kanałem

W celu przeprowadzenia wymiany danych między urządzeniami zewnętrznymi a pamięcią operacyjną należy przygotować program kanałowy, w którego instrukcjach określa się, co należy zrobić z danymi, skąd i dokąd dane przesłać itp. Program kanałowy składa się z instrukcji, zwanych *słowami sterującymi kanałem* (ang. *Channel Command Word* — CCW). Każdy rozkaz CCW jest podwójnym słowem o strukturze przypominającej instrukcje maszynowe:



Poszczególne pola w CCW mają następujące znaczenie:

Pierwszy bajt CCW zawiera kod operacji wykonanej w kanale, czyli kod rozkazu kanału. Każdy typ urządzenia zewnętrznego ma właściwe kody operacji (od kilku do kilkudziesięciu). Wszystkie operacje w kanale można sklasyfikować w sześciu grupach (por. tabl. 4). Rozkaz „Czytaj" oznacza czytanie danych z nośnika urządzenia zewnętrznego, przesłanie i zapisanie ich w pamięci głównej. Rozkaz „Zapisz" powoduje przesłanie danych z określonego pola pamięci głównej do urządzenia zewnętrznego. „Czytaj wstecz" ma sens w stosunku do urządzeń zewnętrznych, w których nośniki informacji mogą przesuwać się w dwóch kierunkach, jak to bywa w pamięciach taśmowych. Rozkaz „Sprecyzuj stan" powoduje przesłanie do pamięci głównej informacji o stanie urządzenia. Informacja ta jest wykorzystywana przez program sterujący. Rozkaz „Przełącz w kanale" zmienia normalną kolejność wykonywania programu kanałowego. Rozkaz „Steruj" powoduje wykonanie czynności pomocniczych w urządzeniu zewnętrznym, np. przewinięcie taśmy magnetycznej.

Wartości kodów rozkazów kanału dla konkretnych urządzeń zewnętrznych są podane w dokumentacji technicznej tych urządzeń. Wybrane kody operacji podstawowych urządzeń zewnętrznych podane są w rozdziale IV.

TABLICA 4

Rozkazy kanałowe we/wy

Rozkazy we/wy w urządzeniach zewnętrznych	Bity w bajcie kodu operacji							
	0	1	2	3	4	5	6	7
Sprecyzuj stan	M	M	M	M	0	1	0	0
Czytaj wstecz	M	M	M	M	1	1	0	0
Zapisz	M	M	M	M	M	M	0	1
Czytaj	M	M	M	M	M	M	1	0
Steruj	M	M	M	M	M	M	1	1
Przełącz w kanale	M	M	M	M	1	0	0	0

M—bit modyfikacji (przyjmuje wartość 1 lub 0 w zależności od typu urządzenia)

Bity 8—31 słowa sterującego kanałem zawierają adres danych w pamięci głównej. Adres ten wskazuje pole w pamięci, z którego dane mają być wyprowadzone do urządzenia zewnętrznego, lub przeciwnie — mają zostać wprowadzone do tego pola.

Wskaźniki (ang. *flags*) — bity 32—36 CCW. Bit 32 jest wskaźnikiem łańcucha danych (ang. *chain data*). Jeśli bit ten równy jest jedynce, to wówczas mamy do czynienia z łańcuchem danych. Oznacza to, że gdy zostaną przesłane wszystkie dane określone w aktualnym CCW (a więc gdy licznik w CCW osiągnie wartość równą zeru) będzie wykonana następna instrukcja CCW, w której kod operacji będzie ignorowany, a obowiązywać będzie nadal kod operacji poprzedniego CCW. Łańcuch danych umożliwia przesyłanie kilku bloków danych przy jednorazowym zainicjowaniu operacji we/wy przez rozkaz w programie użytkowym. Bit 33 określa łańcuch rozkazów (ang. *chain command*). Jeśli bit 32 równy jest zeru, a bit 33 jest jedynką, wówczas występuje łańcuch rozkazów. W tym wypadku po zakończeniu bieżącej operacji będzie wykonywana operacja we/wy z tym samym urządzeniem zewnętrznym, określona w następnym CCW. Sygnał „urządzenie zewnętrzne zakończyło" nie powoduje w tym wypadku przerwania.

Bity 32 i 33 mogą wystąpić w CCW w następujących kombinacjach:

Bit 32	Bit 33	Działanie
0	0	nie ma łańcuchów; dane CCW jest ostatnim rozkazem
0	1	łańcuch rozkazów
1	0	łańcuch danych
1	1	łańcuch danych

Bit 34 umożliwia ignorowanie sygnalizacji nieprawidłowej długości danych (ang. *suppress length indicator*). Zwykle, gdy liczba bajtów, z jaką pracuje dane urządzenie, np. czytnik kart-80, różni się od liczby podanej w liczniku CCW, wówczas generowany jest sygnał błędu. Jeśli bit 34 równy jest jedynce, sygnał błędu ignoruje się.

Bit 35 jest wskaźnikiem blokady zapisywania danych w pamięci głównej (ang. *skip*). Jeśli bit ten ustawiony jest w 1, wtedy dane przesyłane są do kanału, lecz nie są wprowadzane do pamięci głównej. Jeśli bit 35 równy jest 0, wówczas operacja odbywa się normalnie.

Bit 36 określa przerwanie sterowane programowo (ang. *program controlled interrupt*). Jeżeli bit ten równy jest 1, kanał posyła do procesora sygnał żądający przerwania, gdy tylko wybrane zostanie dane CCW i rozpoczyna się operacja we/wy.

Bity 37—39 mają zawsze wartość zerową.

Bity 40—47 ignoruje się.

Bity 48—63 stanowią licznik danych. Wartość licznika wskazuje, ile bajtów danych należy przesłać. W toku wykonywania operacji we/wy wartość licznika jest zmniejszana o liczbę przesłanych bajtów.

PRZYKŁAD 1

W programie wczytuje się kartę perforowaną. Pierwsze 8 znaków należy wczytać pod adres 1833, następne dwa znaki opuścić, a pozostałe wczytać do dalszej części tego samego pola w pamięci głównej. W tablicy 5 przedstawiono program kanałowy w postaci szesnastkowej, konieczny do wykonania tego zadania.

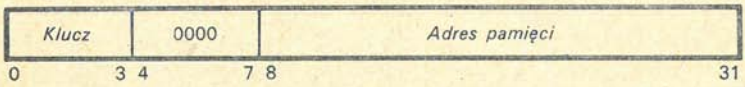
TABLICA 5

Instrukcje kanałowe do przykładu 1 w rozdziale II.

	Kod operacji	Adres pamięci	Wskaźniki		Licznik
CCW—1	02	001833	80	00	0008
CCW—2	00	00183B	90	00	0002
CCW—3	00	00183B	00	00	0046

3. Słowo adresu kanału

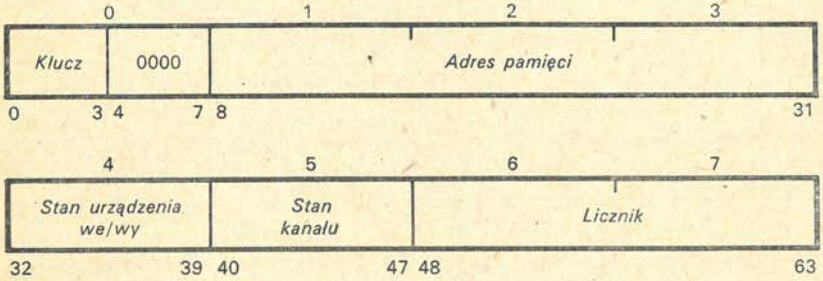
Słowo adresu kanału (ang. *Channel Address Word* — CAW) jest słowem w pamięci głównej o adresie 72. Ma ono następującą postać:



Poszczególne pola mają następujące znaczenie:
Klucz — jest to klucz ochrony pamięci, jeśli nie jest wykorzystywany, powinien być zerowy,
Adres pamięci — jest to adres pierwszego CCW programu kanału. Bity 4—7 mają zawsze wartość zerową. Informacja zawarta w słowie adresu kanału konieczna jest dla zainicjowania operacji we/wy.

4. Słowo stanu kanału

Informacje o przebiegu operacji we/wy umieszcza system w tzw. *Słowie Stanu Kanału* (ang. *Channel Status Word* — CSW). Jest to podwójne słowo zapisywane w pamięci głównej w polu o adresie 64. W CSW zawarta jest następująca informacja:



Klucz — jest to klucz ochrony pamięci, przeniesiony z CAW;
Adres pamięci — adres następnego CCW (adres bieżącego CCW+8);
Bajt stanu urządzenia — opisuje sytuację, jaka pojawiła się w urządzeniu (por. tabl. 6). Każde urządzenie zewnętrzne charakteryzuje się bajtem stanu urządzenia, opisującym szczegółowo sytuację panującą aktualnie w urządzeniu. Informacje te mają jednakową zewnętrzną postać dla

wszystkich urządzeń w Jednolitym Systemie, lecz konkretne znaczenia poszczególnych bitów mogą się różnić. Szczegółowy opis zawarty jest w dokumentacji technicznej poszczególnych urządzeń.

TABLICA 6
Bajt stanu urządzenia

Bit	Nazwa	Równa się jedynce, jeśli
32	Uwaga	operator nacisnął specjalny przycisk w urządzeniu dla łączności z kanałem.
33	Modyfikator stanu	łącznie z bitem 35 lub 37 oznacza zmianę zwykłego stanu.
34	Jednostka sterująca zakończyła	urządzenie sterujące było wykorzystane przez kilka urządzeń zewnętrznych i zakończyło swoją pracę.
35	Zajęte	urządzenie jest zajęte przez inną operację we/wy.
36	*Kanał zakończył*	kanał zakończył przesyłanie danych, urządzenie zewnętrzne może pracować nadal lub zatrzyma się.
37	*Urządzenie zakończyło*	urządzenie zakończyło swoją pracę i jest gotowe do wykonania następnej.
38	Błąd w urządzeniu	błąd w urządzeniu zewnętrznym.
39	Sytuacja wyjątkowa w urządzeniu	wystąpiła sytuacja niezwykła, ale dopuszczalna, np. koniec taśmy, koniec strony.

Bajt stanu kanału — opisuje sygnały kanału i podkanału w danej sytuacji. Znaczenie poszczególnych bitów zawarte jest w tablicy 7.
Licznik — zawiera liczbę, wskazującą do którego miejsca przesunął się licznik w ostatnim CCW. Po prawidłowym zakończeniu operacji we/wy pole równe jest zeru.

Zawartość CSW zależy od tego, jakie zdarzenie spowodowało zapisanie CSW w pamięci głównej. Gdy kanał lub urządzenie zakończy daną operację następuje przerwanie we/wy (jeśli przerwania nie są dla danego kanału

TABLICA 7

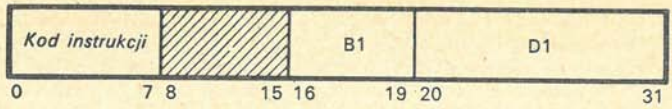
Bajt stanu kanału

Bit	Nazwa	Równy jest jedynce, jeśli
40	Przerwanie sterowane programowo.	CCW zawiera w 36 bicie warunek = 1.
41	Nieprawidłowa długość.	zawartość licznika w programie kanałowym nie pokrywa się z ilością faktycznie przesyłanych danych.
42	Błąd programu.	pojawi się niedopuszczalne CAW lub CCW, np. brak zer w bitach 4—7 w CAW.
43	Naruszenie ochrony pamięci.	klucz w CSW nie pokrywa się z kluczem ochrony pamięci.
44	Błąd danych kanału.	wystąpił błąd parzystości danych.
45	Błąd sterowania kanałem.	pojawił się niedopuszczalny sygnał sterowania kanałem.
46	Błąd w *interface*	została wykryta nieprawidłowa praca urządzenia i niepoprawny sygnał od urządzenia.
47	Błąd łańcuchowania.	wystąpiło przepełnienie kanału w czasie łańcuchowania danych.

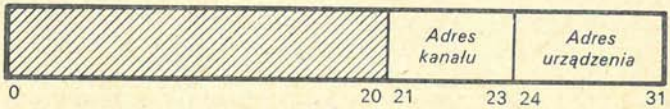
zamaskowane) i CSW wprowadzane jest do pola o adresie 64. Informacja dla CSW brana jest z bajtu stanu kanału, który znajduje się w każdym podkanale i zawiera wyczerpujące dane o stanie kanału oraz z bajtu stanu urządzenia, który przechowywany jest w każdym urządzeniu zewnętrznym. Oprócz tego bajtu każde urządzenie zapamiętuje jeden lub więcej bajtów sprecyzowania stanu. Bajty te umożliwiają dokładne zidentyfikowanie przyczyn przekłamań pojawiających się w urządzeniu zewnętrznym.

5. Instrukcje wejścia/wyjścia

W liście rozkazów maszyn Jednolitego Systemu cztery rozkazy dotyczą operacji we/wy. Wszystkie mają format SI:



Kod instrukcji oznacza tu jeden z kodów instrukcji mających mnemotechniczne symbole **SIO, TIO, HIO** lub **TCH**. Rejestr bazowy B1 i przesunięcie D1 służą do adresacji kanału i urządzenia. Suma otrzymana z dodania zawartości rejestru bazowego B1 do przesunięcia D1 określa kanał i urządzenie i interpretowana jest w następujący sposób:



Adres więc jest 11 bitowym kodem w instrukcji we/wy, w którym trzy najstarsze bity określają numer kanału, osiem pozostałych bitów numer urządzenia. Kanały mają następujące adresy:

000 — kanał multipleksorowy,
001 — kanał selektorowy 1,
010 — kanał selektorowy 2,
011 — kanał selektorowy 3 itd.

Fizycznym adresem urządzenia jest liczba od 0 do 256, co pozwala na przyłączenie do 256 urządzeń w jednym kanale. Przykładowo, jednostki pamięci dyskowej mogą mieć adresy X'190', X'191' itd., a jednostki pamięci taśmowej X'280', X'281' itd.
Instrukcje we/wy mają następujące nazwy i kody:

SIO — rozpocząć operację we/wy, kod 9C,
TIO — testować we/wy, kod 9D,
HIO — zatrzymać operację we/wy, kod 9E,
TCH — testować kanał, kod 9F.

SIO — rozpocząć operację we/wy. W instrukcji **SIO** określa się urządzenie zewnętrzne i kanał, które powinny wykonywać operację we/wy. Może to nastąpić tylko wówczas, gdy wskazane urządzenie i kanał są dostępne i nie występują przy tym żadne błędy i sytuacje wyjątkowe.
Wykonanie instrukcji przebiega w następujący sposób:
1. Następuje sprawdzenie gotowości kanału i urządzenia, przez wykorzystanie z podanego w instrukcji **SIO** adresu. Jeśli kanał nie jest dostępny,

ustawia się kod warunku równy 3 i wykonywanie instrukcji zostaje zakończone.

2. Jeśli kanał lub podkanał jest zajęty, ustala się kod warunku 2 i instrukcja nie jest dalej wykonywana.

3. Sprawdza się bajty stanu kanału i urządzenia. Jeśli w jednym z nich podana jest przyczyna, z powodu której operacja we/wy nie może być wykonana, bajty stanu zostają zapamiętane w CSW (pozostała część CSW nie ulega zmianie), a kod warunku ustawia się w 1. Instrukcja nie jest dalej wykonywana.

4. Jeśli żadna z tych sytuacji się nie zdarzy, tzn., że kanał i urządzenie są gotowe do wykonania operacji we/wy to ustawia się kod warunku równy zeru. Operacja we/wy zostaje rozpoczęta. Odbywa się to w ten sposób, że słowo adresowe kanału (CAW), wskazujące adres pierwszej instrukcji programu kanałowego (pierwszego CCW), zostaje przekazane do kanału. Dalej program kanałowy wykonuje się bez udziału procesora.

TIO — testować we/wy. Wykonanie instrukcji **TIO** powoduje wyłącznie ustawienie kodu warunku, a w określonych warunkach zapamiętanie CSW. Kod warunku ustalony zostaje według następujących zasad:

0 — sytuacja dopuszczająca wykonanie instrukcji **SIO**,

1 — zapisanie CSW,

2 — kanał lub podkanał zajęty,

3 — urządzenia wyłączone.

HIO — zatrzymać operację we/wy.Instrukcja ta powoduje przerwanie wykonywanej operacji we/wy we wskazanym urządzeniu. Ustawia się przy tym odpowiedni kod warunku.

TCH — testować kanał. Instrukcja ta nie wprowadza do kanału żadnych zmian oraz nie powoduje wykonania żadnych operacji we/wy. Jedynym działaniem instrukcji TCH jest sprawdzenie stanu kanału i ustawienie odpowiedniego kodu warunku:

0 — kanał dostępny,

1 — kanał oczekuje przerwania,

2 — kanał pracuje w trybie monopolowym,

3 — kanał nieoperatywny.

Wszystkie te cztery instrukcje we/wy nazywane są instrukcjami uprzywilejowanymi i mogą być wykonane w wypadku, gdy procesor znajduje się w stanie SUPERVISOR.

6. Początkowe ładowanie programu

Uruchomienie maszyny wymaga wprowadzenia do pamięci głównej programu. Dokonuje się tego ręcznie z pulpitu operatora. W tym celu należy nastawić na pulpicie maszyny za pomocą przełączników adres urządzenia i nacisnąć przycisk „Ładuj" (LOAD). Naciśnięcie tego przycisku powoduje:

1. Wyzerowanie wszystkich urządzeń i procesora (bajty stanu i bajty sprecyzowania stanu). Praca wszystkich urządzeń we/wy zostaje wstrzymana.

2. Wykonanie rozkazu SIO przy CAW równym 00000000 i CCW równym 02 000000 60 000018. Ustawiony na pulpicie operatorskim adres określa kanał i urządzenie zewnętrzne. Rozkaz 02 oznacza „Czytaj" dla wszystkich urządzeń we/wy. Następnie 24 bajty zostaną wczytane z określonego urządzenia zewnętrznego do pierwszych 24 komórek pamięci. Informacja ta zostanie zinterpretowana w następujący sposób: pierwsze osiem bajtów to nowe PSW, następne osiem to pierwszy rozkaz kanału (pierwsze CCW), a pozostałe kolejne osiem bajtów to drugi rozkaz kanału. Gdy wykonanie programu zostanie zakończone, zaczyna pracować procesor, który jako PSW bierze podwójne słowo z pola pamięci o adresie 0. Następne podwójne słowa spod adresu 8 i 16 wykorzystywane są jako program kanałowy, służący do prowadzenia programu.

7. Przebieg realizacji operacji we/wy

W celu podsumowania dotychczasowych informacji o systemie we/wy podamy ogólnie zasady wykonania operacji we/wy. Przebieg realizacji typowej operacji we/wy przedstawimy w kolejnych etapach.

1. W celu zainicjowania operacji we/wy należy w CAW w polu pamięci głównej o adresie 72 zapisać adres pierwszego rozkazu programu kanałowego, tzn. pierwszego CCW. Rozkazy CCW zostały wprowadzone do pamięci głównej razem z programem problemowym.

2. W programie problemowym żądanie rozpoczęcia operacji we/wy wyraża instrukcja **SIO**. Instrukcja ta określa potrzebne w danej operacji we/wy kanał i urządzenie zewnętrzne.

3. Instrukcja **SIO** ustawia kod warunku w zależności od istniejącej sytuacji w kanale i urządzeniu.

Jeśli kod warunku po instrukcji **SIO** jest różny od zera, wówczas należy przejść do podprogramu analizującego przyczynę, która nie pozwoliła rozpocząć operację we/wy.

Jeśli kod warunku równy jest zeru, oznacza to, że kanał i urządzenie są dostępne i operacja we/wy została zainicjowana. Operacja we/wy rozpoczyna się od wybrania przez kanał wskazanego urządzenia zewnętrznego. Odbywa się to w ten sposób, że kanał przesyła do wszystkich jednostek sterujących przyłączonych do niego — adres urządzenia zewnętrznego. Urządzenie, które rozpoznało swój adres podłącza się logicznie do kanału informując o tym kanał przez odwrotne przesłanie swojego adresu. Następnie kanał przesyła kod rozkazu kanałowego (o tym, gdzie znajduje się rozkaz kanałowy mówi CAW) i zaczyna się wykonywanie operacji we/wy.

4. Odpowiedni bit maski systemu (bity 0—7 w PSW) ustawia się w 1. Jeśli bit maski systemu ma wartość jeden, wtedy w odpowiadającym mu kanale dopuszczalne są przerwania. Bit zerowy świadczy o zamaskowaniu kanału, co oznacza, że kanał ten nie może spowodować przerwania.

5. Procesor kontynuuje wykonywanie programu problemowego, natomiast kanał realizuje swój program.

Niektóre operacje we/wy polegają na przesłaniu danych do jednego obszaru pamięci, określonego jednym CCW, inne wymagają kilku różnych obszarów pamięci. Mamy wtedy do czynienia z łańcuchem danych, jeśli każde CCW wskazuje różne obszary pamięci dla kodu operacji określonego na początku łańcucha.

6. Po wykonaniu programu kanałowego kanał wydaje sygnał przerwania. Przerwanie od we/wy następuje w wypadkach pojawienia się sygnałów: *Kanał zakończył* lub *Urządzenie zakończyło*. Sygnał *Kanał zakończył* wskazuje na to, że urządzenie przyjęło lub wydało wszystkie informacje związane z daną operacją i kanał nie jest już potrzebny. Sygnał: *Urządzenie zakończyło* pojawia się wówczas, gdy urządzenie zewnętrzne zakończyło wykonywanie operacji we/wy. Sygnał ten pojawia się jednocześnie z sygnałem *Kanał zakończył* lub też później. W niektórych sytuacjach brany jest pod uwagę trzeci sygnał: *Jednostka sterująca zakończyła*, świadczący o tym, że jednostka sterująca gotowa jest do wykonania następnej operacji.

Sygnały te powodują zapamiętanie całego słowa stanu kanału CSW w pamięci głównej, o ile zgłoszenie przerwania zostanie przyjęte. Klucz ochrony brany jest z CAW, a następne trzy bajty oznaczają adres w pamięci głównej ostatniego CCW plus 8. Bajt czwarty zawiera informację o stanie urządzenia, a bajt piąty zapamiętuje bajt stanu kanału. Licznik (bajty 6 i 7) wskazuje zero, jeśli operacja została wykonana całkowicie.

W wypadku łańcucha rozkazów sygnał *Urządzenie zakończyło* nie powoduje przerwania, lecz zmusza kanał do pobrania następnego CCW i wykonywania kolejnej operacji we/wy na tym samym urządzeniu.

7. Program obsługi przerwań, zwykle standardowy program systemowy, bada czy operacja zakończyła się normalnie. Potrzebne do tego dane znajdują się w CSW; analiza bajtów stanu kanału i urządzenia pozwala wykonać określone działania w razie wykrycia nieprawidłowości.

Odnotujemy, że w tym samym czasie może wystąpić więcej niż jeden sygnał przerwania od we/wy. W takim wypadku kanał i procesor ustalają priorytety tych sygnałów. W danym momencie obsługiwane jest tylko jedno przerwanie. Pozostałe oczekują w urządzeniach i podkanałach.

8. Tymczasem program problemowy mógł dojść do takiego miejsca, że dalsze jego wykonywanie byłoby niemożliwe bez wprowadzenia pewnych danych. Jeśli okaże się, że operacja we/wy została do tego czasu zakończona, program jest kontynuowany i może być realizowana następna operacja. Jeżeli jednak operacja we/wy trwa, procesor przechodzi w stan oczekiwania (w przetwarzaniu jednoprogramowym). Dopiero zakończenie operacji we/wy i sygnał przerwania zmieni wartość bitu W w „starym" PSW i program będzie mógł być kontynuowany.

Nawet z tego zwięzłego i uproszczonego przedstawienia procesów zachodzących w kanałach i urządzeniach zewnętrznych widoczna jest złożoność programowania operacji we/wy. Chociaż zrozumienie tych procesów jest bardzo przydatne, w praktyce nie ma konieczności takiego uciążliwego programowania. Program użytkownika, czyli program problemowy, wykonywany jest w stanie PROBLEM, w którym nie mogą być wykonywane podstawowe informacje we/wy, to jest **SIO**, **TIO**, **HIO** i **TCH**. Instrukcje te mogą być wykonywane tylko w stanie SUPERVISOR, co odbywa się bez udziału programisty. Innymi słowy, męczącą pracę sterowania operacjami we/wy na tym poziomie wykonuje program sterujący SUPERVISOR, wchodzący w skład systemu operacyjnego. Program ten jest przedmiotem naszych rozważań w rozdziale V. Natomiast środków do programowania operacji we/wy na wyższym poziomie dostarczają języki programowania: ASSEMBLER (por. rozdz. VI) i PL/1 (por. rozdz. VII).

III. Charakterystyki techniczne komputerów Jednolitego Systemu

Rodzina maszyn cyfrowych Jednolitego Systemu składa się z modeli JS-1010, JS-1020, JS-1022, JS-1021, JS-1030, JS-1032, JS-1050, JS-1060. Z wyjątkiem JS-1010 i częściowo JS-1021 wszystkie modele mają jednakową architekturę i charakteryzują się jednolitością programowania.

Jednolitość struktury logicznej i oprogramowania całej rodziny Jednolitego Systemu nie wyklucza różnic w technicznej realizacji poszczególnych modeli. Modele 20, 30, 40, 50, 60 zostały zaprojektowane w taki sposób, aby procesory sąsiednich modeli różniły się szybkością działania od 3 do 5 razy. Oczywiście ogólną wydajnością modele te mogą różnić się jeszcze bardziej, w zależności od pojemności pamięci operacyjnej, szybkości pracy kanałów i wyposażenia w urządzenia zewnętrzne.

W tablicy 8 przedstawiono podstawowe parametry komputerów Jednolitego Systemu.

a. Model JS-1022

Model JS-1022 jest najmniejszym modelem spełniającym wszystkie standardy Jednolitego Systemu. Jest to udoskonalona wersja modelu JS-1020[1], charakteryzująca się średnią szybkością działania, mierzoną według GIBSONA I[2], równą 80 tys. operacji na sekundę (model JS-1020 działa ze średnią szybkością 11 800 operacji na sekundę). Ze względu na charakterystyki techniczne nadaje się przede wszystkim dla zautomatyzowanych systemów zarządzania małymi i średnimi przedsiębiorstwami, do prac naukowo-technicznych i innych obliczeń.

---

[1] Por. Processor IBM JS-1020 (pod red. A. M. Łarionowa), wyd. cyt.; *Elektronnaja wyczislitelnaja maszyna JS-1020*, Statistika, Moskwa 1975.

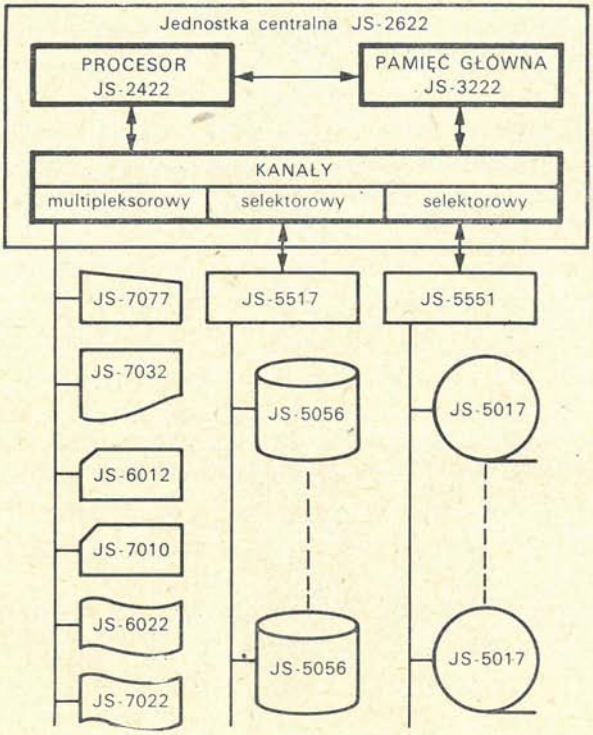
[2] Metodę obliczania wydajności procesorów JS EMC według mieszanki GIBSONA I przedstawia T. Kamburelis. Por. T. Kamburelis, op. cit.

TABLICA 8

Podstawowe parametry maszyn cyfrowych Jednolitego Systemu

Parametry	JS-1010 WRL	JS-1021 CSRS	Model				
			JS-1022 LRB i ZSRR	JS-1030 ZSRR	JS-1032 PRL	JS-1040 NRD	JS-1050 ZSRR
Średnia szybkość działania [operacji/sek]	5 000	7 000	80 000	55 000	200 000	320 000	500 000
Pamięć operacyjna:							
pojemność [Kbajty]	8—64	64	128—512	128—512	128—1024	256—1024	256—1024
cykl [μs]	0,8	2	2	1,25	1,2	1,35	1,25
czas dostępu [μs]	0,4	1	1	0,8	0,5	0,45	0,8
długość słowa (w bajtach)	16 bitów	2	4	4	4	8	8
KANAŁY:							
Szybkość transmisji (Kbajtów/sek)							
multipleksorowy	30	35	80	40	145	25	670
selektorowy		250	500	300	1 500	1 300	1 300
OPROGRAMOWANIE:							
lista rozkazów	specjalna	ograniczona lista JS	pełna uniwersalna lista Jednolitego Systemu				
system operacyjny	OS-10	MOS	DOS (OS)	DOS lub OS		OS (DOS)	

Na rysunku 12 przedstawiono typową konfigurację modelu JS-1022. Jednostka centralna składa się z procesora JS-2622, jednego kanału multipleksorowego, dwóch kanałów selektorowych, pamięci operacyjnej i urządzeń zasilających.



Rys. 12. Typowa konfiguracja modelu JS-1022

Procesor ma stałą pamięć, umożliwiającą realizację pełnej listy rozkazów Jednolitego Systemu na podstawie mikroprogramowej zasady działania. Blok arytmetyczno-logiczny wykonuje operacje na argumentach dwubajtowych, dzięki czemu uzyskano znaczne zwiększenie szybkości w porównaniu z modelem JS-1020, który jest wyposażony w arytmometr jednobajtowy.

Pamięć lokalna jednostki centralnej, zawierająca rejestry uniwersalne, wykonana jest z obwodów scalonych, zapewniających czas dostępu 275 ns.

Pamięć operacyjna może mieć pojemność 128, 256 lub 512 Kbajtów. Cykl pamięci równy jest 2 μs, a czas dostępu 1 μs. W jednym cyklu przesyła się do pamięci (lub z pamięci) cztery bajty informacji.

Kanał multipleksorowy umożliwia przesyłanie informacji między pamięcią operacyjną a wolnymi urządzeniami zewnętrznymi. Szybkość przesyłania danych wynosi 80 Kbajtów na sekundę w trybie multipleksorowym i do 400 Kbajtów na sekundę w trybie monopolowym. Maksymalnie można przyłączyć 248 urządzeń zewnętrznych i 16 urządzeń sterujących. Kanał multipleksorowy wyposażony jest w pamięć operacyjną, która konstrukcyjnie jest częścią pamięci głównej.

Szybkie urządzenia zewnętrzne, takie jak pamięć dyskowa i taśmowa, przyłącza się do kanałów selektorowych. Szybkość przesyłania danych w kanałach selektorowych wynosi maksymalnie 500 Kbajtów na sekundę. Do każdego kanału selektorowego można przyłączyć 10 jednostek sterujących.

Model JS-1022 w typowym zestawie wymaga pomieszczenia o powierzchni 100—120 m^2. Zużycie mocy wynosi około 25 KVA.

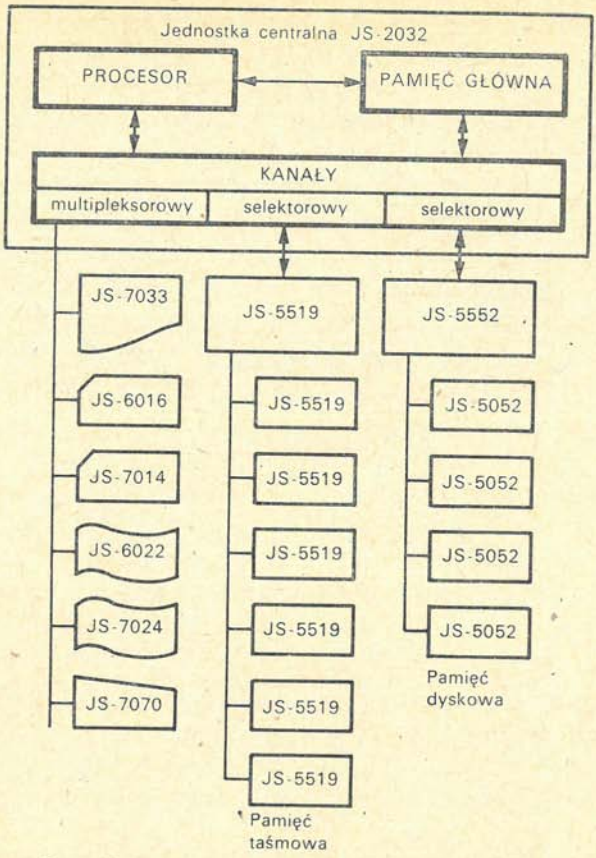
b. Model JS-1032

Produkowany w Polsce komputer JS-1032[3] należy do maszyn średniej wielkości. Przeznaczony jest do rozwiązywania różnorodnych zadań o charakterze naukowo-technicznym, ekonomicznym i administracyjnym. Średnia szybkość według mieszanki Gibsona I wynosi ponad 200 tysięcy operacji na sekundę.

Rysunek 13 przedstawia typowy zestaw komputera JS-1032. Jednostka centralna JS-2032, będąca jądrem modelu JS-1032 składa się z procesora, kanałów i pamięci operacyjnej. Procesor zawiera arytmometr, pamięć lokalną, pamięć mikroprogramów i układy sterowania. Do zadań procesora należy wykonywanie operacji arytmetyczno-logicznych, sterowanie systemem kontroli i diagnostyki, inicjowanie wymiany danych między pamięcią operacyjną a urządzeniami zewnętrznymi.

Procesor sterowany jest mikroprogramowo. Mikroprogramy znajdują się w pamięci stałej, która wraz z układami sterowania nadzoruje wykonywanie rozkazów i koordynuje pobieranie rozkazów z pamięci. Pamięć stała jest pamięcią typu transformatorowego z cyklem równym 300 ns. Pojemność jej równa się 2816 słów 86 bitowych. Użycie tak dużej pamięci stałej umożliwiło wprowadzenie dodatkowych testów technicznych dla procesora i pamięci operacyjnej oraz przeprowadzanie kontroli okresowej.

[3] Por. T. Kamburelis op. cit., *Sprzęt Jednolitego Systemu elektronicznych maszyn cyfrowych*, Ośrodek Badawczo-rozwojowy Informatyki, Warszawa 1976.



Rys. 13. Typowa konfiguracja maszyny JS-1032

Arytmometr wykonuje operacje arytmetyczne i logiczne. Zawiera on dwa typy sumatorów: sumator słowowy i bajtowy, rejestry robocze i adresowe oraz układy kontroli i diagnostyki. Sumator słowowy wykonuje operacje arytmetyczne na argumentach 32 bitowych, natomiast sumator bajtowy przeznaczony jest do działań logicznych na bajtach, m.in. również do operacji na argumentach zmiennej długości.

Wysoką niezawodność komputera osiągnięto dzięki wprowadzeniu rozbudowanego systemu kontroli. Układy kontroli przeprowadzają dynamiczną kontrolę podstawowych zespołów procesora. Oprócz tego przeprowadzana jest kontrola okresowa, w czasie której sprawdza się poprawność pracy ponad 90% sprzętu.

Arytmometr został wyposażony w specjalny układ zwany komutatorem diagnostycznym, dzięki któremu można zapisać stany wszystkich rejestrów procesora w pamięci operacyjnej, czyli utworzyć tzw. erratografię procesora. Erratografia ułatwia zlokalizowanie uszkodzeń.

Pamięć lokalna składa się z 64 słów 84 bitowych wykonanych na układach scalonych. Zawiera ona 16 rejestrów uniwersalnych, 4 rejestry zmiennoprzecinkowe i 40 rejestrów roboczych procesora i kanałów.

Pamięć operacyjna składa się z bloków o pojemności 128 Kbajtów. Całkowita pojemność pamięci może wahać się w granicach od 128 Kbajtów do 1024 Kbajtów. Czas cyklu wynosi 1,2 μs, a czas dostępu 0,5 μs. Najmniejszą jednostką informacji adresowaną w pamięci jest bajt — programista może odwoływać się do oddzielnych bajtów pamięci. Procesor jednakże ma dostęp do czterech bajtów pamięci jednocześnie, w związku z czym informacja zapisywana jest do pamięci oraz z niej odczytywana w pełnych słowach.

Każdy blok pamięci wyposażony jest w pamięć kluczy ochrony o pojemności 64 bajty, zbudowaną z układów scalonych. Dzięki temu zapewniona jest ochrona przy zapisie lub przy zapisie i odczycie z pamięci.

W skład jednostki centralnej komputera JS-1032 wchodzą trzy kanały selektorowe i jeden multipleksorowy. Pracę kanałów organizuje tzw. koordynator kanałów, który współdziała z układami sterowania procesora i pamięcią mikroprogramów oraz pośredniczy w przesyłaniu informacji. Do każdego kanału selektorowego można przyłączyć do 8 jednostek sterujących. Szybkość przesyłania równa jest 1100 Kbajtów na sekundę. Ogólna przepustowość kanałów selektorowych wynosi 2500 Kbajtów na sekundę.

Kanał multipleksorowy może mieć 128 lub 256 podkanałów. Szybkość przesyłania w trybie multipleksorowym wynosi 110 Kbajtów na sekundę, a w trybie selektorowym 250 Kbajtów na sekundę.

Na rysunku 13 przedstawiono komputer JS-1032 w typowej konfiguracji.

Konstrukcja techniczna jednostki centralnej JS-2032 opiera się na wykorzystaniu układów scalonych średniej skali integracji. Układy te zmontowane są na wielowarstwowych płytkach drukowanych o rozmiarach 295 × 150 z 84 łączówkami ze złoconymi stykami. Średnia gęstość elementów scalonych na pakiecie wynosi 81 elementów na cm^3. Jednostka centralna składa się z 28 pakietów (14 pakietów tworzy procesor i 14 kanały).

Do budowy pamięci operacyjnej użyto rdzeni ferrytowych o średnicy 0,54 mm. Pamięć składa się z modułów o pojemności 16 Kbajtów wykonanych w technice planarnej. Jeden blok zawiera 8 modułów, jeden pakiet ste-

rowania i 3 pakiety adaptera. Adapter zawiera m.in. układy nadajników i odbiorników sygnałów przechodzących przez szynę procesor-pamięć operacyjną oraz pamięć kluczy ochrony.

Pamięć operacyjna o pojemności 256 Kbajtów tworzy konstrukcyjnie jedną całość wraz z procesorem i kanałami we/wy. Rozbudowa pamięci o każde dalsze 256 Kbajtów to rozszerzenie systemu o jedną wolnostojącą konstrukcję. W maksymalnej więc konfiguracji jednostka centralna zajmuje łącznie cztery jednostki konstrukcyjne.

Komputer JS-1032 zasilany jest z przemysłowej sieci trójfazowej o częstotliwości 50± 1 Hz i napięciu nominalnym 3 × 380/220V. Jednostka centralna z pamięcią o pojemności 256 Kbajtów pobiera 3,8 KVA mocy, a każda dodatkowa jednostka pamięci operacyjnej o pojemności 256 Kbajtów wymaga 2,0 KVA.

c. Informacje o innych modelach

Ponieważ modele JS-1010 i JS-1021[4] nie realizują pełnej listy rozkazów i nie spełniają wszystkich standardów Jednolitego Systemu, poświęcimy więc im niewiele miejsca.

Model JS-1010 jest minikomputerem produkowanym w WRL. Maszyna ta ma własną listę rozkazów realizowanych programowo. Łączność z innymi urządzeniami Jednolitego Systemu zapewniona jest przez adaptery i standardowe urządzenia sterujące. Konfiguracja maszyny obejmuje konsolę operatora z elektryczną maszyną do pisania, czytnik taśmy perforowanej, dziurkarkę taśmy, pamięć dyskową o pojemności 800 Kbajtów. Zestaw ten może być rozszerzony o drukarkę wierszową, czytnik i perforator kart, pamięć taśmową oraz urządzenia do transmisji danych i terminale.

Praca na maszynie R-10 wykonywana jest pod kierownictwem specjalnego systemu operacyjnego OS-10. Użytkownik ma do dyspozycji translatory następujących języków programowania.

ASSEMBLER JS-1010 — podstawowy język symboliczny. Występuje w dwóch odmianach: ASSEMBLER-1 (skrócony zestaw rozkazów) i ASSEMBLER-2 (pełna lista rozkazów).

FORTRAN IV — przystosowany do wszystkich urządzeń peryferyjnych.

FORTRAN IVD — odmiana translatora działającego na podstawie pamięci dyskowej.

[4] Por. L.H. Korelow, *Struktury EWM i ich matiematiczeskoje obiespieczenije*, Moskwa 1974; J. Sobaniec, *Środki techniczne Jednolitego Systemu elektronicznych maszyn cyfrowych*, *Komputery Jednolitego Systemu* „Informatyka" 1973, nr 8; *Sprzęt Jednolitego Systemu elektronicznych maszyn cyfrowych*, wyd. cyt.

Real-time FORTRAN — wersja translatora w języku FORTRAN, przystosowana do sterowania procesami przemysłowymi.
BASIC — system programowania w trybie podziału czasu, przeznaczony do wykonywania obliczeń numerycznych i techniczno-ekonomicznych.
LISP 1.5 — język służący do przetwarzania danych tekstowych.

Model JS-1021 jest jednym z mniejszych w rodzinie Jednolitego Systemu. Tego typu komputer produkowany jest w CSRS. Może znaleźć zastosowanie w systemach sterowania, w systemach przetwarzania danych, a także jako pomocnicza maszyna w wielkich systemach wielomaszynowych.

W typowy zestaw modelu JS-1021 wchodzi jednostka centralna z kanałami i pamięcią operacyjną, pamięć dyskowa, drukarka wierszowa, monitor, czytniki i dziurkarki kart oraz taśmy perforowanej.

Średnia szybkość maszyny wynosi 25 do 45 tysięcy operacji na sekundę. Model JS-1021 działa na podstawie listy 65 rozkazów, będącą podzbiorem uniwersalnej listy rozkazów maszyn Jednolitego Systemu. Formaty rozkazów i danych pokrywają się ze standardami Jednolitego Systemu. Dzięki temu zapewniona jest kompatibilność z innymi modelami Jednolitego Systemu na poziomie programów napisanych w języku ASSEMBLER lub języków algorytmicznych. Kompatibilność z technicznego punktu widzenia zapewnia standardowy interface, pozwalający na podłączanie dowolnych urządzeń Jednolitego Systemu.

Komputer JS-1021 jest dostarczany z małym systemem operacyjnym (MOS JS). Typowy zestaw wymaga 50 m^2 powierzchni i zasilania 13 KVA.

Komputer JS-1040 produkowany jest w NRD. Jest to maszyna średniej wielkości, przeznaczona do obliczeń naukowo-technicznych i ekonomiczno-administracyjnych. Zrealizowano w niej pełną listę rozkazów maszynowych Jednolitego Systemu. Średnia szybkość maszyny wynosi 320 tysięcy operacji na sekundę. Maszyna może pracować zarówno z systemem operacyjnym DOS jak i OS JS.

Model JS-1040 zbudowano opierając się na procesorze JS-2040. Sterowanie pracą procesora odbywa się za pomocą układów sterowania mikroprogramowego. Podstawowy cykl maszyny wynosi 200 ns. W skład arytmometru wchodzi sumator, w którym wykonywana jest większość operacji maszynowych i arytmometr bajtowy, służący do przetwarzania rozkazów. Mikroprogramy znajdują się w stałej pamięci o pojemności 3072 słów 160 bitowych i cyklu 450 ns. Blok sterowania mikroprogramowego wydaje sygnały sterujące arytmometrem, przerwaniami i pewnymi funkcjami diagnostycznymi. Pamięć lokalna składająca się z 16 rejestrów ogólnych

i 4 rejestrów zmiennoprzecinkowych zbudowana jest z elektronicznych układów scalonych, niezależnie od pamięci operacyjnej. Rozkazy pobierane są do wykonania równolegle z wykonywaniem operacji w arytmometrze.

Model JS-1040 wyposażony jest w kanały multipleksorowy JS-4011 i kanały selektorowe JS-4034, które stanowią samodzielne moduły konstrukcyjne. Maksymalna ilość podkanałów multipleksorowych zależna jest od pojemności pamięci operacyjnej i przykładowo dla pamięci 256 Kbajtów wynosi 128. Sześć kanałów selektorowych podzielono na trzy grupy, które mają różne szybkości działania, od 1300 do 310 Kbajtów na sekundę.

W skład typowego zestawu obok jednostki centralnej z kanałami i pamięcią operacyjną wchodzą: pamięć dyskowa i taśmowa, urządzenia we/wy kartowe i taśmowe, drukarka wierszowa. Zestaw taki wymaga 150 m^2 powierzchni i 65 KVA mocy.

Model JS-1050 jest obecnie największą maszyną w serii RIAD, produkowaną w ZSRR od 1974 r. Przeznaczona jest do rozwiązywania szczególnie złożonych zadań naukowo-technicznych, ekonomicznych i informacyjnych. Maszyna ta znajduje zastosowanie w dużych ośrodkach obliczeniowych, systemach zautomatyzowanego zarządzania i systemach wielomaszynowych.

Średnia szybkość działania maszyny obliczona według Gibsona wynosi 500 tysięcy operacji na sekundę. Pamięć operacyjna może składać się z maksymalnie czterech bloków pamięci JS-3205, co w sumie daje pojemność 1024 Kbajtów. Zastosowanie szybkich kanałów (1 multipleksorowy i 6 selektorowych) umożliwia równoległą pracę wielu urządzeń peryferyjnych. Podstawową częścią systemu JS-1050 jest procesor JS-2050, wykonany na obwodach scalonych typu ECL.

W skład procesora wchodzą:

— blok arytmetyczno-logiczny,
— blok sterowania centralnego,
— blok arytmetyki dziesiętnej,
— blok sterowania pamięcią operacyjną,
— ochrona pamięci,
— urządzenia kontroli i diagnostyki,
— blok przerwań,
— zegar,
— blok łączności zewnętrznej,
— pulpit sterowania systemem,
— zasilanie.

Cykl maszyny wynosi 160 μs.

W modelu JS-1050 zrezygnowano ze sterowania mikroprogramowego na korzyść sterowania układowego, w celu zwiększenia szybkości działania systemu. Duża szybkość działania osiągana jest przez równoczesne wykonywanie operacji i przygotowanie kolejnych rozkazów do realizacji. Istotny wpływ na szybkość działania ma struktura bloku arytmetyczno-logicznego, wyposażonego w rejestry ośmiobajtowe, umożliwiające jednoczesne przetwarzanie kilku kolejnych rozkazów.

Dla organizacji współdziałania bloków procesora i kanałów z pamięcią operacyjną wprowadzono blok sterowania pamięcią. Wszystkie odwołania kanału i procesora do pamięci są spełniane w ustalonej kolejności priorytetów przez blok sterowania pamięcią. Blok ten zapewnia poza tym łączność procesora z kluczami ochrony pamięci.

Blok łączności zewnętrznej przeznaczony jest do wydawania sygnałów do urządzeń zewnętrznych lub zewnętrznego procesora, co umożliwia tworzenie systemów wielomaszynowych.

W maszynie JS-1050 kanały są samodzielnymi urządzeniami z automatycznym sterowaniem. Są to: kanał selektorowy JS-4035 i kanał multipleksorowy JS-4012. Do kanału multipleksorowego mogą być podłączone zarówno szybkie urządzenia zewnętrzne, takie jak pamięć dyskowa lub taśmowa lub wolne urządzenia typu czytników i perforatorów kart lub taśm, drukarki itp. Do kanałów selektorowych podłączone są wyłącznie szybkie urządzenia zewnętrzne. Kanał multipleksorowy ma własną pamięć ferrytową o pojemności 8 192 bajtów i cyklu 1 μs.

Model JS-1050 najbardziej efektywnie pracuje z systemem operacyjnym OS JS, lecz możliwe jest również korzystanie z DOS JS. Maszyna jest kompatybilna z pozostałymi modelami Jednolitego Systemu (oprócz JS-1010 i JS-1021) na poziomie rozkazów maszynowych i zawiera pełen zestaw rozkazów.

Maszyna JS-1050 w typowym zestawie zajmuje powierzchnię 200—250 m^2 i pobiera 70 KVA mocy.

IV. Urządzenia zewnętrzne

O efektywności systemu liczącego świadczą nie tylko parametry jednostki centralnej, ale również w dużym stopniu jego wyposażenie w urządzenia zewnętrzne. Temu właśnie zadaniu kraje współpracujące w zakresie Jednolitego Systemu urządzeniom we/wy poświęcają dużo uwagi. W efekcie liczba różnych typów urządzeń zewnętrznych już w 1974 r. przewyższała 150 i systematycznie powiększa się z każdym rokiem. W tablicy 9 przedstawione są niektóre, obecnie produkowane, urządzenia zewnętrzne podzielone na grupy ze względu na spełniane funkcje.

Wszystkie urządzenia zewnętrzne połączone są, przez swoje jednostki sterujące, z kanałami za pośrednictwem standardowego układu sprzęgającego (*interface*). Dzięki wprowadzeniu standardów obejmujących skład i przeznaczenie linii sprzęgających, sygnały i zależności czasowe między nimi występujące, informacje o stanie urządzeń itp., możliwa jest pełna wymienność urządzeń zewnętrznych produkowanych przez różne przedsiębiorstwa w różnych krajach.

W rozdziale tym przedstawimy kilka podstawowych typów urządzeń zewnętrznych[1], reprezentujący klasy urządzeń niezbędnych praktycznie w każdej konfiguracji maszyny cyfrowej Jednolitego Systemu.

1. Pamięci zewnętrzne

a. Pamięć dyskowa

Pamięć dyskowa[2] jest pamięcią masową, która ma ważną właściwość, a mianowicie bezpośredni dostęp do danych. Omówimy obecnie najbardziej

[1] Por. J. Pelc, J. Sobaniec, F. Świderski, *Środki techniczne Jednolitego Systemu* elektronicznych maszyn cyfrowych. Urządzenia zewnętrzne Jednolitego Systemu, „Informatyka" 1973, nr 9.

[2] Por. *Sprzęt Jednolitego Systemu elektronicznych maszyn cyfrowych* wyd. cyt. *Sistiema IBM/360. Wwiedienija w zapominiejuszczija ustrojstwa priamogo dostupe i mietody organizacyi dannych*, Moskwa 1974.

TABLICA 9

Urządzenia zewnętrzne

Klasa urządzeń zewnętrznych	Oznaczenie klasy	Przykłady urządzeń
Pamięci zewnętrzne:		
a) jednostki pamięci	JS-50	
pamięć dyskowa wymienna		JS-5052(LRB), JS-5056(ZSRR)
pamięć dyskowa stała,		JS-5060(WRL), JS-5051(ZSRR)
pamięć bębnowa,		JS-5035(PRL), JS-5033(ZSRR)
pamięć na kartach magnetycznych,		JS-5071(ZSRR)
pamięć taśmowa;		JS-5012(LRB), JS-5016(NRD), JS-5019(PRL), JS-5017(ZSRR)
b) jednostki sterujące.	JS-55	JS-5512(LRB), JS-5516(NRD), JS-5519(PRL), JS-5517(ZSRR), JS-5551(ZSRR)
Urządzenia we/wy:		
a) urządzenia wejścia	JS-60	
czytnik kart,		JS-6012(ZSRR), JS-6016(CSRS)
czytnik taśmy perforowanej;		JS-6021, JS-6022(PRL, WRL, CSRS)
b) urządzenia wyjścia	JS-70	
perforator kart,		JS-7010(ZSRR), JS-7014(CSRS)
perforator taśmy,		JS-7121(ZSRR), JS-7124(PRL)
drukarka wierszowa.		JS-7031(NRD), JS-7032(ZSRR), JS-7033(PRL), JS-7034(CSRS), JS-7035(NRD), JS-7038(CSRS)
Urządzenia bezpośredniej łączności operatora z maszyną:	JS-707	
elektryczne maszyny do pisania,		JS-7072(CSRS), JS-7073(NRD) JS-7074(LRB)
urządzenia z ekranem.		JS-7063(WRL), JS-7064(ZSRR)
Urządzenia do teleprzetwarzania:		
multipleksery,	JS-84	JS-8401(LRB), JS-8402(ZSRR)
terminale,	JS-85	JS-8501(LRB), JS-8561(ZSRR)
modemy,	JS-80	JS-8001(LRB), JS-8006(PRL, WRL)
urządzenia zabezpieczające.	JS-81	JS-8122(WRL), JS-8121(ZSRR)

TABLICA 9 (cd.)

Klasa urządzeń zewnętrznych	Oznaczenie klasy	Przykłady urządzeń
Urządzenia graficzne.	JS-705	JS-7051, JS-7053(ZSRR), JS-7054 (CSRS)
Urządzenia do przygotowania danych	JS-90	JS-9015(CSRS), JS-9012(ZSRR)

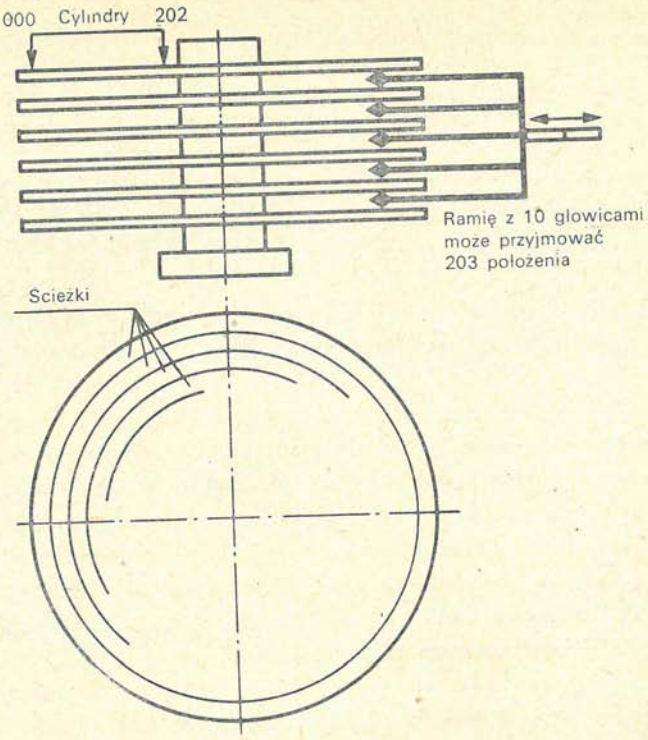
rozpowszechniony rodzaj pamięci dyskowej, pamięć z wymienionymi pakietami dysków, której reprezentantami mogą być urządzenia JS-5052 lub JS-5056.

Dane zapisuje się na zasadzie zapisu magnetycznego na dyskach aluminiowych pokrytych warstwą ferromagnetyczną. Dyski te są połączone w jedną konstrukcyjną całość, zwaną pakietem. W omawianym typie pamięci używa się pakietów składających się z sześciu dysków. Każdy dysk ma dwie powierzchnie czynne (z wyjątkiem zewnętrznych, pierwwszej i ostatniej), co przy pakiecie sześciodyskowym daje dziesięć powierzchni. Na każdej powierzchni zapisuje się informację na koncentrycznie rozmieszczonych ścieżkach za pomocą uniwersalnych głowic elektromagnetycznych (por. rys. 14).

Przy jednym obrocie pakietu dysków każda głowica przechodzi nad jedną ścieżką. Głowice nie dotykają powierzchni dysków, lecz unoszą się nad nimi podtrzymywane poduszką powietrzną, tworzącą się podczas wirowania pakietu. Dziesięć ścieżek, nad którymi jednocześnie znajdują się głowice stanowi cylinder. W pakiecie są 203 cylindry, ponieważ ramię z głowicami może przyjąć 203 pozycje. Na każdej ścieżce można zapisać 3 625 bajtów, a w całym pakiecie 7,25 Kbajtów.

Średni czas dostępu wynosi około 60 ms i można przyjąć, że jest w przybliżeniu stały, niezależny od miejsca położenia danych. Czas ten jest głównie zużywany na przesuwanie mechaniczne głowic.

Na wszystkich urządzeniach dyskowych zapisuje się dane w jednakowym formacie. Na każdej ścieżce, oprócz zapisów z danymi, znajduje się dodatkowa informacja techniczna, określająca adres ścieżki, adresy zapisów i ich długości (por. rys. 15). Na każdej ścieżce naniesiony jest stały punkt, znacznik początku ścieżki, odnośnie do którego można określać początek danych zapisanych na dysku. Za znacznikiem początku następuje adres własny ścieżki, zawierający numer cylindra i głowicy oraz bajty kontroli



Rys. 14. Pamięć dyskowa z pakietem o pojemności 7,2 Mbajtów

cyklicznej. Pierwszy zapis na ścieżce (R_0), zwany deskryptorem ścieżki, zawiera pewne informacje techniczne o ścieżce. Mogą być one wykorzystane, np. do przełączania uszkodzonej ścieżki na ścieżkę zapasową.

Dane zapamiętywane są w zapisach użytkowych w dwóch postaciach: z kluczem lub bez klucza. Zapis z kluczem składa się z trzech pól:

1) pole licznika, zawierające następujące informacje:

— znacznik, wskazujący, czy ścieżka jest czynna czy uszkodzona, oraz czy jest ścieżką zasadniczą czy zapasową,

— identyfikator, zawierający numer cylindra, numer głowicy oraz numer kolejny zapisu, licząc od początku ścieżki,

— długość klucza (od 0 do 225),

— długość danych, dwa bajty określające długość obszaru danych (w granicach od 0 do 65 535),

— liczba kontrolna,

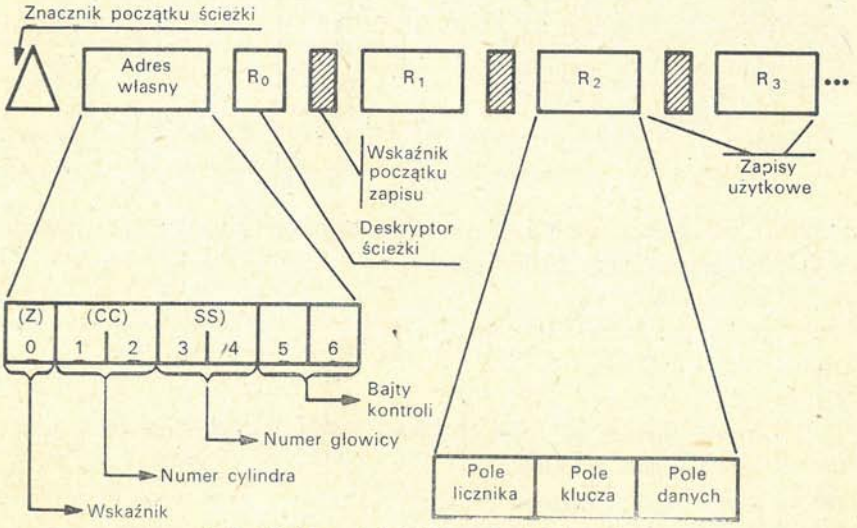
2) pole klucza (od 1 do 225 bajtów), zawierające klucz identyfikujący następujące za nim pole danych,

3) pole danych, zawierające właściwy blok danych.

Format bez klucza różni się od formatu tylko brakiem pola zawierającego klucz. W takim wypadku długość klucza w polu licznika równa jest zeru.

W tablicy 10 podane są pojemności ścieżek wyrażane jako ilość zapisów na ścieżce, w zależności od długości zapisu i jego formatu.

Każdy zapis użytkowy jest poprzedzony dwubajtowym znacznikiem początku zapisu, który jest w toku przetwarzania wykorzystywany przez jednostkę sterującą pamięcią dyskową. Wszystkie zapisy oddzielone są przerwami, koniecznymi dla ich rozróżnienia. Dyski przygotowuje się do pracy za pomocą specjalnych programów systemowych, które zapisują na nich również adresy ścieżek.



Rys. 15. Format ścieżki pamięci dyskowej

Dla zapamiętania na dysku pewnego zbioru danych określa się na nim jeden lub więcej obszarów, zwanych segmentami. Do tego celu służą specjalne operatory sterujące systemu operacyjnego. Zwykle jeden zbiór danych zajmuje kolejne cylindry pakietu dysków. Możliwe jest również zapisywanie zbiorów metodą cylindrów rozdzielonych, polegającej na tym, że jeden zbiór danych zajmuje w pewnej grupie cylindrów ścieżki 0, 1, ..., n, natomiast inny zbiór zajmuje ścieżki pozostałe, tzn. $n+1$, ..., 8, 9.

TABLICA 10

Pojemność ścieżki pamięci dyskowej typu JS-5056

Maksymalna długość zapisu w bajtach		Liczba zapisów na ścieżce	Maksymalna długość zapisu w bajtach		Liczba zapisów na ścieżce
bez klucza	z kluczem		bez klucza	z kluczem	
3625	3605	1	161	142	16
1738	1719	2	148	129	17
1130	1110	3	136	117	18
829	810	4	126	107	19
650	630	5	117	97	20
530	511	6	107	88	21
446	426	7	100	81	22
382	363	8	93	74	23
333	314	9	87	68	24
293	274	10	81	62	25
261	242	11	76	57	26
234	215	12	71	52	27
212	192	13	66	47	28
192	173	14	62	43	29
175	156	15	58	39	30

Lista rozkazów pamięci dyskowej obejmuje kilkadziesiąt pozycji, z których przytaczamy następujące:

03 — bieg jałowy,
07 — ustaw mechanizm dostępu,
0B — ustaw cylinder,
1B — ustaw głowice,
13 — ustaw głowice na zerowej ścieżce zerowego cylindra,
0F — steruj polem licznika,
1F — ustaw maskę kartoteki,
00 — testuj we/wy,
39 — szukaj adresu ścieżki,
31 — szukaj według identyfikatora,
25 — szukaj według klucza,
2D — szukaj według klucza w polu danych,
06 — czytaj dane,
16 — czytaj Ro,
19 — zapisz adres ścieżki,
1D — zapisz pole licznika, pole klucza i pole danych,

11 — kasuj,
05 — zapisz dane,
04 — sprecyzuj stan.

Na rysunku 16 przedstawiono pamięć dyskową typu JS-5052 i jej parametry.

Pojemność pakietu dysków	— 7,25 Mbajtów
Pakiet dysków	— wymienny
Liczba powierzchni roboczych	— 10
Liczba głowic	— 10
Liczba ścieżek na każdej powierzchni	— 200+3 zapasowe
Średni czas dostępu	— 60 ms
Sposób zapisu	— dwuczęstotliwościowy
Gęstość zapisu na cylindrze 000	— 30 bitów/mm
Gęstość zapisu na cylindrze 202	— 45 bitów/mm
Szybkość obrotu dysków	— 2400 obr/min
Szybkość przesyłania danych	— 156 Kbajtów/S
Sposób transmisji do jednostki sterującej	— szeregowy
Ciężar	— 167 KG
Pobór mocy	— 1 kVA
Rozmiary	— 965×765×610 mm

Rys. 16. Pamięć dyskowa typu JS-5052

b. Pamięć taśmowa

W ramach Jednolitego Systemu produkuje się wiele typów pamięci taśmowej. We wszystkich wypadkach nośnikiem informacji jest taśma magnetyczna o szerokości 1/2 cala, wykonana z tworzywa sztucznego pokrytego warstwą ferromagnetyczną. Wzajemne oddziaływanie przesuwającej się taśmy i nieruchomego pola magnetycznego głowic pozwala na utrwalenie, a później odczytanie sekwencji różniących się stanów, traktowanych jako „0" lub „1". Zapisu (i odczytu) dokonuje się jednocześnie dziewięcioma głowicami, można więc w jednym rządku zapisać jeden bajt informacji (dziewiąty bit służy do kontroli parzystości).

Stosuje się zasadniczo dwie metody zapisu magnetycznego: bez powrotu do zera (NRZ) i metodę kodowania fazowego (PE). W pierwszym wypadku zapis jedynek polega na zmianie stanu namagnesowania, a zapis zer – bez zmiany magnesowania. W metodzie fazowej dla bitu mającego taką samą wartość jak poprzedni, kierunek namagnesowania zmienia się dwukrotnie, natomiast jeżeli wartość bitu jest różna od poprzedniej, kierunek namagne-

sowania zmienia się jeden raz. Metoda ta wymaga bardziej złożonych układów elektronicznych, lecz w zamian umożliwia łatwe poprawienie wykrytych błędów. Urządzenia JS-5014 i JS-5015 działają na zasadzie PE, przy gęstości zapisu 63 znaki/mm, natomiast pozostałe typy urządzeń pracują na podstawie metody NRZ, przy gęstości zapisu 8 lub 32 znaki/mm.

Długość taśmy na jednym krążku wynosi zwykle około 750 m. W odległości około trzech metrów od początku i końca taśmy przykleja się znaczniki, które wykrywane są przez układy fotoelektryczne pamięci. Między znacznikami końca i początku zapisywane są dane w postaci bloków, o dowolnej w zasadzie długości, ograniczonej jedynie pojemnością dostępnej pamięci operacyjnej. Bloki rozdzielone są przerwą międzyblokową o długości 15,2 mm.

Zbiorom danych zapisanym na taśmie towarzyszą etykiety woluminu i kartotek, opisujące zawartość krążka taśmy i identyfikujące zbiory danych (o etykietach piszemy w rozdz. IV).

Dla zapewnienia maksymalnej niezależności zastosowano trzy rodzaje kontroli.

Kontrola poprzeczna. Jest to kontrola parzystości, polegająca na tworzeniu w każdym rządku dziewiątego bitu parzystości w toku zapisywania danych na taśmie. Następnie, w czasie czytania taśmy urządzenie podlicza ilość jedynek i sprawdza, czy jest ona zgodna z bitem parzystości.

Kontrola podłużna. W końcu każdego bloku zapisuje się jeden bajt kontrolny w ten sposób, że gdy suma wszystkich bitów wzdłuż jednej ścieżki jest parzysta, wówczas na odpowiedniej pozycji wpisuje się jedynkę. W czasie czytania bloku danych system kontroli sprawdza, czy parzystość ta została zachowana.

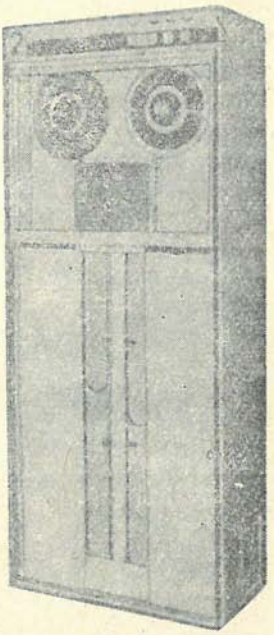
Kontrola cykliczna. W pamięciach taśmowych o gęstości zapisu 32 znaki/mm stosuje się w celach kontroli jeszcze jeden bajt, zapisywany przed bajtem kontroli podłużnej. Bajt ten, obliczany w specjalny sposób, sprawdzany jest w czasie czytania danych.

Lista rozkazów dla pamięci taśmowych zawiera około dwudziestu rozkazów, m.in.:

02 — czytaj,
0C — czytaj wstecz,
01 — zapisz,
00 — testuj we/wy,
04 — sprecyzuj stan,

07 — przewiń taśmę,
1F — zapisz znacznik.

Na rysunku 17 prezentujemy pamięć polskiej produkcji JS-5019.



Szybkość przesuwu taśmy	— 3 m/s
Czas przewinięcia krążka taśmy	— 2,6 min
Gęstość zapisu	— 8 lub 32 bity/mm
Metoda zapisu magnetycznego	— NRZ 1
Liczba ścieżek	— 9
Przerwa międzyblokowa	— 15,2 mm
Maksymalna szybkość przesyłania	— 96 Kbajtów/s
Głowica magnetyczna	— uniwersalna
Szerokość taśmy magnetycznej	— 12,7 mm
Długość taśmy magnetycznej	— maks. 750 m
Stopa błędu przy odczycie	— 10^{-7}
Pobór mocy	— 1,5 kVA
Ciężar	— 350 kG
Wymiary	— 1700×700×700 mm

Rys. 17. Pamięć taśmowa JS-5019

2. Urządzenia wejściowe

Czytnik kart

Czytnik 80-kolumnowych kart perforowanych jest podstawowym urządzeniem wejściowym, przeznaczonym do wprowadzania informacji do maszyny cyfrowej, w tym programów i operatorów sterujących pracą systemu.

Czytniki kart przystosowane są do czytania informacji zapisanej na kartach w kodzie KPK-12 (por. załącznik II). W urządzeniu następuje automatyczne przekształcanie znaków w wewnętrzny kod maszyny (DKOI). Czytanie odbywa się kolejno, kolumna za kolumną. W tym samym czasie przeprowadzana jest kontrola wczytywanej informacji,

podczas której wykrywa się niedopuszczalne kombinacje wyperforowanych dziurek. Możliwy jest inny tryb pracy czytnika, dopuszczający wczytywanie dowolnych kodów, ale wówczas urządzenie nie przeprowadza kontroli poprawności wyprowadzanych znaków.

Dla czytników kart przewidziano następujące kody rozkazów:

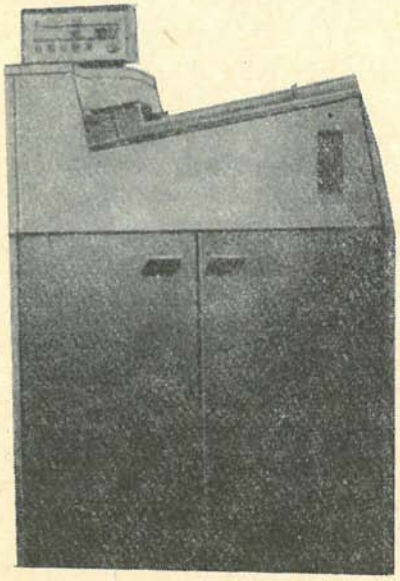
02 — czytaj z przekształceniem informacji w kod wewnętrzny,

22 — czytaj bez przekształcania,

03 — bieg jałowy,

04 — sprecyzuj stan.

Na rysunku 18 przedstawiono parametry czytnika kart typu JS-6016.



Szybkość czytania	— 1000 kart/min
Metoda odczytu	— fotoelektryczna
Sposób podawania kart	— mechaniczny
Typ karty perforowanej	— 80-kolumnowa lub 90-kolumnowa
Pojemność zasobnika podającego	— 2000 kart
Pojemność zasobnika	— 2100 kart
Pobór mocy	— 750 VA
Ciężar	— 200 kG
Wymiary	— 815 × 551 × 1248 mm

Rys. 18. Czytnik kart perforowanych JS-6016

Czytnik taśmy perforowanej

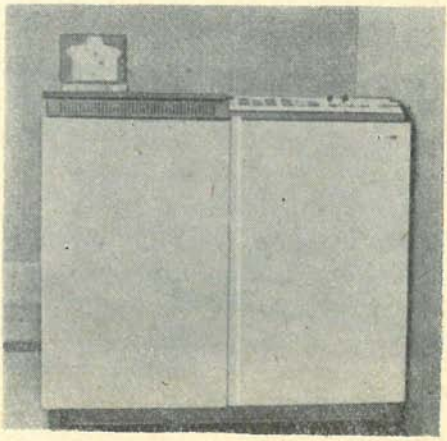
Czytnik taśmy perforowanej służy do wprowadzania do maszyny cyfrowej informacji zapisanej na taśmie perforowanej 5-, 6-, 7- lub 8-kanałowej. Przesyłanie danych do kanału maszyny cyfrowej odbywa się znak po znaku aż do momentu, gdy zostanie wczytana określona w programie ilość znaków lub przeczytany zostanie znak, ustalony jako ostatni znak bloku.

W czytnikach wykorzystuje się fotoelektryczną metodę odczytu, a szybkość waha się w granicach od 1000 do 1500 znaków na sekundę. Dane zapisane na taśmie perforowanej w kodzie KOI-7 są przekształcane w kod maszyny KOI-8. Można również wprowadzić dane w dowolnym kodzie bez przekształcania informacji (w trybie kopiowania).

W czytnikach zastosowano kontrolę danych na parzystość lub nieparzystość. Używa się następujących rozkazów:

02 — czytaj,
03 — bieg jałowy,
04 — sprecyzuj stan,
06 — czytaj do końca bloku.

Rysunek 19 przedstawia czytnik taśmy perforowanej typu JS-6022.



Szybkość czytania	— 1500 znaków/min
Metoda odczytu	— fotoelektryczna
Liczba ścieżek	— 5, 6, 7 lub 8
Kod informacji na taśmie	— KOI-7 (lub dowolny inny w trybie kopiowania)
Kod informacji na wyjściu	— KOI-8 (lub ten sam co na wejściu w trybie kopiowania)
Szerokość taśmy	— 17,5 lub 25,4 mm
Pobór mocy	— 800 VA
Wymiary	— 1200 × 500 × 1190mm

Rys. 19. Czytnik taśmy perforowanej JS-6022

3. Urządzenia wyjściowe

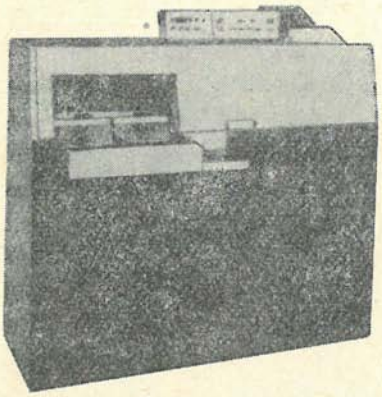
Perforator kart

Perforatory kart służą do wprowadzania informacji z maszyny cyfrowej na 80-kolumnowe karty perforowane. Perforatory mogą dokonywać przekształcenia informacji z kodu wewnętrznego maszyny DKOI na kod KPK-12 lub dziurkować w kodzie dwójkowym. Szybkość perforacji wynosi w zależności od typu urządzenia od 100 do 250 kart na minutę. W czasie perforacji dokonuje się kontroli przez porównanie wyperforowanej

informacji z zawartością buforu. W wypadku wykrycia przekłamań błędne karty kierowane są do pojemnika błędów.

W perforatorach kart zastosowano następujące kody rozkazów:
01 — zapisz z przekształceniem informacji, kartę przekazać do pojemnika 1,
21 — zapisz bez przekształcenia, kartę przekazać do pojemnika 1,
41 — zapisz z przekształceniem, kartę przekazać do pojemnika 2,
61 — zapisz bez przekształcenia, kartę skierować do pojemnika 2,
03 — bieg jałowy,
04 — sprecyzuj stan.

Na rysunku 20 przedstawiono perforator typu JS-7014.



Szybkość perforowania	— 58—117 kart/min
Metoda dziurkowania	— kolumnami
Liczba i pojemność zasobników:	
— podających	— 1 na 1500 kart
— odbiorczych	— 2 po 1400 kart
Typ karty perforowanej	— 80-kolumnowa
Kod informacji na wejściu	— DKOI
Kod informacji na karcie	— KPK-12
Pobór mocy	— 1,2 kVA
Ciężar	— 340 kG
Wymiary	— 1290×1280×730 mm

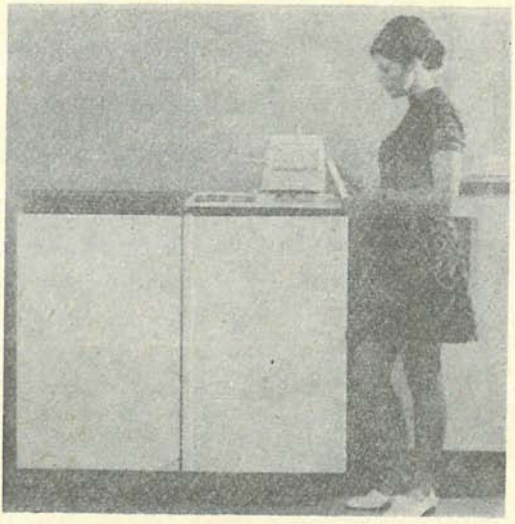
Rys. 20. Perforator kart JS-7014

Perforator taśmy

Zadaniem perforatora taśmy jest wprowadzenie danych z maszyny cyfrowej na taśmę perforowaną pięcio- lub ośmiościeżkową. Perforatory mogą pracować w trybie przekształcenia kodów wewnętrznych maszyny na siedmiobitowy kod taśmy perforowanej (KOI-7), lub bez przekształcania, perforując dowolne kody. Szybkość perforowania wynosi od 100 do 150 znaków na sekundę.

Perforatory taśmy używają następujących kodów rozkazów:
01 — zapisz,
05 — zapisz dane i znak końca bloku,
03 — bieg jałowy,
04 — sprecyzuj stan.

Na rysunku 21 podano podstawowe parametry perforowania taśmy typu JS-7022.



Szybkość perforowania	— do 150 znaków/s
Metoda dziurkowania	— elektromechaniczna
Liczba ścieżek	— 5 lub 8
Kod informacji na taśmie perforowanej	— KOI-7 (lub dowolny w trybie kopiowania)
Kod informacji na wejściu	— KOI-8 (lub dowolny w trybie kopiowania)
Sposób kontroli	— układowy na nieparzystość i zgodność kodu
Pobór mocy	— 800 VA
Ciężar	— 100 kG
Wymiary	— 1200×500×1190 mm

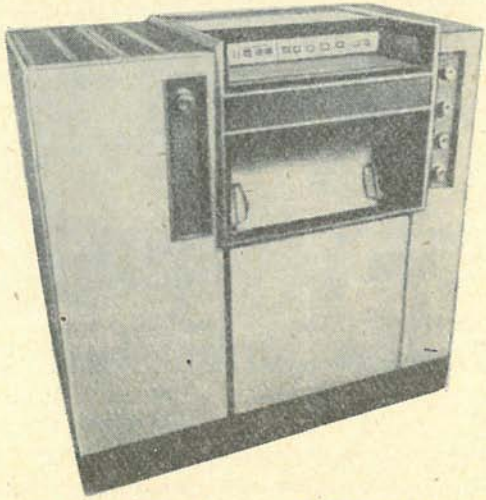
Rys. 21. Perforator taśmy papierowej JS-7022

Drukarka wierszowa

Drukarki wierszowe przeznaczone są do wyprowadzania i drukowania informacji przesyłanej z pamięci maszyny. Wszystkie typy drukarek produkowanych w ramach Jednolitego Systemu (JS-7030, JS-7031, JS-7032, JS-7033, JS-7034, JS-7035, JS-7038) to drukarki bębnowe, w których pełny zestaw znaków drukarskich znajduje się na powierzchni bębnów drukarskich. Każda drukarka wyposażona jest w bufor, w którym gromadzi informację przeznaczoną do wydruku jednego wiersza. Wydruk odbywa się na zasadzie uderzenia młotków drukarskich przez papier i taśmę

barwiącą w bęben drukarski. Dane drukowane są w pewnym określonym formacie. Sterowanie przesuwem papieru odbywa się przez perforowaną taśmę sterującą, przesuwającą się synchronicznie z papierem. Drukarka używa następujących kodów rozkazów:

0000 0001 — zapisz bez odstępu,
000M M001 — zapisz z MM odstępami po wydruku,
1CCC C001 — zapisz z przesunięciem papieru do kodu CCCC na taśmie sterującej,
000M M011 — przesuń papier o MM odstępów,
1CCC C011 — przesuń papier do kodu CCCC na taśmie sterującej,
0000 0000 — testuj we/wy,
0000 0100 — sprecyzuj stan.



Maksymalna szybkość	— 1200 wierszy/min
Gęstość drukowania	— 6 lub 8 znaków/cal
Repertuar znaków	— 96 znaki
Liczba znaków we wierszu	— 160
Liczba drukowanych	— 1+5
Sterowanie przesuwem papieru	— o 1 lub 2 wiersze programowo lub dowolnie przy pomocy taśmy sterującej
Pobór mocy	— 3,4 kVA
Ciężar	— 600 kG
Wymiary	— 1270×1250×820 mm

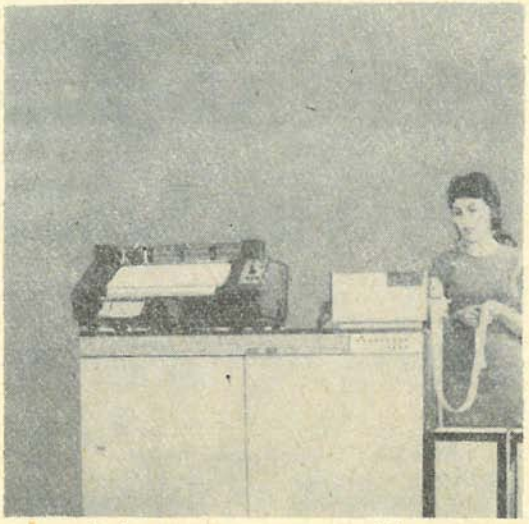
Rys. 22. Drukarka wierszowa JS-7033

Bity MM przyjmują wartość od 01 do 11 i określają ilość odstępów, o jaką ma być przesunięty papier po wydruku wiersza. CCCC może przyjmować wartość od 0001 do 1100 i określa kod na taśmie sterującej.

Rysunek 22 przedstawia drukarkę produkcji polskiej typu JS-7033 i jej parametry.

4. Niektóre inne urządzenia zewnętrzne

W Jednolitym Systemie EMC stosuje się kilka typów urządzeń do bezpośredniej komunikacji operatora z maszyną cyfrową. Urządzenia te podłączone są zwykle do kanału multipleksorowego i wykorzystywane są jako pulpity operatorskie. Przykładem mogą być czechosłowackie urzą-



Typ urządzenia	— bębnowy
Elementarny krok pisaka	— 0,1 lub 0,5 mm
Maksymalna szybkość rysowania	— 200 mm/s
Możliwość jednoczesnego przesuwu w kierunku poziomym i pionowym	— istnieje
Liczba możliwych kolorów	— 3
Format papieru	— 420×80000 mm
Wymiary obszaru roboczego	— 380×600 mm
Grubość linii	— 0,3, 0,5 lub 0,8 mm
Liczba rysowanych symboli	— 64
Pobierana moc	— 1,2 kVA
Wymiary	— 500×1200×1370 mm

Rys. 23. Pisak x-y plotter typu JS-7052

dzenia JS-7071, zbudowane na podstawie mechanizmu drukującego CONSUL 260, lub bułgarskie JS-7074, działające na podstawie mechanizmu Marica 141. Szybkość podanych urządzeń wynosi około 10 znaków na sekundę.

Rozmiar kineskopu	— 43 mm
Rozmiar użytkowy ekranu	— 250×250 mm
Częstotliwość regeneracji kadru	— 50 obrazów/s
Kolor ekranu	— zielony
Liczba punktów na ekranie	— 1024×1024
Liczba znaków w wierszu	— 74 lub 49
Liczba wierszy	— 52 lub 35
Metoda formowania znaków	— odcinkami i łukami
Maksymalna liczba znaków na ekranie	— 2100
Liczba klawiszy na pulpicie	— 32
Pojemność pamięci buforowej	— 2×4096 bajtów
Kod informacji	— DKOI
Pobór mocy	— 2 kVA

Rys. 24. Graficzne urządzenie we/wy z piórem świetlnym typu JS-7064

Do bezpośredniej komunikacji operatora z maszyną służy również urządzenie polskiej produkcji typu JS-7186, znane również jako DZM 180. Jest to drukarka mozaikowa drukująca z szybkością 180 znaków na sekundę, której zastosowanie może być znacznie szersze niż pulpit operatorski. Drukarka DZM 180 wchodzi w skład terminalu KSR. Oto jej podstawowe parametry:

— szybkość drukowania — 180 znaków/sek,
— repertuar znaków — 64 — 128,
— ilość znaków w wierszu — 132, 158,
— pamięć buforowa o pojemności 256 bajtów,
— kod wymiany informacji — DKOI,
— pobór mocy — 0,35 KVA.

Reprezentantem urządzenia graficznego może być pisak typu JS-7052, przedstawiony na rysunku 23.

Na rysunku 24 podano parametry urządzenia we/wy z piórem świetlnym typu JS-7064.

V. Dyskowy System Operacyjny

W miarę rozwoju elektronicznej techniki obliczeniowej komputery stawały się coraz bardziej skomplikowanymi konstrukcjami, trudniejsza stawała się też ich obsługa. Funkcje operatora początkowo obejmowały czynności administracyjne oraz obserwację pulpitu, reakcję na sygnały maszyny i wykonywanie takich prac, jak zakładanie papieru, taśm, kart, zmiana programów i danych itp. Wraz ze wzrostem szybkości działania maszyn przestoje w pracy komputera powodowane powolnością operatora trwały stosunkowo długo w porównaniu z czasem trwania samych obliczeń. Z tego też powodu pewne zrutynizowane czynności operatora powierzono maszynie, a więc inicjowanie pracy programu, ewidencjonowanie obliczeń itd. Jednocześnie dalsze postępy techniki cyfrowej doprowadziły do pojawienia się systemów wieloprogramowych, systemów pracujących w czasie rzeczywistym, systemów wielodostępnych, które wymagają natychmiastowego podejmowania decyzji operatorskich. Obowiązek ten przejęły specjalne programy zarządzające, zwane przez producentów EXECUTIVE, GEORGE, MONITOR, MASTER ROUTINE, DIRECTOR lub jeszcze inaczej, które dały początek systemom operacyjnym,

System operacyjny jest zespołem programów przeznaczonych do wykonywania typowych zadań związanych z zarządzaniem i obsługą operacyjną systemów liczących. Ogólnie można powiedzieć, że system operacyjny ma na celu maksymalne wyeliminowanie interwencji człowieka w toku przetwarzania oraz ułatwienie obsługi maszyny w celu osiągnięcia maksymalnej mocy obliczeniowej komputera.

System operacyjny prowadzi rejestrację wszystkich działań związanych z wykorzystaniem maszyny, rozlicza użytkowników ośrodka obliczeniowego, kieruje pracami uwzględniając priorytety i terminy, zapewnia ochronę zbiorów przed przypadkowym zniszczeniem i szybkie wznowienie pracy po awariach lub błędach maszyny.

Operator maszyny może za pośrednictwem systemu operacyjnego komunikować się z systemem w sytuacjach wymagających interwencji; otrzymuje też komunikaty o stanie urządzeń i poszczególnych prac.

Użytkownik systemu na podstawie systemu operacyjnego może sterować pracą zgodnie ze swoim życzeniem, mając zapewnioną możliwość pracy w trybie wieloprogramowym, wielodostępu, teleprzetwarzania itp.

Programista może korzystać z programów testujących, bibliotek programów i kilku lub kilkudziesięciu translatorów.

Do systemu operacyjnego włącza się również środki dla testowania urządzeń systemu, zapewniając tym samym możliwość automatycznej diagnozy niesprawności maszyny.

Już z pobieżnego przeglądu widoczne jest znaczenie systemu operacyjnego we współczesnym systemie komputerowym. Nic więc dziwnego, że koszty oprogramowania systemu, który mógłby realizować te funkcje, są bardzo wysokie. Oprócz kosztów opracowania programów, użytkownik ponosi również dodatkowe koszty, ponieważ:

— im bardziej skomplikowany jest system operacyjny, tym wymaga większej pamięci operacyjnej,

— wykonanie programów systemowych zajmuje również czas maszynowy, tak więc pewien procent czasu pracy maszyny zużywany jest na potrzeby własne.

W rozdziale tym zajmiemy się Dyskowym Systemem Operacyjnym Jednolitego Systemu (DOS JS), wzorowanym na systemie operacyjnym DOS firmy IBM. Również następne rozdziały VI, VII i VIII są poświęcone przede wszystkim DOS JS[1].

W Jednolitym Systemie maszyn cyfrowych stosuje się w zasadzie dwa rodzaje systemów operacyjnych: DOS (Dyskowy System Operacyjny), OS (System Operacyjny). Systemy te różnią się wydajnością przetwarzania. DOS może pracować ze wszystkimi modelami rodziny Jednolitego Systemu, lecz największe efekty przynosi w małych modelach (R-20, R-30) z pamięcią operacyjną do 256 Kbajtów. W większych maszynach bardziej korzystne jest stosowanie systemu OS.

Jedną z charakterystycznych cech systemów DOS i OS jest ich elastyczność, która pozwala na rozszerzanie systemu operacyjnego o nowe programy i możliwości. Środkami służącymi do zwiększenia wydajności ma-

[1] Por. *Opieracyonnaja sistiema DOS/JS. Obszczije położenija*, Moskwa 1975; *Komplet dokumentacji DOS/JS wymienionych w wykazie literatury* (dalej będziemy się powoływać na poszczególne pozycje dokumentacji przyjmując skrót, np. *DOS/JS FORTRAN — opis języka*).

szyny jest możliwość pracy w trybie przetwarzania wsadowego oraz wieloprogramowości.

System operacyjny pomaga obciążyć równomiernie urządzenia techniczne, przełączając sterowanie z jednego zadania na inne, jeśli powstaje konieczność oczekiwania na jakieś zdarzenie, np. dostarczenie danych. DOS JS pozwala użytkownikowi:

— przechowywać programy w bibliotekach w postaci źródłowej, modułów wynikowych oraz faz,

— dzielić programy źródłowe na moduły; każdy moduł może być zaprogramowany w innym języku — DOS umożliwia łączenie tych modułów w jeden program,

— stosować technikę nakładania programów, co umożliwia wykonywanie dużych programów w warunkach ograniczonej pojemności pamięci operacyjnej,

— tworzyć programy niezależnie od adresów urządzeń i kanałów,

— przetwarzać programy w trybie wieloprogramowym,

— testować programy.

1. Pojęcia podstawowe

Obecnie wyjaśnimy podstawowe pojęcia, których znajomość jest niezbędna do zrozumienia działania systemu operacyjnego DOS JS. Na dalszych stronicach książki pojęcia te zostaną rozwinięte i opisane bardziej szczegółowo.

Generacja systemu. System operacyjny DOS jest zbiorem modułów programowych, w których starano się uwzględnić potrzeby szerokiego ogółu użytkowników. Modułowa struktura systemu umożliwia adaptowanie systemu operacyjnego do wymagań użytkownika i utworzenie wariantu systemu optymalnego z punktu widzenia przeznaczenia komputera i jego konfiguracji. Proces tworzenia konkretnego wariantu systemu operacyjnego nosi nazwę generacji systemu.

Generacja systemów operacyjnych DOS i OS JS jest jedną z najbardziej charakterystycznych cech tych systemów. Wszystkie moduły systemu operacyjnego, o właściwościach określonych przez użytkownika w czasie generacji za pomocą specjalnych instrukcji, znajdują się podczas pracy systemu liczącego w pamięci dyskowej. Stąd sprowadzane są do pamięci operacyjnej tylko te moduły, które biorą udział w bieżącym procesie przetwarzania.

Zadanie. Każda praca systemu liczącego, polegająca na wykonaniu jednego lub kilku programów służących do realizacji jednego celu nosi nazwę *zadania* (ang. *job*). W zadaniu należy określić, jakie programy będą wykonywane i jakich środków wymaga ich realizacja. Jedno zadanie nie może zależeć od wykonania innego zadania i nie powinno wpłynąć na inne zadania.

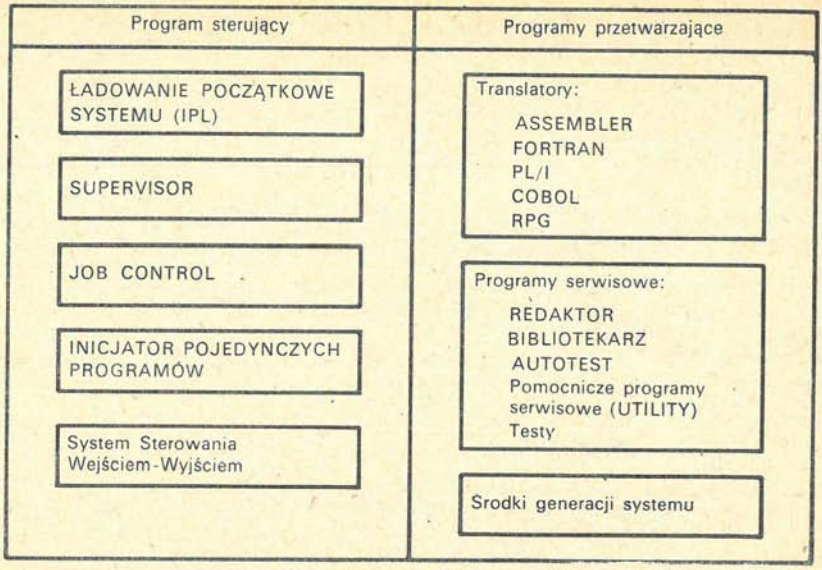
Zadanie składa się z jednego lub kilku *kroków* (ang. *step*). Kroki są to części zadania związane z użyciem jednego z programów przetwarzających. Zadanie może składać się np. z trzech kroków: translacji, redagowania i wykonania.

Przetwarzanie wsadowe. W celu zmniejszenia okresów bezczynności jednostki centralnej zadania grupuje się we wsady. Wsad jest strumieniem zadań wykonywanych w maszynie kolejno jedno za drugim. Przy takiej organizacji przetwarzania program sterujący przygotowuje kolejne zadania do wykonania, inicjuje wykonanie programów i automatycznie po wykonaniu jednego zadania przechodzi do wykonania następnego. W ten sposób likwiduje się znaczne przerwy, jakie powstają przy przetwarzaniu tradycyjnym, gdy jeden użytkownik zmienia drugiego. Jednostka centralna jest efektywnie wykorzystana, ponieważ możliwa jest jednoczesna praca urządzeń zewnętrznych i procesora.

Oprócz niewątpliwych zalet przetwarzania wsadowego, które prowadzą do zmniejszenia ogólnego czasu przetwarzania, należy wspomnieć o podstawowej wadzie tej organizacji przetwarzania, polegającej na braku kontaktu użytkownika z systemem liczącym; utrudnia to bardzo testowanie programów i uniemożliwia stosowanie technik konwersacyjnych.

Przetwarzanie wieloprogramowe. Najbardziej efektywnym sposobem eksploatacji systemu liczącego jest przetwarzanie wieloprogramowe. System operacyjny DOS JS umożliwia jednoczesne przetwarzanie maksymalnie trzech programów. Każdemu z przetwarzanych jednocześnie programów przydziela się pewien stały obszar pamięci, zwany *rozdziałem* oraz oddaje się do dyspozycji urządzenie zewnętrzne, przypisane do danego rozdziału. Praca w każdym rozdziale może przebiegać w trybie wykonywania pojedynczych rozdziałów lub w trybie wsadowym.

Kartoteki i metody dostępu. W systemie operacyjnym DOS JS zbiór danych przechowywanych na dowolnym nośniku informacji i traktowanych jako pewna całość w procesie przetwarzania nosi nazwę *kartoteki* (ang. *file*). Pojęcie to nie jest zarezerwowane wyłącznie dla danych — również programy i zadania mogą być kartotekami.



Rys. 25. Skład systemu operacyjnego DOS JS

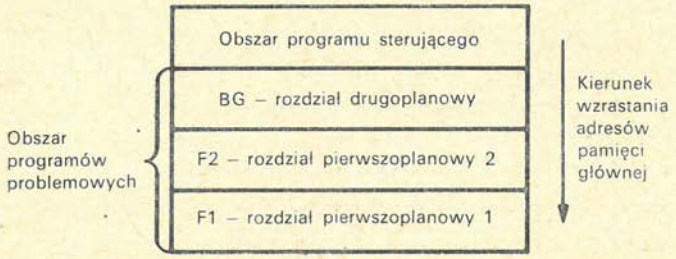
Rodzaje struktur kartotek i metody dostępu do zawartych w nich danych są bardziej szczegółowo omówione w rozdziale VIII. Tu ograniczymy się do zwrócenia uwagi na możliwość bezpośredniego dostępu do kartotek, co jest zapewnione przez obecność pamięci dyskowej w konfiguracji systemu. Bezpośredni dostęp do kartoteki jest podstawą funkcjonowania systemu operacyjnego DOS JS.

Wszystkie programy systemu operacyjnego przechowywane są w pakiecie dysków, zwanym *pakietem rezydencyjnym systemu.* Zawartość systemu DOS JS przedstawia rysunek 25. Programy przygotowujące zadania do wykonania i nadzorujące przebieg ich realizacji w maszynie tworzą tzw. *program sterujący.* Translatory i programy pomocnicze przyjęto nazywać *programami przetwarzającymi.*

2. Przetwarzanie wieloprogramowe w DOS JS

Zastosowana w DOS JS organizacja pracy wieloprogramowej charakteryzuje się stałą liczbą rozdziałów, na jakie dzieli się pamięć główną (por. rys. 26). Podziału tego dokonuje się jeszcze przed wykonaniem programów. Mogą być ustalone najwyżej trzy rozdziały, ale również dwa

lub jeden. W każdym rozdziale wykonywany jest w danym czasie tylko jeden program. Programy, które wykonywane są w różnych rozdziałach nie zależą od siebie i nie mogą na siebie wpływać. System operacyjny umożliwia jednoczesną pracę trzech lub dwóch programów, przy czym każdy wykonuje się w swoim obszarze pamięci.



Rys. 26. Podział pamięci głównej ma rozdziały

Każdy rozdział ma swoją nazwę:

BG — rozdział drugoplanowy (ang. *Background*),

F1 — pierwszy rozdział pierwszoplanowy (ang. *Foreground* 1),

F2 — drugi rozdział pierwszoplanowy (ang. *Foreground* 2).

Każdemu rozdziałowi pamięci przydziela się również urządzenia zewnętrzne.

Obszar drugoplanowy występuje zawsze, niezależnie od tego, czy system operacyjny pracuje wieloprogramowo, czy też nie. W wypadku gdy system działa wieloprogramowo, występuje co najmniej jeden z rozdziałów przedniego planu.

Jednocześnie wykonywane programy ubiegają się o dostęp do procesora. Czas dostępu do procesora przydziela SUPERVISOR, biorąc pod uwagę aktywność i priorytet rozdziałów. Rozdział jest aktywny wówczas, gdy zostaną spełnione następujące warunki:

— została przydzielona pamięć główna,

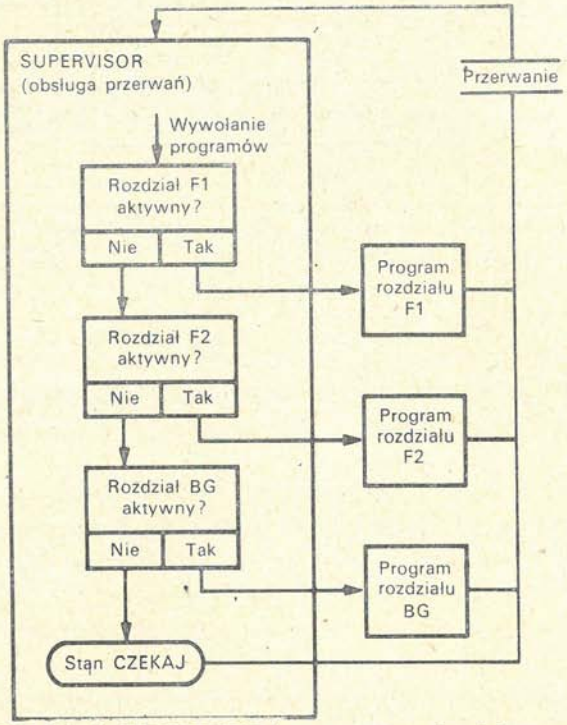
— w rozdziale znajduje się program,

— program ten jest gotowy do wykonania.

Rozdział nie jest aktywny, gdy chociaż jeden z tych warunków nie jest spełniony.

Kolejność rozdziałów jest następująca: najwyższy priorytet ma rozdział F1, następnie F2, a najniższy rozdział BG. Przy przydzielaniu czasu procesora SUPERVISOR przegląda rozdziały w kolejności malejących priorytetów i przekazuje sterowanie pierwszemu aktywnemu rozdziałowi (por. rys.

27). Program ten będzie wykonywany do tego czasu, dopóki nie stanie się aktywny rozdział o większym priorytecie; wtedy SUPERVISOR przekaże sterowanie do tego właśnie rozdziału lub do momentu, gdy w wykonywanym programie nastąpi sytuacja, kiedy program musi czekać na pewne wydarzenie, np. na zakończenie operacji we/wy. W takim wypadku sterowanie może zostać przekazane do programu o mniejszym priorytecie.



Rys. 27. Działanie SUPERVISORA w przetwarzaniu wieloprogramowym

Z punktu widzenia użytkownika pracę wieloprogramową można traktować jako pracę z wieloma maszynami, przy czym każdy rozdział z przydzielonym mu obszarem pamięci głównej i urządzeniami zewnętrznymi stanowi jedną maszynę.

Wielkość obszaru pamięci głównej przydzielonego rozdziałowi podlega pewnym ograniczeniom. Dla rozdziałów prowadzących pracę w trybie wsadowym rozdział nie może mieć mniejszej pojemności od 10 Kbajtów (tyle wymaga program JOB CONTROL). W pozostałych wypadkach

rozdział musi zawierać co najmniej 2 Kbajty. Ponieważ rozdział drugoplanowy może pracować wyłącznie w systemie wsadowym, nie może być więc mniejszy od 10 Kbajtów. W rozdziałach pierwszoplanowych możliwe jest przetwarzanie wsadowe lub pojedyncze. Przy przetwarzaniu pojedynczych programów, program problemowy jest przygotowywany do przetwarzania przez systemowy program INICJATOR POJEDYNCZYCH PROGRAMÓW, co następuje po sygnale danym przez operatora z pulpitu. Po wykonaniu w rozdziale pierwszoplanowym pojedynczego programu system nie żąda następnego programu, lecz likwiduje dany rozdział. W celu uruchomienia innego programu w tym rozdziale operator ponownie musi dać odpowiednie polecenia systemowi.

Praca wieloprogramowa jest możliwa tylko w tym wypadku, jeśli system operacyjny zawiera SUPERVISOR, który ją umożliwia. O tym, jaki wariant SUPERVISORA znajdzie się w naszym systemie operacyjnym decydujemy w czasie generacji systemu. Generacja systemu polega na wyborze ze wszystkich środków DOS tych programów, które będą najbardziej efektywne dla konkretnych zastosowań komputera. Jeśli chodzi o wieloprogramowość, można wygenerować system, który pozwala na pracę w rozdziałach pierwszoplanowych w trybie:

— przetwarzania wsadowego i pojedynczego,

— przetwarzania wyłącznie pojedynczego.

Jeśli DOS jest wygenerowany w wariancie wieloprogramowym, praca przebiega następująco:

1. Po przeprowadzeniu procedury ładowania wstępnego, tzn. po doprowadzeniu systemu do stanu gotowości, rozpoczyna się (bez interwencji operatora) przetwarzanie w rozdziale drugoplanowym i trwa tak długo, dopóki istnieje wejściowy strumień zadań dla tego rozdziału.

2. Aby zainicjować pracę w jednym z rozdziałów pierwszoplanowych, operator musi zakomunikować o tym systemowi. Przedtem powinien jednak sprawdzić, czy przydzielono temu rozdziałowi odpowiedni obszar pamięci i urządzenia zewnętrzne, w tym urządzenie, z którego podawany będzie wejściowy strumień zadań. Jeśli którykolwiek z tych warunków nie jest spełniony, operator powinien uzupełnić brakujące przydziały. Pracę w rozdziale pierwszoplanowym rozpoczyna system po przekazaniu odpowiedniej instrukcji przez operatora z pulpitu maszyny.

3. Podobnie uruchamia się pracę w drugim rozdziale pierwszoplanowym, jeśli on istnieje.

4. System przechodzi do pracy wieloprogramowej.

W celu zabezpieczenia programów zawartych w poszczególnych rozdziałach przed zniszczeniem przez pomyłkowe wtargnięcie do tych rozdziałów, stosowana jest ochrona pamięci. Każdy rozdział ma swój klucz ochrony: BG — 1, F1 — 2, F2 — 3, natomiast obszar zajęty przez program sterujący — 0. Przy próbie dostępu jakiegoś programu nie do „swojego" rozdziału, wytwarzany jest sygnał ochronny pamięci traktowany jako błąd i następuje przerwanie.

3. Urządzenia fizyczne i logiczne

Komputer jest wyposażony w dużą ilość urządzeń zewnętrznych, przyłączonych do procesora przez urządzenia sterujące i kanały. Urządzenia te (por. rozdz. IV) mają swoje numery czterocyfrowe dla oznaczenia typu; symbol ten obowiązuje w ramach całej rodziny Jednolitego Systemu. Niezależnie od tego, każde urządzenie zewnętrzne ma swój adres. Adres ten zapisuje się w następującej postaci: X'*cuu*', gdzie *c* — oznacza numer kanału, do którego podłączone jest urządzenie, *uu* — numer urządzenia w kanale. Są to adresy urządzeń fizycznych, w skrócie — adresy fizyczne. Na przykład, pamięci dyskowe mają często adresy X'190', X'191', X'192' itd., pamięci taśmowe X'280', X'281' itd., lecz nie jest to regułą. Każda maszyna cyfrowa może mieć odrębny zestaw urządzeń zewnętrznych, przy czym urządzenia te w każdym wypadku mogą mieć różne adresy fizyczne.

W celu uniezależnienia programów od konkretnych konfiguracji maszyn w systemie operacyjnym używa się adresów logicznych. Ustalono więc zbiór standardowych urządzeń logicznych i nazwami tych urządzeń posługuje się programista w swoim programie. Są to takie nazwy, jak SYSIPT, SYSLST, SYSPCH itd.; nie oznaczają one jednak konkretnego adresu urządzenia fizycznego, ani też jego typu. Przed wykonaniem programu powinien być jednoznacznie ustalony związek między używanymi w programie nazwami urządzeń logicznych i urządzeniami fizycznymi. Związek ten ustalony zostaje w czasie generacji systemu, jednakże może być w dowolnej chwili zmieniony przez operatora. Czynność ta nazywa się procedurą przydzielania urządzeń fizycznych urządzeniom logicznym.

Urządzenia logiczne dzielą się na dwie grupy: urządzenia systemowe i urządzenia programisty.

TABLICA 11

Urządzenia logiczne systemowę

Nazwa	Funkcja	Może być przydzielone urządzeniom:	Uwagi
SYSRES	Rezydent systemu	dysk	zawiera programy systemu operacyjnego i bibliotek
SYSRDR	Urządzenia do wprowadzania instrukcji programu JOB CONTROL	czytnik kart taśma magnetyczna dysk	
SYSIPT	Urządzenie wejściowe dla danych	czytnik kart taśma magnetyczna dysk	SYSIPT i SYSRDR mogą być przydzielane na te same urządzenia fizyczne, SYSIPT konieczne jest dla generacji systemu i translacji programów
SYSIN	Łączy funkcje SYSRDR i SYSIPT	czytnik kart taśma magnetyczna dysk	
SYSPCH	Urządzenie wyjściowe kartowe	perforator kart taśma magnetyczna dysk	
SYSLST	Urządzenie wyjściowe na drukarkę	drukarka taśma magnetyczna dysk	SYSPCH i SYSLST mogą być przydzielane na te sane urządzenia fizyczne
SYSOUT	Łączy funkcje SYSPCH i SYSLST	taśma magnetyczna	
SYSLNK	Urządzenie używane przez program REDAKTOR dla wprowadzania danych wejściowych	dysk	

TABLICA 11 (cd.)

Nazwa	Funkcja	Może być przydzielone urządzeniom :	Uwagi
SYSLOG	Urządzenie dla komunikacji operatora z maszyną	elektryczna maszyna do pisania drukarka	
SYSRLB	Prywatna biblioteka modułów wynikowych	dysk	pamięć dyskowa musi być tego samego typu co dla SYSRES
SYSSLB	Prywatna biblioteka modułów źródłowych		

Urządzenia logiczne systemowe

Urządzenia logiczne wykorzystywane przez programy systemu operacyjnego noszą nazwę urządzeń logicznych systemowych. Urządzenia te wymienione są w tablicy 11.

Urządzenia logiczne programisty

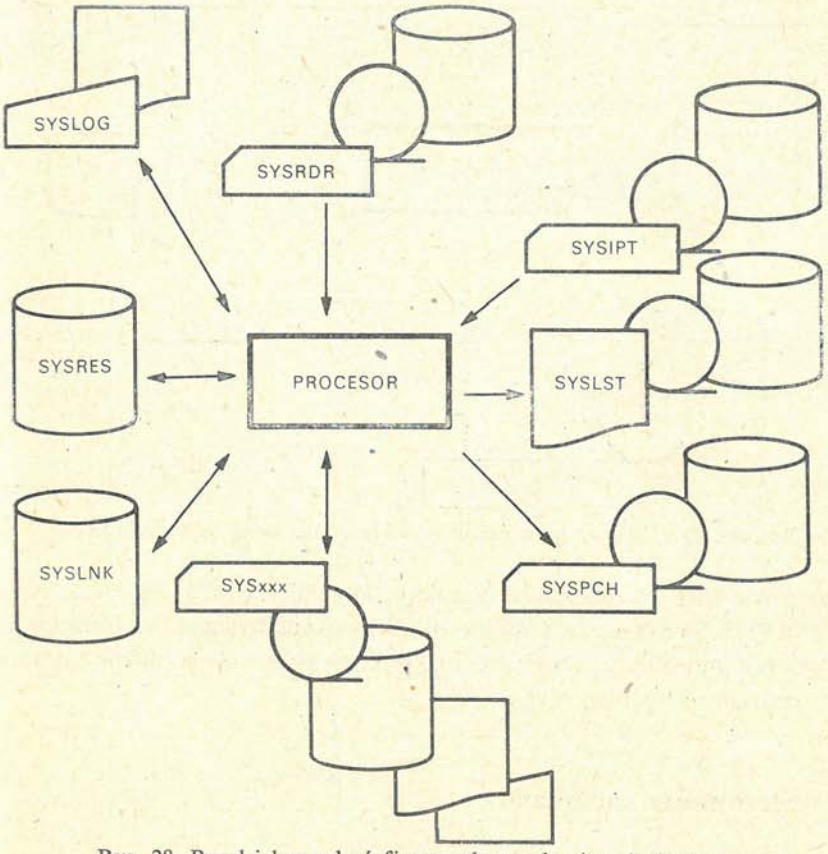
Urządzenia logiczne przeznaczone dla wykorzystania w programach użytkownika nazywa się urządzeniami logicznymi programisty. Adresy tych urządzeń mają postać SYS*xxx*, gdzie *xxx* jest liczbą od 000 do *n*; *n* jest liczbą ustaloną w czasie generacji systemu i nie może być większe od 222.

Z pewnymi ograniczeniami również urządzenia logiczne systemowe mogą być stosowane w programach użytkownika i odwrotnie, urządzenia logiczne programisty mogą być używane w niektórych wypadkach przez system operacyjny.

Przydział urządzeń

Przed wykonaniem programu należy używanym w programie urządzeniom logicznym przydzielić konkretne urządzenia fizyczne. Na rysunku 28 przedstawiono dopuszczalne warianty przydziałów urządzeń fizycznych

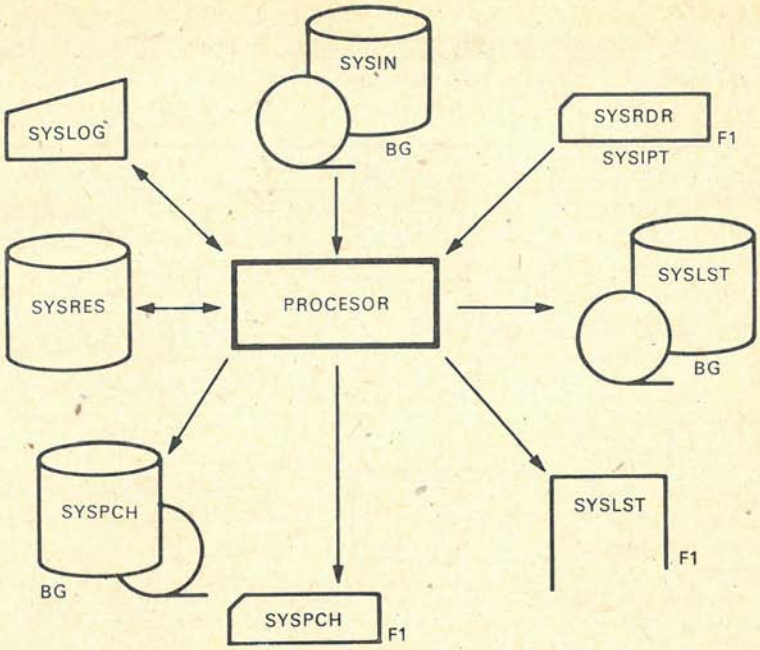
(na rysunku pokazano schematycznie wszystkie możliwe przydziały, ale należy o tym pamiętać, że w danej chwili jednemu urządzeniu logicznemu może odpowiadać tylko jedno urządzenie fizyczne).



Rys. 28. Przydział urządzeń fizycznych urządzeniom logicznym

Przykładowy przydział urządzeń w wypadku wieloprogramowości pokazuje rysunek 29: każdy rozdział ma do dyspozycji urządzenia logiczne systemowe i programowe. Suma tych urządzeń we wszystkich trzech rozdziałach nie może przewyższać 222.

W wieloprogramowym trybie pracy przydział urządzeń do rozdziałów pierwszego planu nie odbywa się w czasie generacji systemu. Jedno urządzenie fizyczne nie może być jednocześnie przydzielone urządzeniom



Rys. 29. Przykładowy przydział urządzeń w trybie pracy wieloprogramowej

logicznym różnych rozdziałów (z wyjątkiem pamięci dyskowej). Urządzenia SYSRES i SYSLOG są wspólne dla wszystkich rozdziałów. Urządzenie SYSLNK może być wykorzystywane tylko w programach wykonywanych w rozdziale drugoplanowym.

4. Sterowanie zadaniami

Przyjęciem zadania i przygotowaniem go do przetwarzania zajmuje się program JOB CONTROL. Pracę tę może wykonać on tylko wtedy, gdy otrzyma wraz z zadaniem odpowiednie informacje. Każde zadanie musi być więc opisane w pewien umowny sposób.

Opisu zadania dokonuje programista posługując się *językiem sterowania zadaniami*. Język sterowania zadaniami jest zbiorem instrukcji, które nazywać będziemy *operatorami sterującymi* (ponieważ zwykle operatory sterujące perforuje się na kartach, nazywa się je również potocznie kartami sterującymi). Niektóre polecenia do programu JOB CONTROL

TABLICA 12

Spis operatorów sterujących i dyrektyw języka sterowania zadaniami

Przeznaczenie	Operator	Dyrektywa	Spełniane funkcje
Indentyfikacja zadania	// JOB		Określenie początku zadania i jego nazwy.
	/&		Koniec zadania.

Opis kartotek	// TLBL
	// DLBL
	// EXTENT
			Dotyczy etykiet kartotek (szczegółowy opis w rozdz. VIII).

	/*		Koniec kartoteki.
Przekazanie programowi informacji	// LBLTYP		Informacja o etykietach (rozdz. VIII).
	// OPTION		Ustalenie warunków pracy systemu (por. rozdz. V).
	// UPSI		Ustawienie przełączników programowych.
	// DATE		Ustawienie daty.
Sterowanie strumieniem zadań	// PAUSE	PAUSE	Zarządzenie przerwy. System oczekuje poleceń operatora.
		CANCEL	Kasowanie zadania.
		STOP	Czasowe zatrzymanie przetwarzania wsadowego.
		UNBATCH	Odwołanie przetwarzania wsadowego i zwolnienie rozdziału.
Ustalenie parametrów systemu		ALLOC	Zmiana przydziału pamięci dla rozdziałów.
		MAP	Wydruk tablicy przydziałów pamięci dla poszczególnych rozdziałów
		SET	Ustawienie daty i pory dnia oraz niektórych innych informacji.

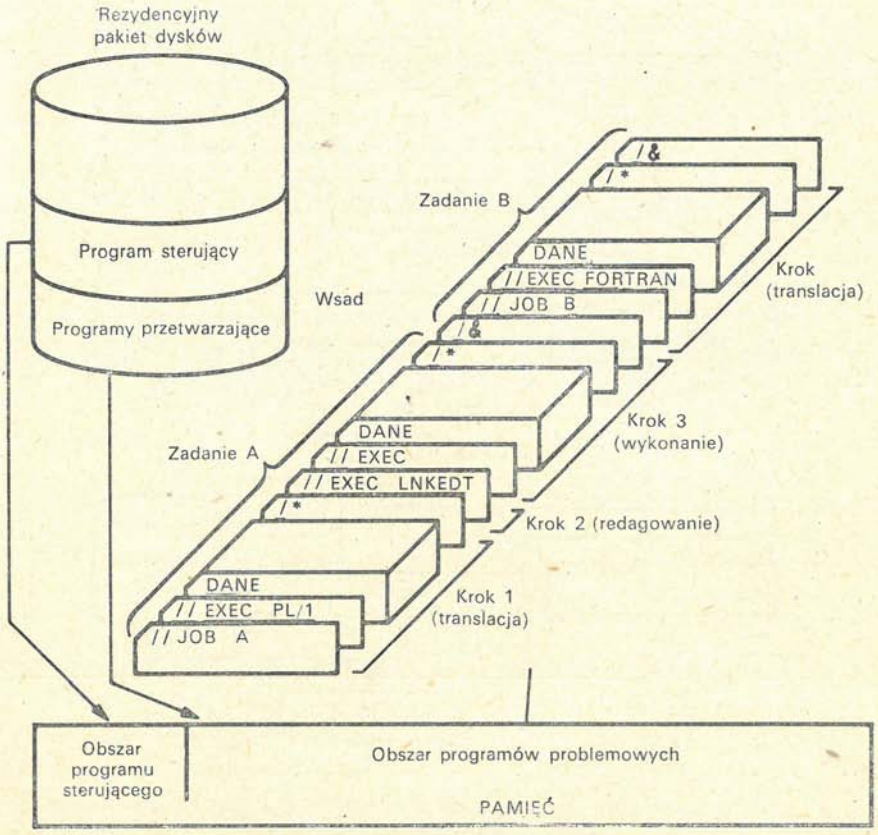
TABLICA 12 (cd.)

Przeznaczenie	Operator	Dyrektywa	Spełniane funkcje
Łączność z operatorem	// LOG	LOG	Wydruk wszystkich operatorów sterujących na monitorze.
	// NOLOG	NOLOG	Odwołanie LOG.
		KT	Koniec łączności operatora emc z systemem.
Sterowanie urządzeniami we/wy	// ASSGN	ASSGN	Przydział urządzeń fizycznych urządzeniom logicznym.
	// RESET	RESET	Przywrócenie standardowych przydziałów urządzeń.
	// LISTIO	LISTIO	Drukowanie tablicy przydziałów urządzeń.
	// CLOSE	CLOSE	Zakończenie pracy z kartoteką.
	// MTC	MTC	Podanie instrukcji sterujących pamięcią taśmową.
		DVCDN	Odłączenie wskazanego urządzenia od systemu z powodu jego niesprawności.
		DVCUP	Odwołanie odłączenia spowodowanego operatorem DVCDN.
		UNA	Odwołanie wszystkich przydziałów urządzeń w rozdziale pierwszoplanowym.
		UCS	Ustalenie niektórych kodów dla drukarki.
Wykonanie programu	// EXEC		Polecenie wykonania programu.
	// RSTRT		Uruchomienie przetwarzania programu z punktu restartu.

może wprowadzać operator elektronicznej maszyny cyfrowej z pulpitu sterującego (monitora). Nazywają się one wówczas dyrektywami i różnią się od operatorów sterujących sposobem zapisu. Pierwszymi dwoma znakami operatora sterującego są dwie skośne kreski, natomiast dyrektywy zaczynają się od razu od swojej nazwy. W tablicy 12 przedstawiono operatory sterujące i dyrektywy języka sterowania zadaniami.

Sposób opisu zadań ilustruje rysunek 30. W podanym przykładzie strumień zadań składa się z zadania o nazwie A oraz zadania B. Zadanie A składa się z trzech kroków (translacja, redagowanie, wykonanie), natomiast jedynym krokiem zadania B jest translacja.

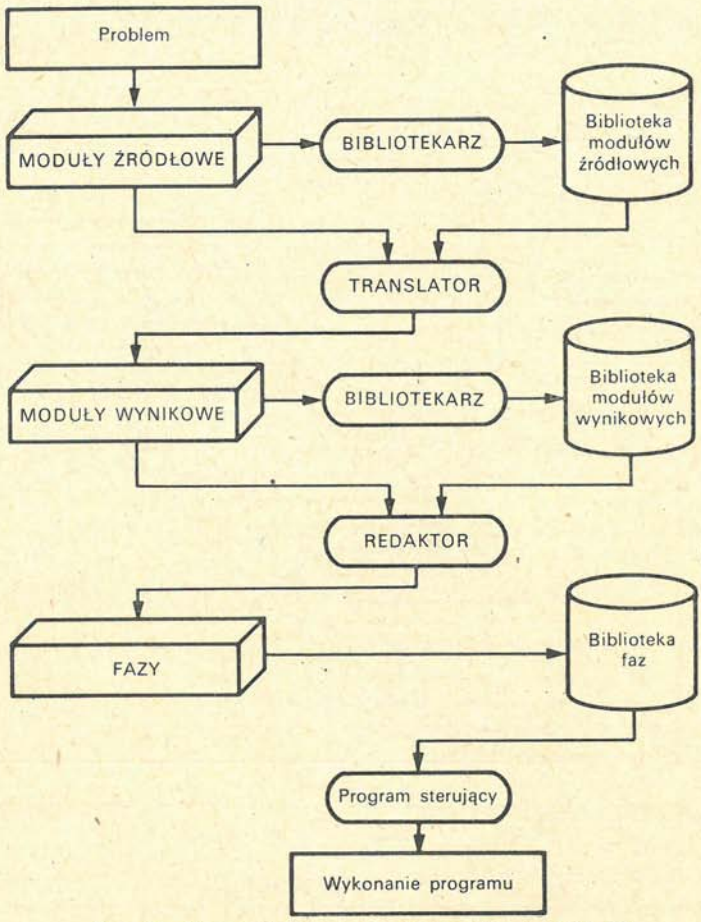
Program JOB CONTROL opisany jest w rozdziale V.



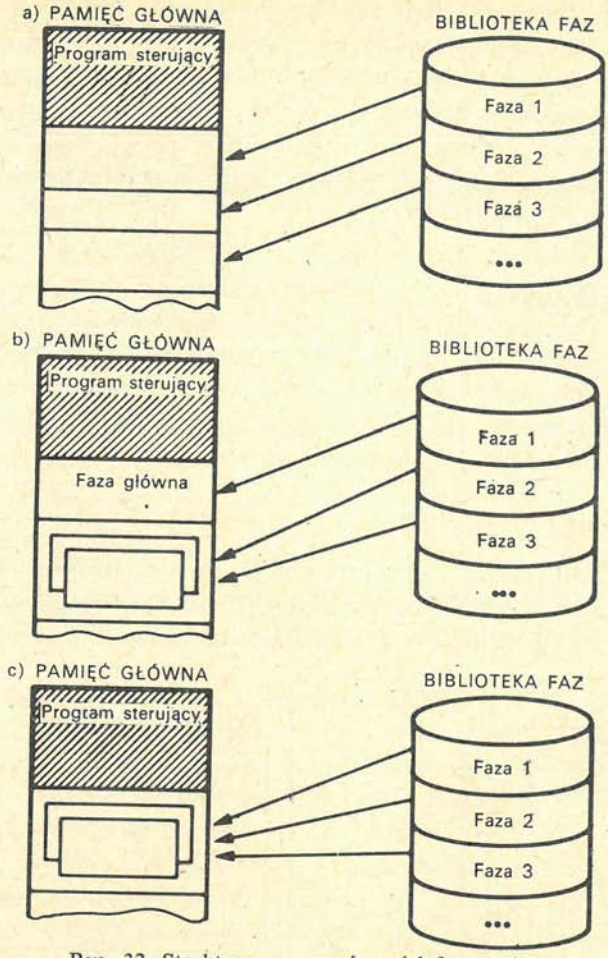
Rys. 30. Przykład strumienia danych

5. Struktury programów i etapy ich przetwarzania

Program napisany w jednym z języków programowania dostępnych w DOS JS (ASSEMBLER, PL/I, RPG, COBOL, FORTRAN) nosi nazwę *modułu źródłowego*. Pierwszym etapem przetwarzania jest translacja modułu źródłowego (por. rys. 31). W wyniku translacji otrzymuje się *moduł wynikowy*, który może zostać wprowadzony do biblioteki modułów wynikowych lub być natychmiast dalej przetwarzany przez program REDAKTOR. Zadaniem tego programu jest łączenie modułów wynikowych w jeden



Rys. 31. Etapy przetwarzania programów w DOS JS



Rys. 32. Struktury programów wielofazowych
a) struktura zwykła, b) struktura nakładowa z fazą główną, c) struktura nakładkowa bez fazy głównej

program i ostateczne przygotowanie go do wykonania. Program gotowy do wykonania, zwany *fazą*, przechowywany jest w bibliotece faz w pamięci dyskowej. Stąd program sterujący przenosi fazy do pamięci głównej i nadzoruje ich wykonanie.

W najprostszym wypadku program składa się z jednej fazy. Większe programy składają się zwykle z kilku faz, które mogą być kolejno wykonywane w tym samym obszarze pamięci głównej. Struktura taka nazywa się

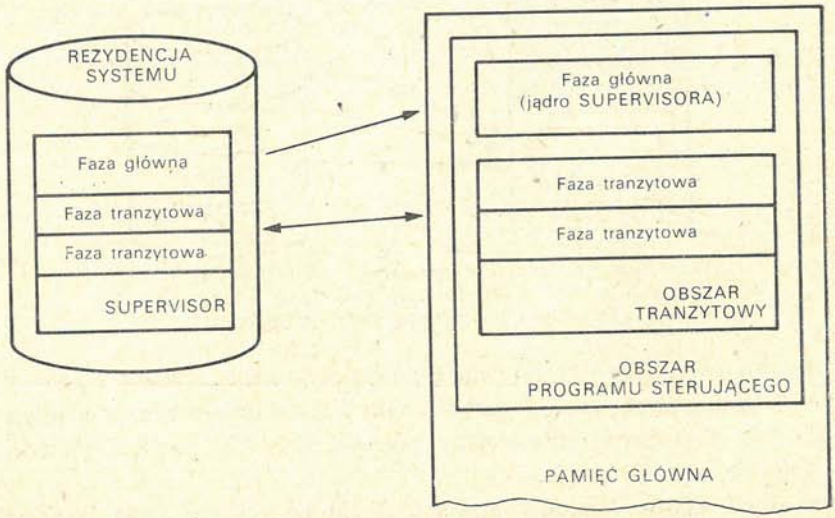
strukturą nakładkową. Użycie struktur nakładkowych pozwala zmniejszyć konieczną dla danego programu pojemność pamięci. Możliwe są dwa sposoby użycia struktur nakładkowych: struktura z fazą główną (która znajduje się zawsze w pamięci operacyjnej i sprowadza kolejno pozostałe fazy do pamięci głównej) oraz struktura bez fazy głównej, w której każda faza oprócz ostatniej sprowadza na swoje miejsce następną fazę (por. rys. 32).

6. SUPERVISOR

SUPERVISOR[2] jest główną częścią systemu operacyjnego. Jego zadaniem jest sterowanie wykonywaniem programu od momentu wprowadzenia go do maszyny aż do wydania wyników. SUPERVISOR przejmuje sterowanie w momencie powstania przerwania.

a. Struktura SUPERVISORA

SUPERVISOR składa się z faz przechowywanych w bibliotece faz na dysku rezydencyjnym systemu (por. rys. 33). Jedna z faz, zwana fazą główną lub jądrem SUPERVISORA, znajduje się w czasie pracy systemu w pamięci



Rys. 33. Struktura SUPERVISORA

[2] Por. *DOS/JS, Programy sterujące.*

głównej w obszarze programu sterującego. W fazie głównej znajdują się podprogramy najczęściej używane. Pozostałe fazy, zwane fazami tranzytowymi, realizują rzadziej używane funkcje SUPERVISORA i są w miarę potrzeby umieszczane w pamięci głównej. Fazy tranzytowe są wprowadzane do pamięci głównej przez jądro SUPERVISORA, natomiast samo jądro zostaje załadowane przez program ŁADOWANIE POCZĄTKOWE SYSTEMU (IPL).

Do zadań SUPERVISORA m. in. należy:

— rozpoznawanie i obsługa przerwań,
— planowanie pracy kanałów,
— obsługa przekłamań urządzeń zewnętrznych,
— sprowadzenie do pamięci faz programowych z biblioteki faz,
— wykonanie czynności związanych z zakończeniem programu i zadania,
— organizacja komunikacji z operatorem maszyny,
— obsługa zegara,
— tworzenie punktów kontrolnych,
— organizacja pracy w systemie wieloprogramowym.

Wymienione funkcje omówimy kolejno.

b. Obsługa przerwań

Wykonywanie programu może zostać przerwane wskutek:

— przekłamania w układach elektronicznych maszyny,
— przekłamania programowego,
— przywołania SUPERVISORA,
— przyczyny zewnętrznej,
— sygnału od systemu we/wy.

W rozdziale I przeanalizowaliśmy przebieg procesu przerwania. Zadaniem SUPERVISORA jest zidentyfikowanie przyczyny wywołującej przerwanie. Podczas przerwania „stare" PSW zostaje zastąpione „nowym", które wskazuje punkt wejścia do odpowiedniego podprogramu SUPERVISORA, w zależności od typu przerwania. SUPERVISOR zapamiętuje pełną informację o stanie przerwanego programu. Obok informacji zawartych w PSW, zostają zapamiętane w *tablicy stanu programu* takie dane, jak stany rejestrów, czas rozpoczęcia zadania, długość etykiet itp.

Podczas pracy wieloprogramowej każdy rozdział ma swoją tablicę stanu programu. Dla rozdziału BG obszar ten znajduje się wewnątrz obszaru

wydzielonego dla programu JOB CONTROL, a dla rozdziałów pierwszoplanowych — na początku każdego rozdziału.

Po zapamiętaniu stanu programu SUPERVISOR przystępuje do obsługi przerwania, która polega na sprecyzowaniu przyczyny przerwania i wykonaniu odpowiednich podprogramów. Konkretna przyczyna przerwania w ramach określonej klasy możliwa jest do ustalenia na podstawie kodu przerwania w „starym" PSW.

Klasy przerwań

Przywołanie SUPERVISORA. W razie użycia rozkazu Przywołaj *SUPERVISOR* (**SVC**), następuje przerwanie. Ma ono na celu przekazanie sterowania SUPERVISOROWI w związku z wykonaniem przez niego określonej funkcji. Mogą to być takie funkcje, jak wykonanie operacji we/wy, załadowanie fazy do pamięci głównej celem jej wykonania, otwarcie lub zamknięcie kartoteki itd.

Programista nie używa zwykle rozkazu **SVC** w jawnej postaci. Rozkaz ten występuje w pewnych makroinstrukcjach inicjujących odpowiednie programy DOS JS. W następujących makroinstrukcjach zawarty jest rozkaz **SVC**:

CANCEL — anulować pozostałe kroki zadania,
CHKPT — utworzyć punkt restartu w programie problemowym,
CLOSE — zamknąć kartotekę,
COMRG — pobrać adres obszaru komunikacji (pozwala na dostęp do obszaru komunikacji w celu pobrania pewnych informacji),
DUMP — wydrukować zawartość pamięci głównej i zakończyć wykonywanie programu problemowego,
EOJ — koniec zadania,
EXCP — wykonać program kanałowy,
EXIT — wrócić do programu problemowego po obsłużeniu przerwania przez własny program obsługi,
FETCH — załadować fazę do pamięci głównej w celu wykonania,
GETIME — podać czas rzeczywisty,
LBRET — wyjść z podprogramu przetwarzania etykiet niestandardowych,
LOAD — ładować z biblioteki faz do pamięci głównej fazę, która nie będzie natychmiast wykonywana,
MVCOM — przesłać informację do obszaru komunikacji,

OPEN — otworzyć kartotekę,
PDUMP — wydrukować zawartość zadanego obszaru pamięci,
SETIME — przerwać wykonywanie programu po upływie zadanego czasu,
STXIT — wskazuje SUPERVISOROWI, że użytkownik będzie korzystał z własnych programów obsługi przerwań,
WAIT — czekać do czasu zakończenia pewnych operacji.

Przerwanie programowe. Przerwanie programowe wywoływane jest nieprawidłowym użyciem instrukcji lub danych. Przyczynę przerwania wskazują 28—31 bity „starego" PSW, zapisanego w polu pamięci o adresie 40.

W wypadku wystąpienia przerwania programowego SUPERVISOR może przedstawić jedno z następujących działań:

— przekazać sterowanie do programu napisanego przez użytkownika w celu obsługi tego przerwania,

— wydrukować zawartość pamięci i zakończyć wykonywanie zadania,

— zakończyć wykonywanie zadania.

Sterowanie zostaje przekazane do podprogramu użytkownika wtedy, gdy poprzednio zgłosił on swój program obsługi przerwań za pomocą makroinstrukcji **STIXT**. W przeciwnym razie zadanie zostanie zakończone i skasowane.

Przerwanie od układów kontroli poprawności działania maszyny. W wypadku wykrycia błędnego działania układów maszyny następuje przerwanie i maszyna przechodzi w stan CZEKAJ. Wszystkie inne przerwania są wtedy niedozwolone.

Przerwania od we/wy. Przerwania tego typu mogą wystąpić w sytuacji, gdy:
— kanał zakończył pracę i jest wolny,
— urządzenie zewnętrzne zakończyło pracę i jest wolne,
— urządzenie sterujące zakończyło pracę i jest wolne,
— wciśnięto przycisk AR (ang. *Attention Request*) na pulpicie operatorskim, żądający komunikacji z systemem,
— przerwanie było zaprogramowane w programie kanałowym.
Obsługa tego typu przerwań należy do SUPERVISORA i będzie omówiona nieco później, gdy zajmiemy się planowaniem pracy kanałów.

Przerwanie zewnętrzne. Przyczyną przerwań zewnętrznych może być sygnał zewnętrzny, sygnał od zegara lub sygnał od przycisku PRZER-

WANIE, znajdującego się na pulpicie operatorskim systemu. O przerwaniach tego typu będzie mowa w dalszej części tego rozdziału.

c. Planowanie pracy kanałów

Planowaniem pracy kanałów zajmuje się grupa programów SUPERVISORA, spełniających następujące funkcje:

— prowadzenie kolejek zgłoszeń żądań dostępu do urządzeń we/wy,
— inicjacja operacji we/wy,
— obsługa przerwań we/wy,
— obsługa przekłamań w operacjach we/wy,
— kontrola pracy z dyskami.

Wszystkie programy użytkownika wykonane są w stanie PROBLEM, w którym niedozwolone są m. in. operacje we/wy. Natomiast w stanie SUPERVISOR dozwolone są wszystkie rozkazy, dlatego do wykonania operacji we/wy potrzebny jest udział SUPERVISORA. Zadaniem programisty korzystającego z fizycznego systemu sterowania we/wy jest przygotowanie programu kanałowego. Oprócz tego programista musi przygotować pewne informacje dla systemu. W tym celu używa on następujących rozkazów:

CCB — utworzyć blok sterowania danymi. Blok sterowania danymi (CCB) przeznaczony jest do przekazywania SUPERVISOROWI informacji koniecznej dla wykonania operacji we/wy. Z drugiej strony, blok CCB używany jest przez SUPERVISOR dla przekazania programowi problemowemu informacji o wynikach wykonanej operacji. Informacja przekazywana SUPERVISOROWI to kod urządzenia logicznego, które realizuje operację we/wy, adres programu kanałowego oraz wskazanie sposobu reagowania na nienormalne sytuacje;

EXCP — wykonać program kanałowy. Instrukcja ta powoduje zainicjowanie operacji we/wy. Zawiera ona w sobie rozkaz SVC;

WAIT — czekać. Instrukcja ta powoduje wstrzymanie przetwarzania programu problemowego do czasu zakończenia operacji we/wy.

Gdy programista sam pisze program kanałowy i używa makroinstrukcji **CCB, EXCP, WAIT**, wówczas mówimy o fizycznym poziomie programowania we/wy. Ten sposób programowania przedstawiamy w rozdziale VI.

Prowadzenie kolejek zgłoszeń żądań dostępu do urządzeń we/wy. Po zgłoszeniu przez program żądania wykonania operacji we/wy SUPER-

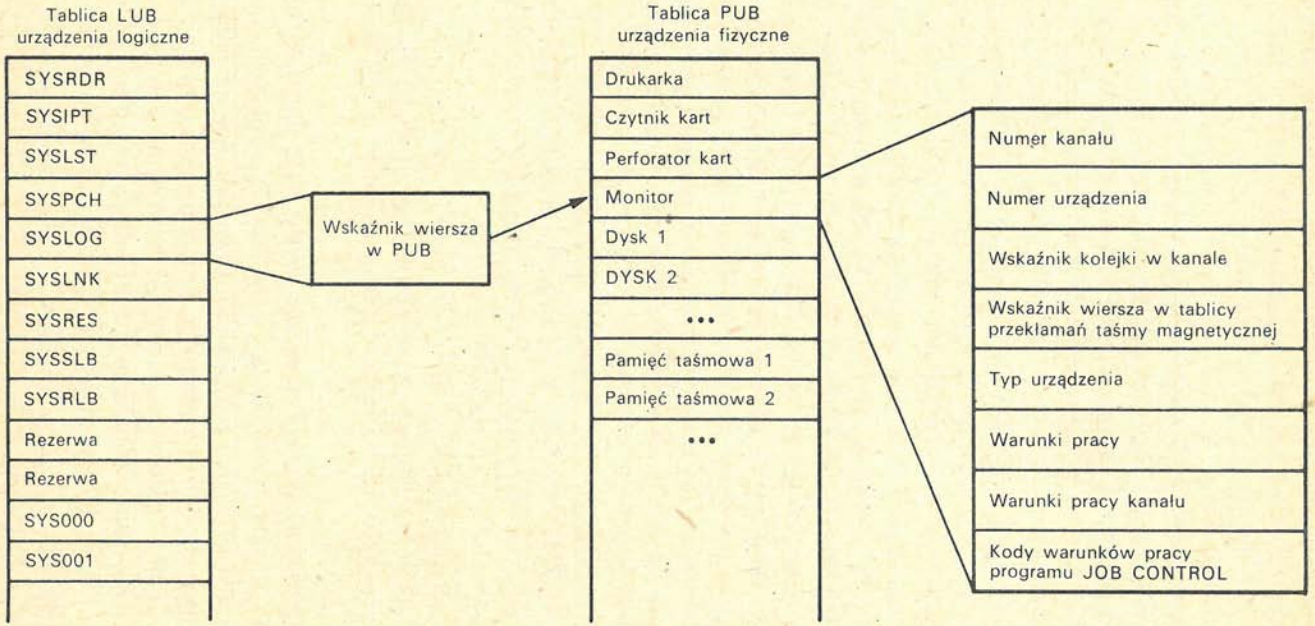
VISOR pobiera informację z bloku CCB wskazującą, na jakim urządzeniu logicznym ma być wykonana ta operacja. Zadaniem SUPERVISORA jest teraz rozstrzygnięcie, jakie urządzenie fizyczne odpowiada danemu urządzeniu logicznemu. W celu wykonania tego zadania SUPERVISOR prowadzi dwie tablice określające związki między fizycznymi a logicznymi urządzeniami: tablice urządzeń logicznych (LUB) i urządzeń fizycznych (PUB). Na podstawie tych tablic SUPERVISOR określa adres fizycznego urządzenia, które ma przeprowadzić operację we/wy i jeśli w danej chwili jest ono niedostępne, tworzy kolejkę do tego urządzenia (por. rys. 34).

Inicjacja operacji we/wy. Po otrzymaniu zgłoszenia żądania dostępu do urządzenia we/wy SUPERVISOR analizuje wiersz w tabeli PUB, opisujący charakterystyki i stan urządzenia. Jeżeli urządzenie jest wolne, SUPERVISOR wykorzystując zawarty w bloku CCB adres programu kanałowego, tworzy słowo adresu kanału (CAW) i wydaje rozkaz **SIO** (rozpocząć operację we/wy). W wypadku normalnego rozpoczęcia operacji sterowanie zostaje przekazane programowi problemowemu i wykonywanie jego zostaje wznowione. Jeżeli programista uważa, że kontynuacja programu problemowego w danym momencie jest niedopuszczalna (przed wykonaniem operacji we/wy), powinien w odpowiednim miejscu wstawić makroinstrukcję **WAIT.**

W razie nieudanej próby zainicjowania operacji we/wy (gdy instrukcja **SIO** spowodowała ustawienie kodu warunku różnego od zera), sterowanie zostaje przekazane programowi obsługi przerwań we/wy analizującemu przyczyny nieudanego rozpoczęcia operacji.

W tych wypadkach, gdy żądanie wykonania operacji we/wy nie może być natychmiast zrealizowane z powodu zajętego urządzenia, żądana operacja będzie wykonana po realizacji zgłoszeń już wcześniej postawionych.

Obsługa przerwań we/wy. Przerwania we/wy mogą pojawić się z powodu zwolnienia kanału, zwolnienia urządzenia sterującego, zwolnienia urządzenia zewnętrznego, przyciśnięcia przycisku AR na pulpicie operatorskim maszyny lub na żądanie programu kanałowego. Przyczyny przerwania i stan urządzeń w momencie przerwania opisuje słowo stanu kanału (CSW), które tworzone jest układowo w chwili występowania przerwania we/wy lub wówczas, gdy próba rozpoczęcia operacji we/wy nie powiodła się.



Rys. 34. Struktura tablic LUB i PUB

Obsługa przerwania polega na analizie informacji zawartej w CSW przez odpowiedni podprogram SUPERVISORA. Przypatrzmy się bliżej przyczynom wywołującym przerwanie we/wy.

1. Sygnał *Kanał zakończył*. Jeśli nie towarzyszy mu sygnalizacja świadcząca o sytuacji awaryjnej, traktowany jest przez SUPERVISOR jako normalne zakończenie operacji we/wy. W takim wypadku SUPERVISOR zapisuje w CCB informację o wykonaniu operacji i usuwa z kolejki zgłoszenie żądania wykonania tej operacji. Jeżeli operacja nie zakończyła się normalnie, sterowanie zostaje przekazane odpowiedniemu programowi obsługi przerwań.

2. Przerwanie, spowodowane sygnałem *Jednostka sterująca zakończyła* jest przez SUPERVISOR ignorowane, o ile towarzyszy mu sygnalizacja o sytuacji awaryjnej. W tym ostatnim wypadku sterowanie przejmuje podprogram obsługi sytuacji wyjątkowych.

3. Sygnał *Urządzenie zakończyło* umożliwia rozpoczęcie kolejnej operacji we/wy na danym urządzeniu.

4. Przerwanie spowodowane przez sygnał AR jest obsługiwane przez SUPERVISOR tylko w tym wypadku, jeśli sygnał ten przekazany jest przez operatora z urządzenia logicznego SYSLOG. Obsługa tego przerwania polega na przekazaniu sterowania specjalnego podprogramowi AR, którego zadaniem jest przyjmowanie poleceń operatora.

5. Przerwania wywołane programowo są wykorzystywane w teleprzetwarzaniu i mogą być przyjmowane tylko od urządzeń transmisji danych.

Kontrola współpracy z pamięcią dyskową

Pamięć dyskową można przydzielić urządzeniom logicznym systemowym: SYSRDR, SYSIPT, SYSPCH i SYSLST. Dopuszczalny jest w tych wypadkach tylko jeden sposób organizacji danych — organizacja sekwencyjna. SUPERVISOR kontroluje adresy danych w kolejnych dostępach do pamięci dyskowej i sygnalizuje naruszenie sekwencyjności adresów.

SUPERVISOR może również organizować ochronę zbiorów danych zapisanych w obszarach pamięci dyskowych.

Obie te funkcje, to jest kontrola urządzeń logicznych na dyskach i ochrona zbiorów w pamięci dyskowej należy przewidzieć podczas generacji systemu operacyjnego.

d. Obsługa przekłamań urządzeń zewnętrznych

Sytuacje, w których następuje:
— nieprawidłowa długość danych,
— sytuacja wyjątkowa w urządzeniu zewnętrznym,
— przekłamanie w kanale lub urządzeniu zewnętrznym
sygnalizowane są w CSW. Pierwsze dwie traktuje SUPERVISOR jako normalne. Odpowiednia informacja o tych sytuacjach zostaje wprowadzona do CCB, a dalsze działanie powinien przewidzieć sam użytkownik.

Przekłamania pojawiające się w operacjach we/wy są analizowane przez SUPERVISOR. Podjęte przez niego działanie zależy od rodzaju przekłamania. Przekłamania w urządzeniach kanału i *interface* powodują przejście maszyny w stan CZEKAJ, z którego wyjście możliwe jest przez ponowne początkowe ładowanie systemu. Przekłamania spowodowane błędami programowymi i naruszeniem ochrony pamięci prowadzą do skasowania wykonywanego programu problemowego.

Reakcją SUPERVISORA na przekłamania może być próba uwolnienia się od nich automatycznie lub za pomocą operatora, ignorowanie przekłamania, przekazanie sterowania programowi problemowemu w celu podjęcia przez niego określonego działania lub skasowanie zadania.

e. Systemowy program załadowczy

Programy bezpośrednio przed wykonaniem znajdują się w bibliotece faz, znajdującej się w pamięci dyskowej. System operacyjny pobiera stąd programy i przenosi je do pamięci głównej. Procedurę tę realizuje podprogram SUPERVISORA, zwany systemowym programem załadowczym. Umieszczenie programów w bibliotece faz (katalogowanie programów) odbywa się w czasie pracy programu REDAKTOR. Program w bibliotece może być przechowywany w postaci jednej lub kilku faz programowych. Każdej fazie towarzyszy pewna informacja pomocnicza, jak nazwa fazy, adres pamięci, od którego program powinien być umieszczony w pamięci głównej, adres startu, określający punkt wejścia do programu.

Wszystkie programy przeznaczone do wykonania zostają wprowadzone do pamięci głównej za pomocą systemowego programu załadowczego (jedynym wyjątkiem jest ładowanie jądra SUPERVISORA).

Ładowanie programu odbywa się na specjalne żądanie. Dla załadowania pierwszej fazy żądanie takie wydaje JOB CONTROL. Następne fazy programu wielofazowego muszą być wywoływane przez program pro-

blemowy. Istnieją dwa sposoby wezwania faz — za pomocą makroinstrukcji **FETCH** lub **LOAD**:

— Makroinstrukcja **FETCH** powoduje pobranie przez systemowy program załadowczy wskazanej fazy z biblioteki faz i umieszczenie jej w pamięci głównej pod adresem określonym w fazie. Następnie SUPERVISOR przekazuje sterowanie załadowanej fazie do wskazanego w fazie adresu startu. Jeśli w makroinstrukcji **FETCH** adres startu nie jest podany, sterowanie przekazywane jest do adresu określonego w czasie redagowania tej fazy.

— Makroinstrukcja **LOAD** wywołuje podobną procedurę ładowania. Jedyną różnicą jest sposób przekazania sterowania po załadowaniu. W tym wypadku SUPERVISOR przekazuje sterowanie do programu problemowego do rozkazu bezpośrednio następującego po instrukcji **LOAD.**

f. Zakończenie wykonywania zadania

O końcu zadania informuje system operator /&, który jest ostatnim operatorem zadania. Jeśli nie pojawiła się sytuacja uniemożliwiająca całkowite wykonanie zadania, mówimy, że zadanie zostało zakończone normalnie. W przeciwnym wypadku uważamy, że zadanie zostało zakończone nienormalnie. Przyczyny nienormalnego zakończenia programu mogą być następujące:

— przekłamanie programowe,
— przekłamanie w systemie we/wy,
— żądanie operatora skasowania programu,
— żądanie zakończenia programu postawione przez program problemowy.

Po normalnym zakończeniu zadania SUPERVISOR uruchamia program JOB CONTROL, który przygotowuje następne zadanie do wykonania.

Po nienormalnym zakończeniu kroku zadania następuje przerwanie, zadanie zostaje skasowane i pozostałe jego kroki nie są wykonywane. System wydaje przy tym komunikat wyjaśniający przyczynę skasowania zadania. SUPERVISOR może w tym wypadku wydrukować zawartość pamięci głównej na urządzeniu SYSLST, jeśli podany jest operator **DUMP.** Następnie do pamięci głównej wzywany jest program JOB CONTROL, który przygotowuje następne zadanie do wykonania.

Jak już mówiliśmy, skasowanie zadania może być wywołane żądaniem zawartym w programie problemowym w postaci makroinstrukcji **CANCEL**

lub **DUMP**. Po makroinstrukcji **DUMP** następuje zawsze wydruk zawartości pamięci głównej, natomiast **CANCEL** wydruku nie powoduje.

Zakończenie programu w rozdziale pierwszoplanowym, który pracuje w trybie wykonywania pojedynczych programów, prowadzi do zwolnienia rozdziału. Drukuje się przy tym komunikat dla operatora.

g. Organizacja łączności z operatorem

Sterowanie pracą maszyny wymaga wymiany informacji między operatorem[3] a systemem operacyjnym. Wymiana ta jest dwustronna i odbywa się przez urządzenie logiczne SYSLOG. Operator nawiązuje łączność z programami SUPERVISORA lub też z innymi programami, np. z programem JOB CONTROL.

Przekazywanie informacji od systemu do operatora EMC. Komunikaty wydawane przez system operacyjny noszą charakter informacyjny lub są poleceniami, żądającymi od operatora określonego działania. Wszystkie komunikaty drukowane są na urządzeniu SYSLOG i każdy składa się z trzech części:

— kod komunikatu (cztery znaki),
— wskaźnik reakcji operatora (jeden znak),
— tekst komunikatu (dowolna długość).

Pierwszy znak kodu komunikatu określa program, który wydał komunikat:

0 — SUPERVISOR,
1 — JOB CONTROL,
2 — REDAKTOR,
3 — BIBLIOTEKARZ,
4 — Logiczny SSWW,
5 — PL/I,
6 — RPG,
7 — program sortowania,
8 — programy serwisowe pomocnicze,
9 — AUTOTEST,
A — ASSEMBLER,
B — FORTRAN.

Trzy następne znaki są numerem komunikatu.

Wskaźnik reakcji operatora określa rodzaj czynności, jaką powinien operator wykonać:

[3] Por. *DOS/JS Podręcznik operatora; DOS/JS Komunikaty operatorskie.*

A — działanie — żądanie od operatora wykonania konkretnego działania,

D — decyzja — stawia operatora przed koniecznością wyboru jednego z kilku możliwych działań,

I — informacja — komunikat jest często informacyjny i nie wymaga interwencji operatora,

W lub S — wymaga od operatora wykonania procedury początkowego ładowania systemu. Komunikat z takim wskaźnikiem w odróżnieniu od innych komunikatów nie jest drukowany na SYSLOG, lecz zapisuje się w bajty 0 i 1 pamięci głównej. Komunikaty te mogą być wydane przez SUPERVISOR jako wynik przekłamania, uniemożliwiającego dalsze funkcjonowanie systemu.

Jeżeli system pracuje wieloprogramowo, to przed każdym komunikatem drukuje się dodatkowo symbole BG, F2, F1, które wskazują jakiego rozdziału, a więc i programu dotyczy komunikat.

Przekazywanie informacji od operatora do systemu

Operator przekazuje systemowi informację w postaci dyrektyw lub w postaci odpowiedzi na komunikaty systemu.Dyrektywy można podzielić na grupy w zależności od tego, dla jakiego programu są przeznaczone:

1) dyrektywy programu ŻĄDANIE KOMUNIKACJI (AR),
2) dyrektywy programu INICJATOR POJEDYNCZYCH PROGRAMÓW (SPI),
3) dyrektywy programu sterującego zadaniami (JOB CONTROL).

Program AR składa się z tranzytowych faz SUPERVISORA. Łączność z nim jest nawiązywana przez wciśnięcie specjalnego przycisku AR na pulpicie sterującym komputera. Program AR, sprowadzony do pamięci głównej, wydaje komunikat:

READY FOR COMMUNICATIONS

Od tej chwili mogą być wprowadzane dyrektywy tego programu za pośrednictwem urządzenia SYSLOG.

Łączność z programem INICJATOR POJEDYNCZYCH PROGRAMÓW nawiązuje się przez użycie dyrektywy **START**.

Łączność operatora z programem JOB CONTROL nawiązuje się automatycznie po zakończeniu procedury początkowego ładowania systemu.

Później, w toku pracy systemu, kontakt z programem JOB CONTROL może być nawiązany przez podanie dyrektywy **PAUSE**, obsługiwanej przez program JOB CONTROL i AR, dyrektywy **BATCH** lub **START**, obsługiwane przez program AR i wreszcie automatycznie — przez osiągnięcie końca partii zadań na SYSRDR.

Istnieje możliwość wprowadzenia dyrektyw programu JOB CONTROL razem z zadaniem wczytywanym z SYSRDR.

Operator systemu DOS może stosować następujące dyrektywy dla programu AR:

ALLOC — rozdzielić pamięć główną na rozdziały pierwszoplanowe i drugoplanowe,
BATCH — rozpocząć lub kontynuować przetwarzanie wsadowe w jednym z rozdziałów pierwszoplanowych,
CANCEL — skasować zadanie,
LOG — drukować operatory sterujące lub dyrektywy na urządzeniu SYSLOG,
MAP — drukować tabelę podziału pamięci głównej,
MSG — przekazać sterowanie podprogramowi łączności z operatorem w czasie wykonywania programu pierwszoplanowego,
NOLOG — odwołać dyrektywę LOG,
PAUSE — zorganizować przerwę,
START — rozpocząć przetwarzanie w rozdziale pierwszoplanowym,
TIMER — przedzielić zegar wskazanemu rozdziałowi,
KT — koniec łączności (przycisk na pulpicie).

h. Obsługa zegara

SUPERVISOR, działając na podstawie zegara czasu rzeczywistego, stwarza programiście możliwości:

— określa aktualny czas dnia,
— wyznacza pewny odcinek czasu, po którym powinno nastąpić przerwanie programu i przekazanie sterowania do wskazanego adresu w programie problemowym,
— czasowo wstrzymuje wykonanie zadania, a po wyznaczonym okresie następuje kontynuacja programu.

W sytuacjach tych można używać następujących makroinstrukcji:

GETIME — podaj aktualny czas dnia.

Możliwe są trzy warianty: w postaci godziny-minuty-se-

kundy, w postaci całkowitej liczby binarnej wyrażającej ilość sekund lub w postaci liczby całkowitej określającej ilość jednostek 1/300 sekundy. Początkową wartość zegara ustala się w czasie procedury ładowania początkowego systemu,

SETIME — ustalić czas, po którym ma nastąpić przerwanie od zegara,

STXIT — nawiązać łączność z własnym podprogramem obsługi przerwań wywołanych zegarem,

EXIT — przekazać sterowanie po wykonaniu programu obsługi przerwania (do punktu, którym nastąpiło przerwanie),

TECB — utworzyć blok komunikacji z zegarem (wykorzystywany jest dla powiadomienia programu problemowego o przerwaniu od zegara),

WAIT — czekać (program problemowy informuje SUPERVISOR o konieczności czekania do wystąpienia przerwania wywołanego zegarem).

Zegar może pracować tylko w jednym rozdziale pamięci, wyznaczonym w czasie generacji systemu lub później, za pomocą dyrektywy **TIMER**. Zegar zlicza czas, jaki rzeczywiście upływa, a nie, np. czas wykonania zadania w rozdziale, do którego jest przypisany.

i. Organizacja punktów restartu

Podczas wykonywania długich programów istnieje niebezpieczeństwo, że jakaś nie przewidziana przyczyna przeszkodzi w zakończeniu danego zadania i konieczne będzie powtórne przetwarzanie programu od początku. Zabezpieczenie przed wynikającymi stąd dużymi stratami czasu można osiągnąć przez stosowanie tzw. punktów restartu, które umożliwiają podjęcie pracy od miejsca wskazanego przez programistę. We wskazanym miejscu programu system organizuje punkt kontrolny, tzn. zapamiętuje informację o stanie programu i systemu. Informacje te muszą być na tyle wyczerpujące, aby można było od danego punktu kontynuować przetwarzanie programu. O tym, gdzie zapisuje system te informacje decyduje programista, używając dowolnego urządzenia logicznego programisty SYSxxx, któremu przydzielono taśmę magnetyczną lub dysk. Opis punktu restartu obejmuje zawartość pamięci głównej oraz położenie taśm magnetycznych. Nie zapamiętuje się informacji o stanie innych urządzeń.

Organizacja punktu restartu następuje po wydaniu instrukcji **CHKPT**.

Punkty kontrolne mogą być organizowane dla programu w dowolnym rozdziale, pracującym w trybie przetwarzania wsadowego.

Rozpoczęcie przetwarzania od punktu kontrolnego powoduje operator RSTRT, który ukazuje również numer punktu restartu.

j. Tablice SUPERVISORA

SUPERVISOR wykonuje swoje funkcje na podstawie informacji zawartej w specjalnych tablicach. Są to takie tablice, jak tablica urządzeń fizycznych, tablica urządzeń logicznych, tablica kolejek do urządzeń zewnętrznych itp. Do jednej z tablic ma dostęp również użytkownik. Jest to *obszar komunikacji*. Zawartość informacji w tym obszarze przedstawia tablica 13.

TABLICA 13
Zawartość obszaru łączności

Bajty	Informacja
0 — 7	Data kalendarzowa.
8 — 9	Adres początku obszaru programów problemowych.
10—11	Adres początku ochrony pamięci kluczem równym 1.
12—22	Obszar użytkownika (dla łączności wewnątrz programów i między programami).
23	Przełączniki programowe (UPSI).
24—31	Nazwa zadania (wpisuje program JOB CONTROL).
36—39	Adres końca fazy (ostatni adres ostatnio wezwanej dla wykonania fazy).
40—43	Adres końca fazy (największy z adresów końca faz programu).
44—45	Długość obszaru etykiet programu problemowego.

Obszar komunikacji znajduje się w przedziale pamięci głównej, przeznaczonym dla programu sterującego. Obszary komunikacji prowadzi się

tylko dla tych rozdziałów, w których możliwe jest przetwarzanie wsadowe. Każdy taki rozdział ma wówczas swój obszar komunikacji.

Dostęp do obszaru komunikacji umożliwiają makroinstrukcje **COMRG** i **MVCOM:**

COMRG — ustalenie adresu obszaru komunikacji (adres ten umieszcza SUPERVISOR w rejestrze uniwersalnym 1),

MVCOM — zapisanie danych w obszarze komunikacji.

Przez podanie makroinstrukcji **COMRG** użytkownik uzyskuje w rejestrze uniwersalnym 1 adres obszaru komunikacji. Na podstawie tego adresu można odczytać dowolne pola z obszaru komunikacji lub zapisywać w nim dane w polach użytkownika (bajty 12-22) i w bajcie przełączników programowych (bajt 23).

7. JOB CONTROL

Program JOB CONTROL[4] steruje wykonywaniem programów w trybie przetwarzania wsadowego. Praca jego polega na przyjęciu strumienia wejściowego zadań i każdorazowym przygotowaniu systemu do wykonania określonego zadania. Program JOB CONTROL znajduje się w bibliotece faz i stamtąd jest wzywany przez SUPERVISOR do obszaru programów problemowych. Dzieje się tak w następujących wypadkach:

— po normalnym zakończeniu zadania lub kroku zadania,

— po zakończeniu obsługi nienormalnie zakończonego zadania,

— wskutek żądania operatora rozpoczęcia przetwarzania wsadowego w rozdziale pierwszoplanowym,

— po procedurze ładowania początkowego.

Zwykle praca programu JOB CONTROL kończy się przekazaniem sterowania SUPERVISOROWI, który realizuje przesłanie żądanej fazy programu z bibliotek faz do odpowiedniego rozdziału pamięci operacyjnej, przy czym faza ta zajmuje to samo miejsce pamięci, które do tej chwili zajmował JOB CONTROL. Tym samym program JOB CONTROL zostaje w danym rozdziale skasowany.

Wykonując swoje funkcje JOB CONTROL korzysta z informacji dostarczonej przez programistę w postaci operatorów sterujących oraz z informacji przekazanej przez operatora w postaci dyrektyw. Oprócz tego JOB CONTROL korzysta z tablic SUPERVISORA, do których może też wnieść nowe dane.

[4] Por. *DOS/JS Programy sterujące.*

a. Przygotowanie programu do wykonania

Wszystkie programy przed wykonaniem są pobierane do pamięci głównej z biblioteki faz. Programista podaje w operatorze sterującym nazwę fazy, która ma być wykonana. Przygotowanie programu do wykonania polega na tym, że system przegląda spis faz znajdujących się aktualnie w bibliotece faz i wybiera stamtąd adresy żądanych w zadaniu faz i umieszcza je w specjalnej tablicy. Z tablicy tej będzie następnie korzystał SUPERVISOR przy wyszukiwaniu żądanej fazy w toku wykonywania programu.

Po utworzeniu tablicy zawierającej spis faz, JOB CONTROL zeruje dany rozdział pamięci (wpisuje zera), z wyjątkiem pierwszych 150 bajtów. Następnie sterowanie przyjmuje SUPERVISOR, który sprowadza pierwszą fazę programu do pamięci głównej.

Następne fazy kroku zadania, jeżeli składa się ono z więcej niż jednej fazy, ładuje do pamięci głównej również SUPERVISOR, lecz odbywa się to na żądanie programu problemowego i bez udziału programu JOB CONTROL.

b. Przydział urządzeń fizycznych urządzeniom logicznym

Już mówiliśmy, że niezależność programów od konkretnych fizycznych adresów zapewnia system urządzeń logicznych. Urządzeniom tym przydziela się urządzenia fizyczne. Przydziały mogą być standardowe, czasowe i stałe. Przydziały standardowe są dokonywane podczas generacji systemu i dotyczą tylko rozdziału drugoplanowego. Przydziały te są zawsze aktualne od momentu początkowego ładowania systemu do czasu ich zmiany przez przydział stały. Przy następnej procedurze ładowania początkowego znowu aktualne są wszystkie przydziały standardowe.

Przydziały stałe są ustalone przez operatora i działają do czasu ich odwołania przez inny przydział lub przez początkowe ładowanie systemu. Przydział stały następuje za pomocą dyrektywy

ASSGN SYS*xxx*, X'*cnn*',

gdzie SYS*xxx* — urządzenie logiczne, X'*cnn*' — adres urządzenia zewnętrznego, *c* — numer kanału, *nn* — numer urządzenia.

Przydział czasowy ustala zależność między urządzeniami logicznymi a fizycznymi tylko na okres wykonywania jednego zadania, po czym znowu

są w mocy poprzednie ustalenia (przydziały stałe lub standardowe). Przydziałów czasowych dokonuje się za pomocą operatora sterującego:

/ /ASSGN SYS*xxx*, X' *cnn*',

w którym znaczenie parametrów jest takie same jak w przydziałach stałych.

Program JOB CONTROL na podstawie informacji dostarczonej w operatorach sterujących lub dyrektywach dokonuje żądanych przydziałów urządzeń.

c. Ustalenie warunków pracy systemu

Dla ustalenia warunków pracy systemu podczas przetwarzania każdego zadania używa się operatora sterującego OPTION[5]:

```
                [ LINK  ][ LIST   ][ SYM   ][ DECK   ][ XREF   ]
// OPTION       [       ][ ,      ][ ,     ][ ,      ][ ,      ]
                [ CATAL ][ NOLIST ][ NOSYM ][ NODECK ][ NOXREF ]

[ 48C ] [ DUMP   ][ STDLABEL ][ LISTX   ] [ ERRS   ]
[ ,   ] [ ,      ][ ,        ][ ,       ] [ ,      ]
[ 60C ] [ NODUMP ][ USRLABEL ][ NOLISTX ] [ NOERRS ]
                  [ PARSTD   ]
```

gdzie:

LINK — ustala tryb pracy systemu, w którym natychmiast po redagowaniu (łączeniu) program zostaje umieszczany czasowo w bibliotece faz i natychmiast wykonywany,

CATAL — oznacza, że program będzie redagowany, a następnie zapisany w bibliotece faz na stałe,

---

[5] W opisach składni języków programowania i operatorów systemu operacyjnego będziemy stosowali kilka poniższych prostych reguł, które umożliwią krótkie i jednoznaczne wyjaśnienie dopuszczalnych konstrukcji językowych.

Do opisu elementów lub zdań języka używać będziemy identyfikatorów metajęzyka, którymi są dowolne ciągi małych liter alfabetu polskiego, cyfr i znaku myślnika, zaczynające się od litery, np. [*cyfra*, *nazwa-kartoteki*, *liczba-calkowita-bez-znaku*.

Są to zmienne składniowe. Natomiast stałe składniowe mogą składać się tylko z dużych łacińskich liter lub niektórych znaków specjalnych, traktowanych jak litery.

Używać będziemy nawiasów kwadratowych i klamrowych, które nie wchodzą w skład żadnego z alfabetów przedstawianych tu języków, np.

```
{ FIXUNB }
{ VARUNB }
```

Zapis ten oznacza, że jedna z wymienionych nazw musi zostać użyta. To samo można zapisać w inny sposób:

{FIXUNB|VARUNB}

LIST — translator drukuje moduł źródłowy na urządzeniu SYSLST,

SYM — parametr ten powoduje wydruk tablicy wszystkich nazw symbolicznych użytych w programie,

ERRS — podanie tego parametru powoduje wydruk wszystkich wykrytych przez translator błędów w module źródłowym,

DECK — jeśli parametr ten jest podany, wówczas translator wyprowadza moduł wynikowy na SYSPCH,

LISTX — jeżeli moduł źródłowy napisany w języku PL/1, wówczas translator wydrukuje go w postaci assemblerowej,

XREF — parametr ten powoduje wydruk tablicy nazw, jeśli program napisany jest w języku ASSEMBLER,

DUMP — w wypadku nienormalnego zakończenia programu zostanie wydrukowana na SYSLST zawartość pamięci głównej,

48C — informacja dla translatora PL/1 o użyciu skróconego 48-znakowego alfabetu,

60C — świadczy o użyciu pełnego alfabetu języka PL/1,

STADLABEL

USRLABEL

PARSTD — operatory powodujące wpisanie informacji o etykietach kartotek systemowych do specjalnych obszarów pamięci znajdujących się w cylindrze etykiet na pakiecie rezydencyjnym.

Przedrostki **NO** odwołują odpowiednie parametry.

Pewne parametry są ustalone w czasie generacji systemu i można ich nie podawać w operatorze **OPTION**. Zwykle, jako standardowe, przyjmuje się następujące parametry:

//OPTION LIST, NOSYM, ERRS, NODECK, DUMP, NOLISTX, 60C.

Podkreślenie oznacza, że dany parametr zadany jest w systemie standardowo i można go w programie nie wyszczególniać:

{STREAM|RECORD}

W przykładzie tym, jeśli programista opuści w swoim opisie obie nazwy, system będzie się tak zachowywał, jak gdyby zadany został parametr STREAM.

Elementy opcjonalne języka ujęte są w nawiasy kwadratowe, np.

GET *nawias-kartoteki* [*nazwa-obszaru-roboczego*].

W przykładzie tym nazwa kartoteki musi koniecznie wystąpić w instrukcji, natomiast nazwa obszaru roboczego nie jest obowiązkowa.

Trzy kropki oznaczają wielokrotne powtórzenie elementu występującego bezpośrednio przed nimi Na przykład zapis:

OPEN *nazwa-kartoteki*, [*nazwa-kartoteki*]...

oznacza, że instrukcji OPEN można użyć dla wielu kartotek, ale co najmniej dla jednej.

Tak więc, w operatorze **OPTION** wystarczy podać tylko te parametry, które chcemy zmienić w stosunku do standardowych.

d. Przetwarzanie etykiet kartotek

Program JOB CONTROL pobiera informację o etykietach z operatorów sterujących (**TLBL, DLBL, EXTENT**), redaguje ją nadając jej postać zbliżoną do formatu etykiet i umieszcza na pakiecie rezydencyjnym systemu. Informacja ta zajmuje jeden cylinder: każdy rozdział pamięci głównej ma przydzieloną jedną ścieżkę dla informacji czasowej i jedną dla informacji stałej. Informację tę wykorzystują programy systemu sterowania we/wy przy otwieraniu kartotek (dla kontroli kartotek wejściowych) lub zamykaniu kartotek (dla tworzenia etykiet kartotek wyjściowych).

Szerzej o etykietach piszemy w rozdziale VIII.

e. Zapis informacji do obszaru komunikacji

Program JOB CONTROL przenosi z operatorów sterujących do obszaru komunikacji następującą informację:

— nazwę zadania, pobieraną z operatora **JOB,**

— datę, zadaną w operatorze **DATE.** Jeśli nie było tego operatora, system przyjmuje datę podaną w operatorze **SET,** podaną podczas początkowego ładowania systemu,

— przełączniki programowe, pobierane z operatora **UPSI**; każdy z tych przełączników może być wykorzystany w programie problemowym.

f. Przygotowanie restartu

JOB CONTROL przygotowuje system do rozpoczęcia przetwarzania programu do punktu restartu. W tym celu przywołuje podprogram SUPERVISOR, który ustawia we właściwych miejscach taśmy magnetyczne, odtwarza obszar komunikacji i inicjuje rozpoczęcie przetwarzania.

8. Inicjator pojedynczych programów

Jeżeli w rozdziale pierwszoplanowym ma być wykonane pojedyncze zadanie, wówczas przygotowania tego zadania i zainicjowanie jego wykonania dokonuje się za pomocą programu INICJATOR POJEDYNCZYCH PROGRAMÓW (ISP).

Przed wykonaniem pojedynczego programu w rozdziale pierwszoplanowym operator ze swego pulpitu (SYSLOG) wywołuje program INICJATOR POJEDYNCZYCH PROGRAMÓW za pomocą dyrektywy **START**. Dyrektywa ta, przeznaczona dla programu AR, powoduje sprawdzenie czy dany rozdział jest wolny i czy przydzielono mu pamięć. Jeżeli tak jest, wtedy INICJATOR POJEDYNCZYCH PROGRAMÓW zostaje wezwany z biblioteki faz, wpisany do tego rozdziału i zaczyna się w nim wykonywać. Teraz operator może wprowadzić dyrektywy niezbędne dla tego rodzaju przetwarzania.

Funkcje realizowane INICJATOREM POJEDYNCZYCH PROGRAMÓW pokrywają się w dużym stopniu z zadaniami wykonywanymi przez program JOB CONTROL. Po wykonaniu programu rozdział, w którym program ten był przetwarzany, zostaje zwolniony, a następny program może być wykonywany tylko na żądanie operatora.

9. Ładowanie początkowe

Pracę na maszynie cyfrowej z użyciem systemu operacyjnego DOS można rozpocząć tylko wówczas, jeżeli system został przygotowany do przetwarzania. Osiąga się to przez przeprowadzenie procedury ładowania początkowego, która polega na wykonaniu przez operatora następujących działań pomocniczych:

— umieszczenie na jednostce pamięci dyskowej, wybranej na rezydencję systemu, odpowiedniego pakietu systemowego,

— ustawienie adresu urządzenia z pakietem systemowym na pulpicie maszyny za pomocą odpowiednich przełączników,

— naciśnięcie przycisku ŁADUJ.

Po tej ostatniej czynności z pakietu systemowego zostanie wprowadzony do pamięci operacyjnej specjalny program ŁADOWANIE POCZĄTKOWE (IPL), który wykona następujące czynności:

— wyzeruje pamięć główną i ustawi klucze ochrony pamięci,

— wprowadzi do pamięci głównej jądro SUPERVISORA, znajdujące się na dysku systemowym,

— skoryguje tablice przydziałów urządzeń, zawarte w jądrze SUPERVISORA (wprowadza ewentualne zmiany w stosunku do ustaleń poczynionych w czasie generacji systemu),

— zainicjuje przetwarzanie wsadowe w rozdziale drugoplanowym.

Pracę programu ŁADOWANIE POCZĄTKOWE finalizuje dyrektywa **SET**, w której operator podaje datę i aktualną porę dnia. Ma ona następującą postać:

SET DATE = *tt/dd/rr* [, **CLOCK** = *gg/mm/ss*]

gdzie:

tt — miesiąc,
dd — dzień miesiąca,
rr — rok,
gg — godzina,
mm — minuta,
ss — sekundy.

Parametr **CLOCK** występuje tylko wówczas, gdy w czasie produkcji systemu przewidziano używanie zegara.

10. Dyrektywy i operatory programu sterującego

W tablicy 14 zestawiono wszystkie dyrektywy i operatory programu sterującego. Dla każdej dyrektywy i operatora podano, w jakich programach są one używane. Wszystkie operatory programu JOB CONTROL rozpoczynają się dwiema skośnymi kreskami (//), po których musi wystąpić

TABLICA 14
Dyrektywy i operatory programu sterującego

Funkcja	JOB CONTROL		INICJATOR POJEDYNCZYCH PROGRAMÓW	AR	ŁADOWANIE POCZĄTKOWE
	operatory	dyrektywy	dyrektywy	dyrektywy	dyrektywy
Identyfikowanie zadania	// JOB /&		/&		

Opisanie kartotek	// DLBL
	// EXTENT
	// TLBL
	/*
			[//] DLBL
			[//]
			EXTENT
			[//] TLBL
			/*		

TABLICA 14 (cd.)

Funkcja	JOB CONTROL		INICJATOR POJEDYNCZYCH PROGRAMÓW	AR	ŁADOWANIE POCZĄTKOWE
	operatory	dyrektywy	dyrektywy	dyrektywy	dyrektywy
Informacje dla operatora	*				

Informacje dla programu	// DATE
	// OPTION
	// LBLTYP
	// UPSI
			// LBLTYP		

Sterowaie strumieniem zadań	// PAUSE	PAUSE CANCEL STOP UNBATCH	PAUSE CANCEL READ	PAUSE BATCH CANCEL START	
Ustawienie parametrów systemowych		ALLOC MAP SET	MAP TIMER	ALLOC MAP TIMER	SET

Łączność z operatorem	// LOG
	// NOLOG
	KT
		LOG
		NOLOG
		KT
			LOG
			MSG
			NOLOG
			KT
				LOG
				MSG
				NOLOG
				KT	

Sterowanie systemem we/wy	// ASSGN
	// CLOSE
	// LISTIO
	// MTC
	// RESET
		ASSGN
		CLOSE
		DVCDN
		DVCUP
		HOLD
		LISTIO
		MTC
		RELSE
		RESET
		UCS
		UNA
			ASSGN
			LISTIO
			RELSE
			UCS
			UNA
					ADD

Przygotowanie do wykonania	// EXEC // RSTRT		[//] EXEC		

przynajmniej jedna spacja. Następnie pisze się kod operatora sterującego, którym jest po prostu jego nazwa, a następnie parametry rozdzielone przecinkami. Dyrektywy pisze się w taki sam sposób, lecz bez znaków // (z wyjątkiem niektórych dyrektyw programu INICJATOR POJEDYNCZYCH PROGRAMÓW).

Operatory sterujące przygotowuje się zwykle na kartach i wprowadza z urządzenia SYSRDR. Jeśli jednak istnieje taka potrzeba, można wprowadzać je z SYSLOG. Dyrektywy programów JOB CONTROL i INICJATOR POJEDYNCZYCH PROGRAMÓW wprowadza się zwykle z urządzenia SYSLOG, natomiast dyrektywy przeznaczone dla programu AR — wyłącznie z SYSLOG.

11. REDAKTOR

Moduły wynikowe, otrzymane w rezultacie translacji nie mogą być natychmiast przetwarzane, ponieważ nie mają jeszcze odpowiedniej postaci do tego celu. Oprócz zasadniczego tekstu programu zawierają pewną dodatkową informację przeznaczoną dla łączenia modułów i ich przesuwania. Zewnętrzna struktura modułów wynikowych jest standardowa dla wszystkich translatorów DOS JS.

Stosowanie modułów wynikowych w procesie tworzenia programów pozwala użytkownikowi dzielić program na części i wybierać dla każdej części najbardziej odpowiedni język programowania. Każdą część można oddzielnie przetwarzać odpowiednim translatorem. Moduły wynikowe mogą być połączone w jeden gotowy do wykonania program za pomocą programu serwisowego, zwanego REDAKTOR[6] (lub program łączący, ang. *Linkage Editor*). Podstawowe cele procesu redagowania są następujące:

— połączyć w jeden program moduły wynikowe lub sekcje programowe,

— przekształcić tekst połączonego programu tak, aby mógł on wykonać się w zadanym podczas redagowania obszarze pamięci,

— umieścić zredagowany program w bibliotece faz (moduł przetworzony przez program REDAKTOR i gotowy do wykonania nazywa się fazą programową).

[6] Por. *DOS/JS Program łączący.*

a. Procedura redagowania

Podstawową czynnością wykonywaną przy łączeniu programów jest ustalenie połączeń między modułami, które były określone przez programistę symbolicznie za pomocą języka źródłowego.
Praca REDAKTORA polega w tym wypadku na zmianie stałych adresowych modułów wynikowych, zgodnie z adresem pamięci operacyjnej, zadanym przez użytkownika jako adres początkowy obszaru, w którym dany program ma się wykonać.

Każdy moduł wynikowy, tworzony przez translator DOS JS składa się z tekstu właściwego i informacji uzupełniającej. Tekst modułu to rozkazy i stałe. Przy translacji moduł źródłowy przetwarzany jest względem pewnego konkretnego adresu, zadanego przez programistę lub przyjętego przez translator. Przy redagowaniu modułów wynikowych i tworzeniu faz programowych trzeba często przesuwać w pamięci tekst modułu. Dane potrzebne do tego przesuwania mieszczą się w uzupełniającej informacji modułu. Oprócz tego informacja dodatkowa zawiera dane konieczne dla ustalenia związków między modułami.

Program po zredagowaniu jest fazą zawierającą wyłącznie rozkazy maszynowe, w której wszystkie elementy adresowe są przygotowane do wykonania programu we wskazanym obszarze pamięci.

Formowanie tekstu. Tekst faz programowych formuje się z tekstu modułów wynikowych na podstawie operatorów sterujących programu REDAKTOR. Utworzona faza zapisywana jest do biblioteki faz, po czym może rozpocząć się tworzenie następnej fazy. REDAKTOR wykonuje następujące działania:

— komponuje tekst fazy z modułów wynikowych,
— dołącza z biblioteki brakujące moduły, jeśli to konieczne,
— zamienia pewne części modułów wynikowych nowym tekstem, jeżeli wystąpiły karty zmian,
— przemieszcza tekst fazy odpowiednio do adresu punktu ładowania do pamięci operacyjnej,
— ustala powiązania zewnętrzne między modułami,
— zeruje obszary robocze, jeśli użytkownik tego sobie życzy,
— tworzy wspólne obszary dla całego redagowanego programu.

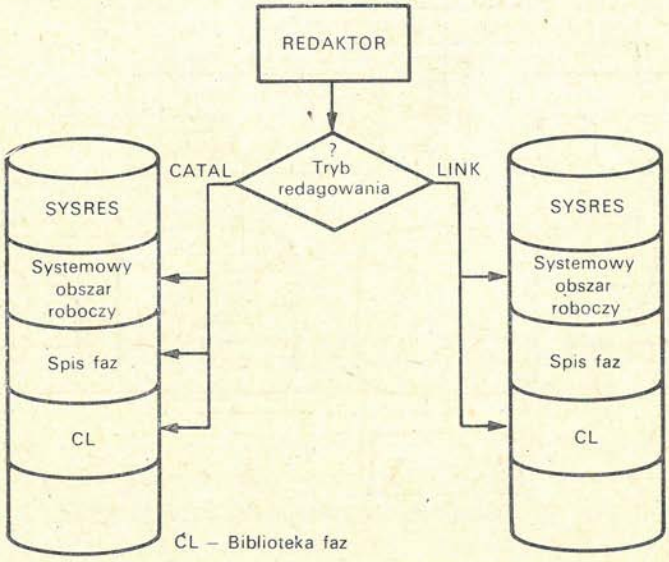
Formowanie spisów faz. Po utworzeniu każdej fazy i wniesieniu jej do biblioteki faz REDAKTOR tworzy dla fazy specjalny zapis. Zawiera on informację potrzebną systemowi do odszukania danej fazy i sprowadzenia

jej do pamięci głównej w celu wykonania. Każdy zapis zawiera nazwę fazy, adres ładowania, a także miejsce położenia fazy w bibliotece na dysku. Tak utworzone zapisy umieszczone są w specjalnym obszarze na pakiecie rezydencyjnym, zwanym systemowym obszarem roboczym. Po zakończeniu procedury redagowania w obszarze tym znajdą się zapisy dotyczące wszystkich faz programu. Zbiór ten nazywa się spisem (katalogiem) faz. Dalsze działania REDAKTORA zależeć będą od podanych w kartach sterujących warunków pracy.

b. Tryby redagowania

Fazy programu są zawsze umieszczane w bibliotece faz. Zapisy spisu faz mogą być pozostawione w roboczym obszarze systemowym lub mogą być przeniesione do biblioteki faz. Jeżeli przewiduje się wielokrotne korzystanie z programu, fazy należy zakatalogować, tzn. przenieść do biblioteki na stałe przechowanie. W czasie procedury katalogowania spis faz programu zostaje przepisany z roboczego obszaru systemowego do katalogu biblioteki.

Jeśli przewiduje się wykorzystanie zredagowanej fazy tylko w bieżącym zadaniu, wówczas powinna być ona czasowo umieszczona w bibliotece, a opis faz pozostawiony w obszarze roboczym.



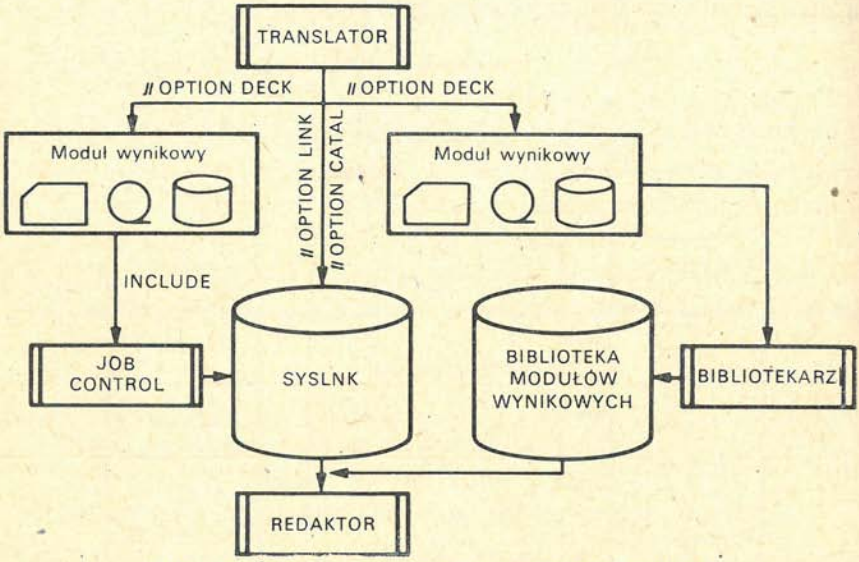
Rys. 35. Dwa rodzaje pracy REDAKTORA

Każdy z tych dwóch trybów redagowania określa się w operatorze **OPTION,** stosując odpowiednio paramerty **CATAL** lub **LINK** (por. rys. 35).

c. Informacje wejściowe dla programu REDAKTOR

Całość informacji wejściowej odczytywana jest przez program REDAKTOR z urządzenia systemowego SYSLNK, przydzielonego zawsze pamięci dyskowej, oraz z biblioteki modułów wynikowych. Informacja ta składa się z operatorów sterujących opisujących strukturę programu oraz z modułów wynikowych, które mają być połączone w jeden program. Informację tę zapisuje na SYSLNK program JOB CONTROL lub dowolny translator, pod warunkiem że zadany jest jeden z operatorów — **CATAL** lub **LINK**. Zwykle operatory sterujące REDAKTORA umieszcza się na urządzeniu SYSRDR, skąd JOB CONTROL przenosi je na SYSLNK. Translatory zapisują na SYSLNK rezultaty translacji — moduły wynikowe.

Zagadnienie to ilustruje rysunek 36.



Rys. 36. Źródła informacji wejściowej REDAKTORA

d. Struktura modułu wynikowego

Moduły wynikowe, niezależne od rodzaju użytego translatora, mają zawsze standardową formę. Mogą być zapisane na kartach, dyskach, taśmie magnetycznej, lecz zawsze mają taką samą strukturę — strukturę kartową.

Moduł wynikowy składa się z kart następujących typów:

SYM — tablica nazw symbolicznych, użytych w module źródłowym,
ESP — słownik nazw zewnętrznych,
TXT — tekst właściwy programu,
RLD — słownik przemieszczalnych stałych adresowych,
END — karta końca modułu wynikowego,
REP — karta zmian.

e. Operatory sterujące programu REDAKTOR

Operatory sterujące przeznaczone są dla opisania modułowej i fazowej struktury redagowanego programu oraz dla zadania warunków przetwarzania.

PHASE — opis fazy:

spacje **PHASE** *nazwa-fazy, adres ładowania*

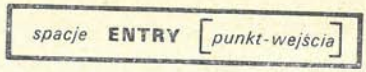
Operator ten nadaje fazie nazwę i wskazuje adres ładowania do pamięci głównej. Adres ładowania można określać w różny sposób, a najprościej przez podanie adresu symbolicznego, wcześniej określonego; litera *S* oznacza, że adresem ładowania jest adres pierwszego podwójnego słowa za obszarem pamięci zajmowanym przez SUPERVISOR.

INCLUDE — opis struktury fazy:

spacje **INCLUDE** [*nazwa-modułu*] [,(spis-sekcji)]

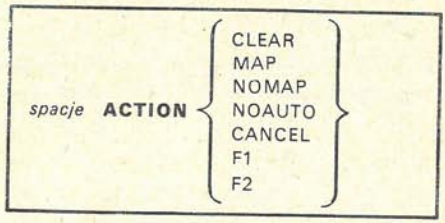
Operator ten wskazuje, jakie moduły wynikowe i sekcje programowe powinny być włączone do fazy. Moduł, którego nazwę podaje się w operatorze **INCLUDE** nie zawsze jest w całości włączany do fazy. W takim wypadku w spisie sekcji podaje się nazwy sekcji programowych, które powinny wejść do fazy.

ENTRY — koniec informacji wejściowej REDAKTORA:



Operator **ENTRY** może zadawać także symboliczną nazwę punktu wejścia do pierwszej fazy programu.

ACTION — ustalenie warunków pracy REDAKTORA:



W operatorze tym występuje zawsze jeden z następujących parametrów:

CLEAR — zerować pamięć główną przed załadowaniem fazy,

MAP — drukowanie na SYSLST spisu faz i komunikatów diagnostycznych,

NOAUTO — odwołanie zwykłego warunku **AUTOLINK**, który automatycznie dołącza do programu inne fazy z bibliotek, np. fazy modułów logicznego systemu sterowania we/wy,

CANCEL — kasowanie programu w określonych błędnych sytuacjach,

F1, F2 — stosuje się przy wieloprogramowości w celu wskazania adresów początkowych programów jako początków odpowiednich rozdziałów **F1** lub **F2.**

Program REDAKTOR może wykonywać się tylko w obszarze BG. Liczba faz jednego programu nie może być większa od 120, a tekst fazy nie może mieć więcej niż 440 640 bajtów.

f. Wyniki pracy REDAKTORA

W razie nienormalnego zakończenia zadania w wyniku błędu, zadanie zostanie skasowane, a rezultatem pracy REDAKTORA są jedynie komunikaty diagnostyczne.

W wypadku normalnego zakończenia redagowania otrzymujemy:

— gotowy do wykonania program, zapisany w bibliotece faz,

— spis faz, przechowywany w roboczym obszarze systemowym lub w bibliotece faz.

Na żądanie można dodatkowo otrzymać:
— przegląd faz drukowany na SYSLST,
— komunikaty na SYSLST, w razie nienormalnego zakończenia redagowania.

PRZYKŁAD 1

Bardzo często zadanie składa się z translacji, redagowania i natychmiastowego wykonania programu. W tym wypadku przygotowuje się następujące karty sterujące:

// **JOB PRZYK 1**	— początek zadania o nazwie PRZYK1,
// **OPTION LINK**	— określenie warunków przetwarzania,
// **EXEC PL/I**	— dokonaj translacji z języka PL/I,
〈...〉	— moduł źródłowy w języku PL/I,
/*	— koniec danych modułu źródłowego,
// **EXEC LNKEDT**	— wezwanie do pamięci głównej fazy REDAKTORA,
// **EXEC**	— wykonaj ostatnio zredagowaną fazę,
〈...〉	— dane dla programu,
/*	— koniec danych,
/&	— koniec zadania.

PRZYKŁAD 2

Zadanie PRZYKL składa się z czterech kroków: dwa kroki translacji, redagowanie z katalogowaniem i wykonanie programu. Moduły IKS, IGREK i ZET znajdują się w bibliotece modułów wynikowych, moduł ASS powstaje w wyniku translacji z języka ASSEMBLER, moduł FOR — z języka FORTRAN.

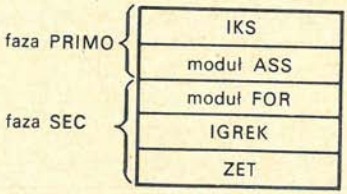
// **JOB PRZYKL**	
// **OPTION CATAL**	— katalogowanie w bibliotece faz,
PHASE PRIMO,S	— nadanie fazie nazwy PRIMO i ustalenie punktu ładowania do pamięci bezpośrednio po SUPERVISORZE,
INCLUDE IKS	— włącza do fazy z biblioteki modułów wynikowych moduł IKS,
// **EXEC ASSEMBLY**	— wykonać translację,
〈...〉	— program źródłowy w języku ASSEMBLER,

```
/*
PHASE SEC,*          — utworzyć fazę o nazwie SEC, PHASE
                       SEC*, punkt ładowania — po zakoń-
                       czeniu fazy poprzedniej,
// EXEC FORTRAN      — translować moduł źródłowy w
                       FORTRANIE,
  <...>              — moduł źródłowy,
/*
INCLUDE IGREK        — włączyć do fazy SEC moduł IGREK,
INCLUDE ZET          — włączyć do fazy moduł ZET,
ENTRY                — koniec modułów,
// EXEC LNKEDT       — redagować,
// EXEC PRIMO        — wykonać program PRIMO.
/&
```

W rezultacie, do momentu wczytania karty // **EXEC LNKEDT**, na SYSLNK będzie znajdować się następująca informacja:

```
PHASE,S
INCLUDE IKS
<moduł wynikowy ASS>
PHASE,*
<moduł wynikowy FOR>
INCLUDE IGREK
INCLUDE ZET
ENTRY
```

Po zredagowaniu program będzie się składał z następujących faz:



Programy PRIMO i SEC zostaną zapisane w bibliotece faz i będą zakatalogowane. W następnych zadaniach będą mogły być użyte dowolną ilość razy.

12. BIBLIOTEKARZ

W systemie operacyjnym DOS JS istnieją trzy rodzaje bibliotek:
— biblioteka faz (CL — ang. *System Core Image Library*),
— biblioteka modułów wynikowych (RL — ang. *Relocatable Library*),
— biblioteka modułów źródłowych (SL — ang. *Source Statement Library*).

Program BIBLIOTEKARZ[7] (ang. *LIBRARIAN*) jest przeznaczony do tworzenia i korekcji bibliotek oraz wydruku ich zawartości i kopiowania.

Biblioteka faz musi występować zawsze, natomiast pozostałe biblioteki nie są obowiązkowe. Wszystkie te biblioteki mieszczą się w pamięci dyskowej na pakiecie rezydencyjnym systemu.

Oprócz wymienionych bibliotek w systemie mogą być prowadzone prywatne biblioteki modułów wynikowych i modułów źródłowych. Biblioteki prywatne, w odróżnieniu od bibliotek systemowych, nie są przechowywane na pakiecie rezydencyjnym.

a. Struktura bibliotek

1. Biblioteka faz (CL).

Bibliotekę faz tworzą fazy programowe. Każda faza jest wynikiem przetworzenia modułu wynikowego przez program REDAKTOR. Biblioteka prowadzi katalog, w którym zapamiętane są podstawowe informacje o każdej fazie:
— nazwa fazy,
— adres fazy w pamięci dyskowej,
— ilość bloków fizycznych, tworzących fazę,
— ilość bajtów w ostatnim bloku fazy,
— adres pamięci głównej, według którego faza zostaje umieszczona w pamięci do wykonania,
— punkt wejścia do fazy.

Najmniejszą jednostką biblioteki faz jest *faza*. Jeżeli program składa się z kilku faz, wówczas wymaga się, aby pierwsze cztery znaki nazwy każdej fazy były jednakowe.

2. Biblioteka modułów wynikowych (RL).

Biblioteka modułów wynikowych składa się z dowolnej liczby modu-

[7] Por. *DOS/JS Bibliotekarz.*

łów (ograniczonej tylko pojemnością pamięci), będących wynikiem translacji dowolnego translatora DOS JS.

Biblioteka modułów wynikowych umożliwia przechowywanie często używanych modułów, a następnie włączenie ich do programów razem z innymi modułami podczas redagowania.

Biblioteka modułów zawiera również katalog, w którym w każdej pozycji znajduje się nazwa modułu, adres modułu na dysku, wielkość modułu i inne dane.

3. Biblioteka modułów źródłowych (SL).

Biblioteka modułów źródłowych składa się z *ksiąg*. Księgą jest zbiór instrukcji napisanych w dowolnym języku programowania. Większym elementem biblioteki jest *podbiblioteka*. Jest to zbiór ksiąg, przeznaczonych do przetworzenia przez określony translator DOS JS.

Podstawowym zadaniem biblioteki modułów źródłowych jest zgromadzenie i przechowywanie makroinstrukcji języka ASSEMBLER oraz modułów źródłowych napisanych w innych językach DOS JS. Oprócz tego biblioteka może zawierać księgi, tworzące strumienie zadań.

Moduły źródłowe mogą być umieszczone w bibliotece tylko przez program BIBLIOTEKARZ. Biblioteka wyposażona jest w katalog, który zawiera następujące informacje o każdej księdze:

— nazwa podbiblioteki,
— nazwa księgi,
— adres księgi na dysku,
— wielkość księgi,
— wersja i modyfikacja.

4. Biblioteki prywatne

Biblioteki prywatne znajdują się na innym pakiecie dysków niż rezydencja systemu. Używanie bibliotek prywatnych umożliwia użytkownikowi rozszerzenie pojemności bibliotek, w szczególności biblioteki faz przez przeniesienie zawartości z pozostałych bibliotek systemowych do bibliotek prywatnych. Biblioteki prywatne mogą być zorganizowane tematycznie.

Biblioteki prywatne zakłada się na specjalnych urządzeniach logicznych — SYSRLB (biblioteka modułów wynikowych) i SYSSLB (biblioteka modułów źródłowych).

b. Funkcje programu BIBLIOTEKARZ

BIBLIOTEKARZ spełnia trzy podstawowe rodzaje zadań: prowadzenie bibliotek, wydawanie informacji z biblioteki, tworzenie i kopiowanie

bibliotek. Zadania te są wykonywane przez następujące programy BIBLIOTEKARZA:

MAINT — obsługiwanie wszystkich bibliotek,
CSERV — wydawanie informacji z biblioteki faz,
RSERV — wydawanie informacji z biblioteki modułów wynikowych,
SSERV — wydawanie informacji z biblioteki modułów źródłowych,
DSERV — wydawanie katalogów wszystkich biblioteki,
CORGZ — tworzenie i kopiowanie bibliotek.

1. Prowadzenie bibliotek polega na:
   — katalogowaniu (włączenie fazy, modułu wynikowego lub źródłowego do odpowiedniej biblioteki),
   — usuwaniu pozycji z biblioteki,
   — przesuwaniu (stosuje się dla nadania nowych nazw elementom biblioteki),
   — korekcji, polegającej na zmianie treści oddzielnych instrukcji w księgach (dotyczy tylko biblioteki modułów źródłowych),
   — koncentracji, polegającej na fizycznej eliminacji usuniętych pozycji biblioteki,
   — reorganizacji, pozwalającej zmieniać rozmieszczenie pozycji biblioteki.
2. Funkcje wydawnicze pozwalają na:
   — wydrukowanie lub wyperforowanie zawartości całych bibliotek lub tylko niektórych pozycji,
   — wydrukowanie lub wyperforowanie katalogów bibliotek.
3. Funkcje tworzenia i kopiowania umożliwiają:
   — kopiowanie zawartości pakietu rezydencyjnego,
   — tworzenie nowych zbiorów rezydencyjnych,
   — tworzenie bibliotek prywatnych.

Programy BIBLIOTEKARZA używają własnych instrukcji. Podajemy przykłady typowych zadań z zastosowaniem BIBLIOTEKARZA.

PRZYKŁAD 3

Zakatalogować moduł BETA w bibliotece modułów wynikowych.

```
// JOB ALFA
// EXEC MAINT
CATALR BETA
<moduł wynikowy>
/*
/&
```

PRZYKŁAD 4

Usunąć z biblioteki faz dwie fazy: DELTA i GAMMA

```
// JOB USUN
// EXEC MAINT
DELETC DELTA, GAMMA
/*
/&
```

PRZYKŁAD 5

Zmienić nazwę modułu w bibliotece modułów wynikowych z STR na NOW.

```
// JOB ZMIANA
// EXEC MAINT
RENAMR STR, NOW
/*
/&
```

PRZYKŁAD 6

Ścieśnić bibliotekę faz usuwając nieaktualne pozycje.

```
// JOB DD
// EXEC MAINT
CONDS CL
/*
/&
```

PRZYKŁAD 7

Wydrukować fazę PI z biblioteki faz.

```
// JOB DRFAZ
// EXEC CSERV
DSPLY PI
/*
/&
```

PRZYKŁAD 8

Wydrukować zawartość całej biblioteki modułów źródłowych.

```
// JOB DRSR
// EXEC SSERV
```

```
DSPLY ALL
/*
/&
```

PRZYKŁAD 9

Wydrukować katalog biblioteki modułów wynikowych.

```
// JOB KATAL
// EXEC DSERV
DSPLY RD
/*
/&
```

13. Pomocnicze programy serwisowe

Pomocnicze programy serwisowe[8] (ang. *UTILITY*) wchodzące w skład DOS JS umożliwiają przepisywanie kartotek z jednych nośników na inne. System zawiera następujące programy:

KARTY — DRUK (drukowanie kartoteki kartowej),
KARTY — DYSK (przepisywanie zawartości kartoteki z kart na dysk),
KARTY — TAŚMA (przepisywanie kartoteki z kart na taśmę magnetyczną),
DYSK — KARTY (perforowanie kartoteki zapisanej w pamięci dyskowej),
DYSK — DYSK (kopiowanie kartoteki dyskowej),
DYSK — DRUK (drukowanie kartoteki zapisanej na dysku),
DYSK — TAŚMA (przepisywanie kartoteki z dysku na taśmę magnetyczną),
TAŚMA — KARTY (perforowanie kartoteki przechowywanej na taśmie),
TAŚMA — DYSK (przepisywanie kartoteki z taśmy magnetycznej na dysk),
TAŚMA — DRUK (drukowanie kartoteki przechowywanej na taśmie magnetycznej),
TAŚMA — TAŚMA (kopiowanie kartoteki na taśmie magnetycznej).

Wszystkie te programy, z wyjątkiem programu DYSK — DYSK, mogą pracować wyłącznie z kartotekami sekwencyjnymi (por. bliższe dane o kartotekach w rozdziale VIII).

[8] Por. *DOS/JS Programy transmisji.*

Pomocnicze programy serwisowe umożliwiają przepisywanie kartotek bez zmiany lub ze zmianą struktury zapisów, z blokowaniem lub rozblokowaniem kartotek wyjściowych. Informacja, konieczna przy przepisaniu kartoteki, zadawana jest w operatorach sterujących programów serwisowych. Używa się następujących operatorów:

— operator modyfikacji,
— operator żmiany struktury zapisu,
— operator początku,
— operator końca.

W zależności od wartości parametrów tych operatorów przygotowuje się programy UTILITY do wykonania konkretnego zadania. Programy te mogą być wykonane w dowolnym rozdziale pamięci głównej w trybie przetwarzania wsadowego.

Podamy przykład przygotowania zadania z użyciem programu serwisowego UTILITY.

PRZYKŁAD 10

Zadanie WYDRUK używa serwisowego programu pomocniczego TAŚMA — DRUK w celu wydrukowania zawartości kartoteki zapisanej na taśmie magnetycznej.

```
// JOB WYDRUK
// ASSGN SYS004,X'280'      — urządzenie wejściowe,
// ASSGN SYS005,X'00F'      — urządzenie wyjściowe (drukarka)
// UPSI 10000               — przełączniki programowe
// EXEC TPPR                — wykonać program TAŚMA —
                                — DRUK
//UTP TLF, FF,A = (121,121), B = (120), IU, SD
//FS 2,120,1
// END
/&
```

W programie tym UTP jest operatorem modyfikacji z następującymi parametrami:

TLF — oznacza, że zapisy z taśmy magnetycznej będą przekomponowane,
FF — oznacza zapisy stałej długości,
A(121,121) — opisuje zbiór wejściowy. Pierwsza liczba w nawiasie podaje długość zapisu, druga długość bloku,
B(120) — opisuje zbiór wyjściowy. Liczba w nawiasie określa liczbę znaków w wierszu,

IU — oznacza, że należy przewinąć taśmę przed i po przepisaniu,
SD — sterowanie formatem wydruku odbywa się według pierwszego symbolu w zapisie,
FS jest operatorem dekompozycji, w którym:
2 — oznacza numer pierwszego znaku pola wejściowego,
120 — oznacza długość pola w bajtach,
1 — numer początkowego znaku zapisu wyjściowego względem początku zapisu.

14. Inne programy systemu operacyjnego DOS JS

Translator języka RPG

Język RPG[9] (skrót ang. *Raport Program Generator*) jest językiem stosowanym w systemach przetwarzania danych. Jest szczególnie przydatny do generowania skomplikowanych wydruków z różnego typu kartotek. Oprócz tego język ten umożliwia wykonywanie prostych obliczeń. Programista, posługujący się językiem RPG, może:

— tworzyć kartoteki o organizacji sekwencyjnej i indeksowo-sekwencyjnej,
— drukować zawartość kartotek o dowolnej organizacji,
— wyszukiwać żądane zapisy w kartotekach,
— aktualizować kartoteki,
— wykonywać niezłożone obliczenia.

Programowanie w języku RPG polega na wypełnianiu specjalnych formularzy, co nie jest rzeczą trudną i nie wymaga większych znajomości zasad programowania.

Translator języka FORTRAN

Dyskowy System Operacyjny pozwala programować również w najbardziej znanym na świecie języku — FORTRAN[10]. Jest to możliwe dzięki translatorowi tego języka, wchodzącemu w skład DOS JS. Jest to FORTRAN BASIC, czyli FORTRAN podstawowy, niewiele odbiegający od standardowej wersji tego języka, przyjętej przez organizację ANSI (*American National Standards Institute*). Wersja FORTRANU DOS JS zawiera środki dla sterowania we/wy dla pamięci z bezpośrednim dostę-

[9] Por. *DOS/JS RPG — opis języka; DOS/JS RPG w systemie DOS/JS.*
[10] Por. *DOS/JS FORTRAN — opis języka; DOS JS, FORTRAN — w systemie DOS/JS; DOS/JS, FORTRAN IV — opis języka; DOS/JS, RPG — opis języka.*

pem, tablice trójwymiarowe, stałe podwójnej precyzji, operator EXTERNAL i kilka innych dodatkowych możliwości, przy całkowitym zachowaniu standardu.

Programy sortowania i łączenia

DOS JS zawiera dwa programy sortowania i łączenia[11]:

— sortowanie i łączenie zbiorów taśmowych,

— sortowanie i łączenie zbiorów dyskowych i taśmowych.

Programy te umożliwiają sortowanie kartotek w porządku rosnącym lub malejącym, zgodnie z kluczem składającym się maksymalnie z 12 pól. Program sortowania/łączenia zbiorów taśmowych może sortować jednocześnie 8 kartotek i łączyć w jedną kartotekę do 7 zbiorów. Program sortowania zbiorów dyskowych i taśmowych może sortować do 9 zbiorów lub łączyć do 8 kartotek.

Obydwa programy przetwarzają etykiety kartotek, pozwalają na włączanie procedur użytkownika, zawierają procedury przerywające i wznawiające operację sortowania i łączenia.

Autotest

Program AUTOTEST[12] jest bardzo pomocnym środkiem, służącym do uruchomienia programów napisanych w języku ASSEMBLER. Program ten umożliwia:

— korygowanie programu w postaci modułu wynikowego (zamienianie, usuwanie lub dołączanie instrukcji, zamiana stałych); może się to odbywać bez ponownej translacji programu,

— drukowanie zawartości pamięci głównej i rejestrów w dowolnej postaci.

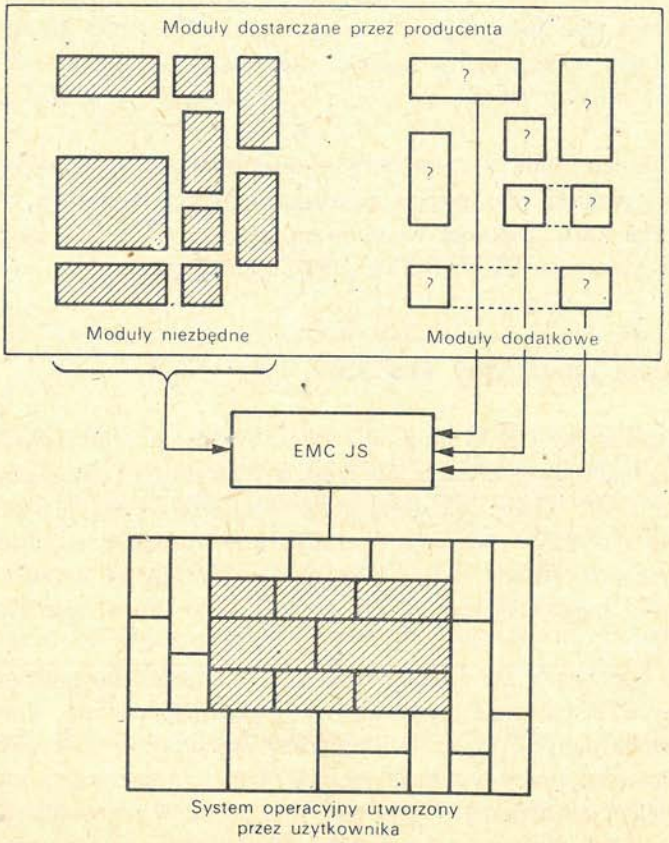
15. Generacja systemu

Dyskowy System Operacyjny Jednolitego Systemu[13], podobnie jak inne współczesne systemy operacyjne, charakteryzuje się budową modułową. Oznacza to, że system składa się z wielkiej ilości modułów, pełniących różne funkcje. Nie wszystkie moduły będą wykorzystywane przez użytkownika. Może on wybrać tylko te moduły, które są konieczne dla jego zastosowań. Proces tworzenia konkretnej wersji systemu operacyjnego,

[11] Por. *DOS/JS Sortowanie zbiorów taśmowych; DOS/JS Sortowanie zbiorów taśmowych i dyskowych.*
[12] Por. *DOS/JS AUTOTEST — program testujący.*
[13] Por. *DOS/JS Generowanie systemu.*

uwzględniający specyficzne cechy zadań wykonywanych na danej maszynie i jej konfigurację nazywa się generacją systemu.

Pełen zestaw modułów dostarczany jest przez producenta na taśmie magnetycznej, zwanej taśmą dystrybucyjną. Użytkownik precyzuje swoje wymagania co do zawartości swojej wersji systemu operacyjnego przez przygotowanie odpowiednich kart sterujących. Generowanie systemu rozpoczyna się od zainicjowania pakietów dyskowych. Po przygotowaniu pamięci dyskowej wprowadza się do niej wszystkie moduły niezbędne do działania systemu, jak również te dodatkowe, których zażądał użytkownik. Procesem tym kieruje tzw. prasystem, który mieści się na początku taśmy dystrybucyjnej. Proces tworzenia systemu DOS ilustruje rysunek 37.



Rys. 37. Schemat tworzenia systemu DOS JS

Podczas generacji system tworzy biblioteki:

— bibliotekę źródłową,
— bibliotekę modułów wynikowych,
— bibliotekę faz.

Biblioteka faz, niezależnie od woli użytkownika, musi zawierać program sterujący, program REDAKTOR i program BIBLIOTEKARZ.

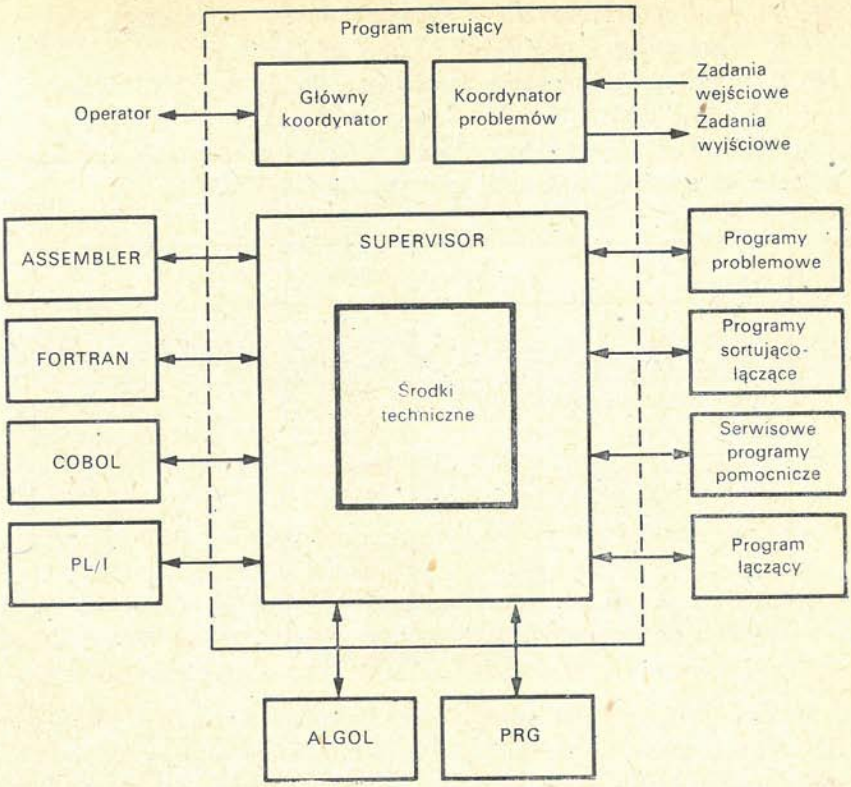
Jednym z ostatnich etapów generacji systemu jest generacja SUPERVISORA. Użytkownik w tym wypadku decyduje o tym, czy system będzie wieloprogramowy, czy możliwe będzie przetwarzanie wsadowe w rozdziałach pierwszoplanowych, czy ma być zapewniona ochrona pamięci, czy będzie zegar itd. Makroinstrukcje generacji SUPERVISORA umożliwiają opisanie konfiguracji systemu liczącego (ilość urządzeń zewnętrznych i ich typy), dokonać standardowych przydziałów urządzeń, ustalić standardowy tryb pracy systemu (np. wydruk zawartości pamięci przy nienormalnym zakończeniu zadania, drukowanie operatorów sterujących, ilość wierszy na jednej stronie w urządzeniu SYSLST i tym podobne).

Dla wygenerowania systemu DOS konieczna jest co najmniej jedna jednostka pamięci dyskowej, jednostka pamięci taśmowej, czytnik i dziurkarka kart, drukarka wierszowa i monitor. Znacznie efektywniej przebiega generacja na podstawie dwóch jednostek pamięci dyskowej.

16. System operacyjny OS JS

System Operacyjny OS JS (ang. *Operating System*) przeznaczony jest dla maszyn średnich i dużych, a więc dla R-30, R-32, R-40, R-50, R-60. Podobnie jak system DOS, składa się z dużej ilości modułów, z których użytkownik może stworzyć wariant systemu operacyjnego najbardziej odpowiadający jego potrzebom. Jednak należy wyraźnie podkreślić, że nie ma żadnej wymienności między systemami OS a DOS; są to dwa zupełnie różne systemy.

System operacyjny OS JS zapewnia właściwe funkcjonowanie środków technicznych Jednolitego Systemu, ich wykorzystanie dla wsadowego przetwarzania danych, pracę wieloprogramową, pracę w trybie bezpośredniego dostępu, pracę w czasie rzeczywistym, automatyczną rejestrację danych, teleprzetwarzanie, bogaty repertuar środków ułatwiających pracę programisty itd. System operacyjny OS JS zapewnia również funkcjono-



Rys. 38. System operacyjny OS JS

wanie systemów dwuprocesorowych ze wspólną pamięcią operacyjną. Taki typ pracy przewidziano w modelach JS-1040 i JS-1050.

Strukturę systemu OS JS przedstawia rysunek 38. System operacyjny składa się z programu sterującego i programów przetwarzających. Do zadań programu sterującego należy planowanie strumienia zadań, sterowanie operacjami we/wy, realizacja przyjętej organizacji zbiorów danych, realizacja wieloprogramowości i wiele innych. Całością działania systemu operacyjnego kieruje Główny Koordynator (ang. *Master Scheduler*), natomiast Koordynator Problemów (ang. *Job Scheduler*) wprowadza zadania do systemu, planuje kolejność ich wykonania i inicjuje ich realizację.

Istnieją trzy zasadnicze konfiguracje programu sterującego:
PCP — system jednoprogramowy,

MFT — wieloprogramowanie z ustaloną ilością zadań (ang. *Multiprogramming with a Fixed Number of Task*),

MVT — wieloprogramowanie ze zmienną ilością zadań (*Multiprogramming with a Variable Number of Task*).

Użycie takiej czy innej konfiguracji zależy od potrzeb użytkownika oraz od pojemności pamięci konkretnej maszyny (por. rys. 39).

Pojemność pamięci	64K	128K	256K	512K	1024K
R-20 R-22	DOS	DOS OS-PCP	DOS OS-PCP OS-MFT		
R-30 R-32		DOS OS-PCP	DOS OS-PCP OS-MFT	OS-PCP OS-MFT OS-MVT	
R-40			DOS OS-PCP OS-MFT	OS-PCP OS-MFT OS-MVT	OS-PCP OS-MFT OS-MVT
R-50			DOS OS-PCP OS-MFT	OS-PCP OS-MFT OS-MVT	OS-PCP OS-MFT OS-MVT

Rys. 39. Zależność między pojemnością pamięci operacyjnej a konfiguracją systemu operacyjnego

System operacyjny OS zapewnia użytkownikowi znacznie więcej możliwości niż system DOS. Bardzo rozwinięty system wieloprogramowości, szeregowanie zadań w strumieniach zadań wejściowych i wyjściowych w pamięci dyskowej, możliwość przetwarzania równoległego podzadań w ramach jednego zadania, duży wybór translatorów z pełnymi wersjami języków PL/I, COBOL, FORTRAN, wielka ilość pakietów programów użytkowych — wszystko to zapewnia dużą efektywność systemu i szerokie jego stosowania.

VI. ASSEMBLER

Podstawowym językiem programowania maszyn cyfrowych Jednolitego Systemu jest język ASSEMBLER[1]. Jest to język symboliczny, ukierunkowany maszynowo, będący kluczem do zrozumienia całego systemu. W języku ASSEMBLER przyjęto zasadę symbolicznego (literowego) oznaczania kodów operacji, adresów i nazw danych. Wyznaczanie adresów rzeczywistych instrukcji i danych odbywa się automatycznie. Program napisany w języku ASSEMBLER musi być przed wykonaniem tłumaczony na język wewnętrzny maszyny. Proces ten, zwany translacją, realizuje specjalny program — translator języka symbolicznego, nazywany też assemblerem. Z reguły assembler przetwarza program z języka symbolicznego na język maszynowy w stosunku „jeden do jeden", tzn. jedną instrukcję języka symbolicznego na jeden rozkaz maszynowy. Translacja odbywa się w dwóch przebiegach: w pierwszym następuje podział pamięci i nadanie wartości nazwom symbolicznym, w drugim tworzy się program w standardowej postaci modułu wynikowego. W procesie translacji odbywa się syntaktyczna kontrola poprawności programu, a następnie wydruk komunikatów diagnostycznych o wykrytych błędach.

W języku ASSEMBLER przewidziano wykonanie pewnych funkcji pomocniczych takich, jak sterowanie procesem translacji i wydrukiem kontrolnym. Do tego celu służą specjalne instrukcje, zwane instrukcjami assemblerowymi, które nie mają zwykle żadnych odpowiedników w module wynikowym programu.

W języku ASSEMBLER można posługiwać się również makroinstrukcjami, które podczas translacji generują wiele rozkazów maszynowych. Programista może stosować w swoich programach makroinstrukcje własne, napisane przez siebie, lub może używać makroinstrukcji dostarczonych przez producenta z systemem operacyjnym.

[1] Por. *Programmirowanije na jazykie ASSEMBLERA JS IBM*, Moskwa 1975.

Programowanie w języku ASSEMBLER, chociaż bardziej czasochłonne niż programowanie w językach algorytmicznych czy proceduralnych, takich jak FORTRAN lub PL/1, może być bardziej efektywne, ponieważ programy są z reguły krótsze, czas trwania translacji i wykonywania programu mniejszy, a wykorzystanie pamięci bardziej racjonalne.

ASSEMBLER jest językiem uniwersalnego przeznaczenia.

1. Struktura języka

a. Elementarna struktura języka

Program źródłowy w języku ASSEMBLER jest sekwencją zdań, wprowadzonych do maszyny cyfrowej zwykle za pośrednictwem 80-kolumnowych kart perforowanych. Zdania źródłowe mogą być czterech typów: instrukcje maszynowe, instrukcje assemblerowe, makroinstrukcje, komentarze.

Instrukcje maszynowe. Każda instrukcja maszynowa zostaje w wyniku translacji przetłumaczona na jeden rozkaz maszynowy, wchodzący w skład listy rozkazów standardowych maszyn Jednolitego Systemu.

Instrukcje assemblerowe. Instrukcje assemblerowe służą do sterowania translacją i nie zostają zamieniane na rozkazy maszynowe w module wynikowym. Zadaniem ich jest określenie początkowego adresu w pamięci, od którego umieszczony zostanie w pamięci program, rezerwowanie obszarów roboczych pamięci, wprowadzanie do pamięci stałych itp.

Makroinstrukcje. Makroinstrukcją jest pseudorozkaz, który powoduje umieszczenie w module wynikowym sekwencji rozkazów maszynowych. W skład systemu operacyjnego wchodzi szereg makroinstrukcji niezbędnych programiście, np. dla organizacji wejścia/wyjścia.

Komentarze. Komentarze służą do opisu programu i nie są przez translator przetwarzane. Komentarz pojawia się w całości na wydruku kontrolnym programu.

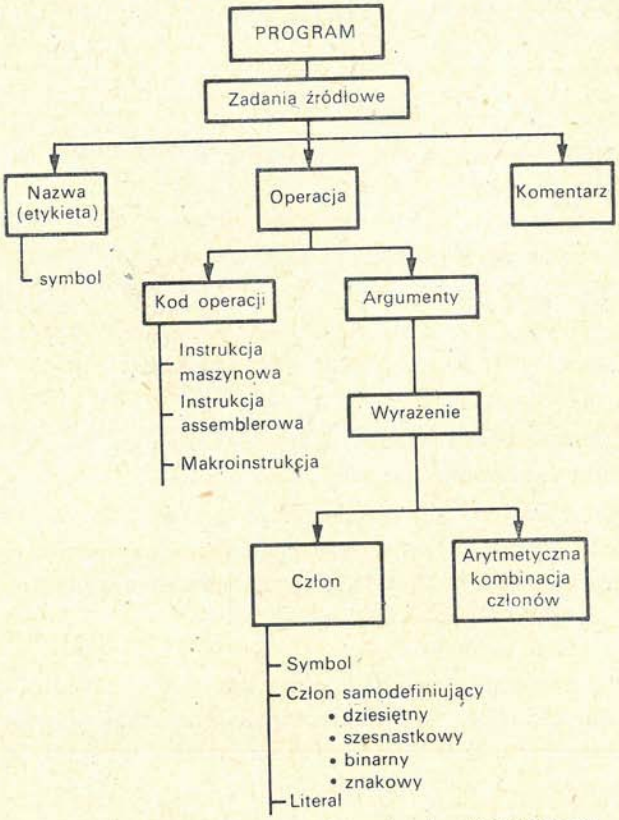
Wszystkie instrukcje i makroinstrukcje zawierają następujące składniki:

— nazwę (etykieta),
— mnemotechniczny kod operacji,
— argumenty,
— komentarze (tekst po ostatnim argumencie, oddzielony od niego co najmniej jedną spacją).

W każdym zdaniu źródłowym musi wystąpić kod operacji. Inne składniki zdania, tzn. nazwa i komentarz są nieobowiązkowe. Argumenty wymagane są we wszystkich instrukcjach maszynowych w większości instrukcji assemblerowych.

Każdy program składa się z jednej lub kilku sekcji programowych. Złożone programy można podzielić na sekcje i translować każdą sekcję oddzielnie. Ułatwia to testowanie programu. Sekcje łączy się w jeden program za pomocą programu systemowego REDAKTOR.

Rysunek 40 przedstawia elementarną strukturę języka ASSEMBLER.



Rys. 40. Elementarna struktura języka ASSEMBLER

b. Elementy programu

Zdania źródłowe, tworzące program w języku ASSEMBLER, mogą składać sie z następujących znaków*:

litery: ABCDEFGHIJKLMNOPQRSTUVWXYZ $ # @

cyfry: 0123456789

znaki specjalne: + −,=.*( )' /& spacja.

Zdania źródłowe mogą zawierać następujące elementy.

Symbol — oznacza w programie adres i występuje w charakterze nazwy instrukcji lub nazwy obszaru danych. Symbol składa się z od jednego do ośmiu znaków: liter lub cyfr, przy czym pierwszym znakiem musi być litera (znaki $, #, @ traktuje się jako litery). Symbol nie może zawierać spacji.

PRZYKŁAD 1

Przykłady prawidłowych symboli:

SKOK

A466

$21

Te zaś nazwy są błędne:

PISZTABUL (więcej niż 8 znaków),

1 Jeden (rozpoczyna się od cyfry i zawiera małe litery),

DWA+1 (zawiera niedopuszczalny znak specjalny +),

PL 1 (występuje spacja). ■

Człony reprezentują pewną wartość. Wartość ta może być nadana członowi przez translator lub wynikać z samej istoty członu.

Człon samodefiniujący — jest to taki człon, którego wartość zawarta jest w nim samym. Wartość ta nie jest zmieniana przez kompilator, jest wartością bezwzględną, tzn. nie zmienia jej przemieszczanie programu. Istnieją cztery typy członów samodefiniujących:

1. Dziesiętny człon samodefiniujący.

Dziesiętnym członem samodefiniującym jest całkowita liczba dziesiętna bez znaku, np. 21, 1024. Translator przekształca liczby dziesiętne w binarne. Wartość członu dziesiętnego nie może przekroczyć liczby 16 777 215.

2. Szesnastkowy człon samodefiniujący.

Szesnastkowy człon samodefiniujący składa się z ciągu znaków szesnast-

* Spację będziemy oznaczać w druku jako puste miejsce lub znakiem ␣, jeżeli istotna będzie liczba spacji. W celu odróżnienia zera od litery O — w sytuacjach wywołujących wątpliwości — zero oznaczamy znakiem Ø.

kowych (do sześciu znaków). Znaki te ujęte są w apostrofy i poprzedzone literą X. Każde dwie cyfry szesnastkowe tworzą jeden bajt. Przykłady szesnastkowych członów samodefiniujących: X'2A3', X'F1F8', X'FFFFFF'

3. Binarny człon samodefiniujący.

Binarny człon samodefiniujący składa się z ciągu bitów bez znaku, ujętych w apostrofy i poprzedzonych literą B, np. B'000001010'.
W binarnym członie można umieścić nie więcej niż 24 bity.

4. Znakowy człon samodefiniujący.

Znakowy człon samodefiniujący stanowią jeden, dwa lub trzy znaki ujęte w apostrofy, poprzedzone literą C. W członie tym mogą wystąpić dowolne znaki. Oto kilka przykładów: C'ALA', C'', C'''A''' (w wypadku gdy elementem członu znakowego ma być apostrof lub ampersant, wówczas znaki te należy zapisać dwukrotnie).

Literal — jest to stała poprzedzona znakiem równości. Podczas translacji assembler napotykając literal tworzy stałą o wartości w nim określonej i zapisuje ją do specjalnej tablicy literali, a do pola argumentów tłumaczonej instrukcji wstawia adres tej stałej. Literale umożliwiają wprowadzenie do programu stałych, takich jak liczby, adresy lub nazwy. Przykłady literali: = X'FO', = B'1010', = C'A1'.

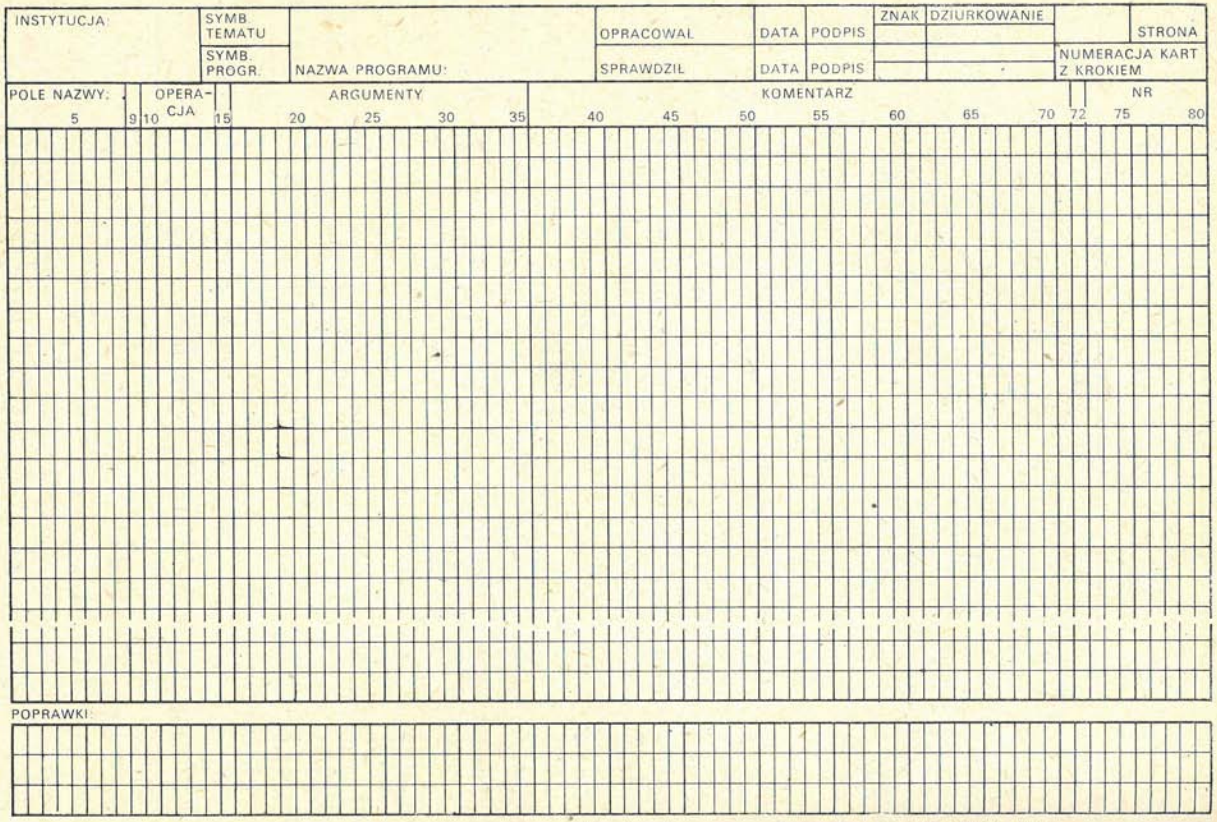
Symbole, literale i człony samodefiniujące noszą wspólną nazwę członów. Kombinacja członów połączonych znakami + (plus), − (minus),* (znak mnożenia), / (znak dzielenia) nazywa się *wyrażeniem.* W wyrażeniu mogą być użyte nawiasy, zgodnie z zasadami algebry.

PRZYKŁAD 2

Przykłady poprawnych wyrażeń:
ALFA*8
DWA/TRZY
((A/B)-POLE)*7

c. Kodowanie programów w języku ASSEMBLER

Jak już wspominaliśmy, program jest wprowadzany do maszyny cyfrowej przede wszystkim z czytnika kart. Dla ułatwienia naniesienia programu na karty i uniknięcia błędów perforowania program zapisuje się na standardowych formularzach (por. rys. 41). Zawartość jednego wiersza formularza jest dziurkowana na jednej karcie.



Rys. 41. Formularz standardowy do pisania programów w języku ASSEMBLER

Formularz składa się z dwóch pól: pola zdań, zapisywanych w kolumnach 1—71 oraz pola identyfikacyjno-porządkowego, któremu odpowiadają kolumny 73—80. Pole identyfikacyjno-porządkowe nie jest częścią zdania źródłowego. Tekst zdania źródłowego może zajmować kolumny 1—71, a w razie potrzeby dodatkowo kolumny 16—71 następnej karty. W takim wypadku musi być zasygnalizowana kontynuacja przez umieszczenie w kolumnie 72 dowolnego znaku różnego od spacji.

Instrukcje perforuje się w następującej kolejności:

— nazwa etykiety — kolumny 1—8.

— mnemotechniczny kod operacji — kolumny 10—14,

— argumenty, oddzielone od siebie przecinkami — kolumny 16—71,

— komentarze (po ostatnim argumencie musi następować co najmniej jedna spacja, po czym można pisać dowolny tekst, który jest traktowany jako komentarz).

Kolumny 9 i 15 nie są perforowane.

Zdanie, które jest komentarzem, musi rozpoczynać się od gwiazdki w pierwszej kolumnie.

2. Instrukcje definiowania danych

W programie, oprócz instrukcji realizujących żądane obliczenia występują dane, które należy uprzednio zdefiniować. Istnieją dwie instrukcje assemblerowe definiowania danych: **DC** — zdefiniuj stałą i **DS** — zdefiniuj obszar pamięci. Instrukcje te służą do zapisywania stałych w pamięci, a także do definiowania i rezerwowania obszarów pamięci.

a. Definiowanie stałych

Za pomocą instrukcji **DC** można określić stałe używane w programie. Postać instrukcji **DC** jest następująca:

{ *symbol* | *spacja* }	**DC**	*argumenty*

Każdy argument składa się z czterech pól, a mianowicie:

1	2	3	4
współczynnik krotności	*typ*	*modyfikatory*	*stała*

Pola drugie i czwarte muszą występować zawsze, natomiast pozostałe mogą być opuszczone. Niedopuszczalne jest rozdzielanie tych pól spacjami, jak również używanie spacji w poszczególnych polach, chyba że wartością stałej jest spacja. Rozpatrzmy poszczególne pola:

1. *Współczynnik krotności.* Jest dziesiętnym członem samodefiniującym, który wskazuje, ile razy ma być generowana dana stała. Brak współczynnika krotności świadczy o tym, że jest on równy jedności. Dozwolona jest zerowa wartość współczynnika krotności.

2. *Typ.* Określony jest przez jedną literę, która wskazuje translatorowi jak ma być interpretowana stała. Kody typów stałych zawiera tablica 15.

3. *Modyfikatory.* Opisują długość, skalowanie lub wykładnik. Jeśli modyfikatorów jest więcej niż jeden, powinny występować w kolejności: długość, skala, wykładnik.

a) modyfikator długości, zapisuje się jako L*n*, gdzie *n* jest dziesiętnym członem samodefiniującym albo dodatnim wyrażeniem bezwzględnym ujętym w nawiasy. Jeśli modyfikator długości nie jest zadany, obowiązuje tzw. długość domyślna, wynikająca z kontekstu. Długości domyślne dla różnych typów stałych podajemy w tablicy 15,

b) modyfikator skali (S*n*) — dla liczb stałoprzecinkowych podaje potęgę dwójki (*n*), przez którą ma być pomnożona stała po jej konwersji na postać binarną. Dzięki temu można uzyskać przesunięcie przecinka w liczbie dwójkowej stałoprzecinkowej, który w liczbach stałoprzecinkowych ma swoje zasadnicze położenie na prawo od skrajnej prawej pozycji. Dla liczb zmiennoprzecinkowych modyfikator ten wskazuje liczbę pozycji szesnastkowych, o które część ułamkowa ma być przesunięta w prawo,

c) modyfikator wykładnika (E*n*) — może być użyty ze stałymi stałoprzecinkowymi (F,H) i zmiennoprzecinkowymi (E,D). Modyfikator ten oznacza potęgę dziesięciu (*n*), przez którą ma być pomnożona stała przed jej konwersją na właściwy format wewnętrzny.

4. *Stała.* W polu tym podaje się wartość stałej, ujętą w apostrofy. Wyjątek stanowią stałe adresowe, które należy ujmować w nawiasy.

Tablica 15 zawiera ważniejsze informacje o stałych w języku ASSEMBLER.

PRZYKŁAD 3

Podamy przykłady prawidłowo zdefiniowanych stałych. W przykładach tych pod każdą instrukcją podany jest obraz pola pamięci, w którym umieszczona będzie definiowana stała (1 kratka oznacza 1 bajt).

TABLICA 15

Instrukcje definiowania stałych

KOD	Typy stałych	Postać maszynowa	Długość domyślna	Długość maksymalna w bajtach	Granica umieszczenia w pamięci	Obcięcie (uzpełnienie)
C	znakowa	8-bitowy kod znaku	—	DC-256 DS-65336	bajt	z prawej strony
X	szestnastkowa	4-bitowy kod cyfry szestnastkowe	—	, jw.	bajt	z lewej strony
B	binarna	cyfry dwójkowe	—	256	bajt	jw.
F	stałoprzecinkowa	cyfry dziesiętne	4	8	słowo	jw.
H	stałoprzecinkowa	cyfry dziesiętne	2	8	półsłowo	jw.
E	zmiennoprzecinkowa	liczba dziesiętna zmiennoprzecinkowa krótka	4	8	słowo	z prawej strony
D	zmiennoprzecinkowa	liczba zmiennoprzecinkowa długa	8	8	słowo podwójne	jw.
P	dziesiętna	liczba dziesiętna spakowana		16	bajt	z lewej strony
Z	dziesiętna	liczba dziesiętna rozpakowana	—	16	bajt	jw.
A	adresowa	adres symboliczny	4		słowo	jw.
Y	adresowa	adres symboliczny	2		półsłowo	jw.
S	adresowa	adres symboliczny lub jawny	2		półsłowo	
V	adresowa	adres symboliczny zewnętrzny	4		słowo	z lewej strony

a) stała znakowa:

```
          ALFA    DC    C'SUMA='      (instrukcja)
ALFA    | S | U | M | A | = |          (pole pamięci)
```

b) stała znakowa z modyfikatorem długości:

```
          B1      DC    CL5'JEDNOLITY SYSTEM'
B1      | J | E | D | N | O |
```

Ponieważ zadana długość 5 znaków jest mniejsza od wartości stałej, do pamięci zapisuje się tylko pięć znaków z lewej strony.

c) stała znakowa:

```
          E       DC    C'''ILONA'''
E       | ' | I | L | O | N | A | ' |
```

Znaki apostrof (') i ampersand (&) należy zapisywać dwukrotnie obok siebie. Translator uzna taką parę za jeden znak.

d) stała znakowa:

```
          LIT     DC    CL8'PL'
LIT     | P | L | ß | ß | ß | ß | ß | ß |
```

Po translacji stała o nazwie **LIT** będzie miała wartość **PL** uzupełnioną spacjami z prawej strony.

e) stała znakowa:

```
          TTT     DC    3CL4'YES'
TTT     | Y | E | S | ß | Y | E | S | ß | Y | E | S | ß |
```

f) stała szesnastkowa:

```
          HEX     DC    X'2F11D'
HEX     | ø | 2 | F | 1 | 1 | D |
```

Stała szesnastkowa zajmuje w pamięci pole o długości równej połowie liczby znaków szesnastkowych. W wypadku nieparzystej liczby znaków szesnastkowych assembler działa tak, jak gdyby z lewej strony stałej został dopisany dodatkowy znak — zero szesnastkowe.

g) stała szesnastkowa z modyfikatorem długości;

```
                H6      DC      XL6'0251A'
H6          | 0 0 | 0 0 | 0 0 | 0 0 | 2 5 | 1 A |
```

Ponieważ modyfikator długości określa długość stałej na sześć bajtów, sześciobajtowe pole pamięci o nazwie **H6** uzupełniane jest z lewej strony zerami szesnastkowymi.

h) stała binarna:

```
                BIN     DC      BL2'10110101'
BIN         | 00000000 | 00101101 |
```

Długość stałej została określona na dwa bajty, czyli 16 bitów. Ponieważ zadana wartość stałej jest krótsza, zostanie więc uzupełniona z lewej strony zerami binarnymi.

i) stała binarna:

```
                DWOJ    DC      BL1'11011011'
DWOJ            | 11011011 |
```

Długość stałej wynosi w tym wypadku jeden bajt, a więc w procesie translacji dwa lewe bity '1Ø' zostaną pominięte.

j) stała stałoprzecinkowa:

```
                PI      DC      F'314'
PI          | 0000 0000 | 0000 0000 | 0000 0001 | 0011 1010 |   (postać binarna)
            |    0    0 |    0    0 |    0    1 |    3    A |   (postać szesnastkowa)
```

Długość stałej stałoprzecinkowej równa jest czterem bajtom (jedno słowo). Ponieważ zadana stała jest krótsza, słowo w pamięci dopełniane jest z lewej strony zerami.

k) stała stałoprzecinkowa:

```
                P1Ø     DC      H'-896'
P1Ø         | 1111 1100 | 1000 0000 |
            |    F    C |    8    0 |
```

Stała typu H ma długość równą jednemu półsłowu. Ujemną wartość assembler zapisuje jako uzupełnienie do dwóch;
l) stała dziesiętna spakowana:

```
                UPAK  DC  PL4'45'
UPAK    | Ø | Ø | Ø | Ø | Ø | 4 | 5 | C |
```

Ponieważ zdefiniowano długość stałej równą czterem bajtom, pozostałe wolne miejsce zapełnia się zerami. Pierwsza połowa prawego bajtu zawiera znak C, który oznacza znak +. Maksymalna długość stałej dziesiętnej wynosi 16 bajtów;
ł) stała dziesiętna rozpakowana:

```
                ZONA  DC  Z'-328'
ZONA    | F | 3 | F | 2 | D | 8 |
```

Stała o nazwie **ZONA** zajmuje w pamięci 3 bajty. Pierwszy bajt z prawej strony zawiera znak **D** oznaczający minus, pozostałe bajty zawierają strefy **F**.
m) stała adresowa:

```
POCZ  DC  A(ALFA)
```

Stała adresowa o nazwie **POCZ** otrzyma wartość równą adresowi zmiennej **ALFA**. Długość, jeśli nie występuje modyfikator długości, wynosi jedno słowo.

b. Definiowanie obszaru pamięci

Dla rezerwowania obszarów pamięci służy instrukcja z kodem operacji **DS** (ang. *Define Storage*). Format instrukcji **DS** jest podobny do instrukcji **DC**, z dwoma wyjątkami: 1 — nie występuje tu wartość stałej, 2 — maksymalna długość dla pól typu znakowego (**C**) i szesnastkowego (**X**) wynosi 63 535 bajtów.

Zadaniem instrukcji **DS** jest wyłącznie rezerwowanie i nadawanie nazwy określonym obszarom pamięci. Instrukcja **DS** nie zmienia natomiast w żaden sposób zawartości pamięci definiowanego obszaru. Do momentu umieszczenia przez program problemowy w rezerwowym obszarze pamięci określonych danych nieznana jest rzeczywista zawartość tego pola.

PRZYKŁAD 4

ZAPIS DS CL8Ø

Instrukcja ta zarezerwuje 80 kolejnych bajtów pamięci, tworząc pole o nazwie **ZAPIS**. Taki sam skutek można osiągnąć zapisując instrukcję **DS** inaczej, np:

ZAPIS DS 8ØX'ØØ'
ZAPIS DS 8CL1Ø itp.

Porównując instrukcje **DC** i **DS** warto zwrócić uwagę, że instrukcja

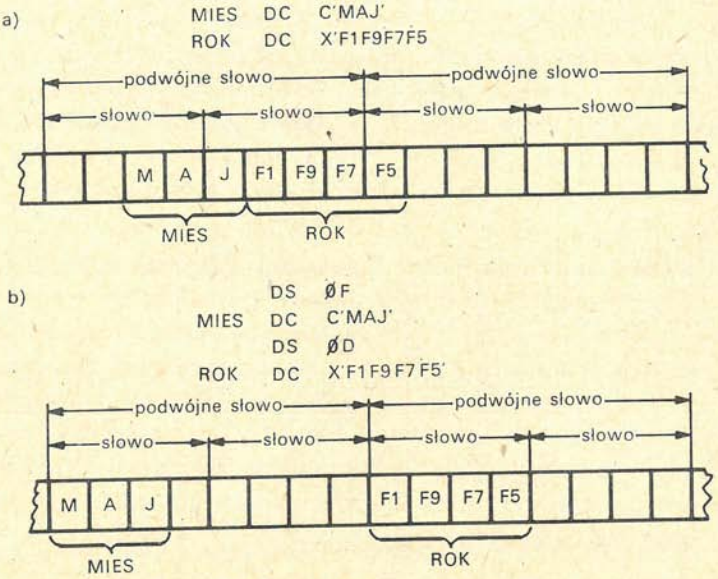
ZAPIS DC CL8Ø'Ø'

również spowoduje zarezerwowanie pola o długości 80 bajtów, lecz w odróżnieniu od instrukcji **DS** zapełni je zerami w postaci znakowej. ■

PRZYKŁAD 5

ALFA DS ØF

Instrukcja ta nie spowoduje zarezerwowania pola pamięci, może być jednak użyta w programie w celu ustawienia definiowanych stałych w pamięci



Rys. 42. Zawartość fragmentu pamięci

na granicy słowa. Rysunek 42a pokazuje zawartość fragmentu pamięci po wykonaniu instrukcji:

MIES DC C'MAJ'

ROK DC X'F1F9F7F5'

Stałą **MIES** można ustawić na granicy słowa za pomocą instrukcji **DS ØF** (por. rys. 42b). Natomiast instrukcja **DS ØD** ustawi stałą ROK na granicy podwójnego słowa.

3. Instrukcje maszynowe

Instrukcje maszynowe występują w postaci symbolicznej jako zadania języka ASSEMBLER. W rozdziale tym omówimy najczęściej używane instrukcje maszynowe, niektóre pomijając. Dla przykładu, pominiemy arytmetykę zmiennoprzecinkową, która jest ważnym rozdziałem języka ASSEMBLER, lecz jest dość kłopotliwa w użyciu. Pełną listę instrukcji maszynowych zawiera załącznik I.

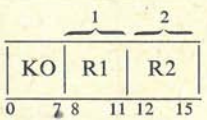
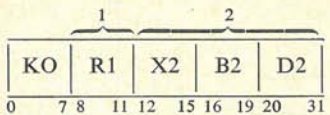
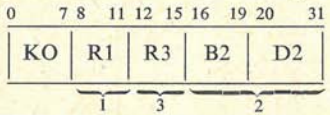
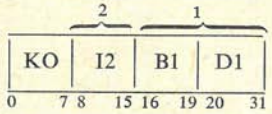
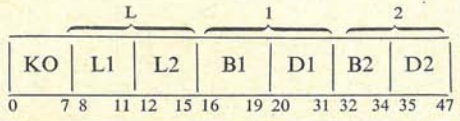
Formaty maszynowe instrukcji analizowaliśmy w rozdziale I. W arytmetyce stałoprzecinkowej jest regułą, że jeden z argumentów znajduje się w jednym z rejestrów uniwersalnych, drugi argument mieści się w innym rejestrze lub w pamięci operacyjnej. W rezultacie wykonania instrukcji w większości wypadków ustawia się w Słowie Stanu Programu (PSW) kod warunku, który będzie oznaczony literą **C** lub **CC** (ang. *Condition Code*). W dalszej części programu kod warunku może być użyty w instrukcjach skoków.

Tablica 16 przedstawia sposób zapisu instrukcji maszynowych w języku ASSEMBLER w zależności od typu instrukcji. Dla każdego typu instrukcji oprócz podstawowej formy zapisu (w której stosuje się mnemotechniczny kod instrukcji, a argumenty określa się jawnie za pomocą numerów rejestrów i przesunięć) podano sposób zapisu tych instrukcji z użyciem adresów symbolicznych. W zapisie tym litera S oznacza symbol, czyli ciąg liter i cyfr rozpoczynający się od litery, natomiast cyfry 1, 2, 3 oznaczają odpowiednio pierwszy, drugi lub trzeci argument. Instrukcje typu **RX** mogą więc mieć jedną z postaci:

KO R1,D2(X2,B2) lub *KO* R1,S2.

TABLICA 16

Instrukcje maszynowe języka ASSEMBLER uporządkowane według formatów

	Format maszynowy	Format assemblerowy	Instrukcje
RR		KO R1,R2	AR,ALR,BALR,BCR,BCTR,BASR, CLR,CR,DR,ISK,LCR,LNR, LPR,LR,LTR,MR,NR,OR,SSK,SLR, SR,XR
RX		KO R1,D2(X2,B2) KO R1,S2(X2) KO R1,D2(∅,B2) KO R1,S2	A,AH,AL,BAL,BC,BCT,C,CH,CL, CVB,CVD,D,EX,IC,L, LA,LH,M,MH,N,O,S,SH,SL,ST,STC, STH,X
RS		KO R1,R3,D2(B2) KO R1,R3,S2 KO R1,D2(B2) KO R1,S2	BXH,BXLE,LM,STM SLA,SLDA,SLDL,SLL,SRA,SRDA, SRDL,SRL
SI		KO D1(B1) KO S1 KO D1(B1),I2 KO S1,I2	LPSW,SSM,HIO,SIO,TIO,TCH,TS CLI,MVI,NI,OI,TM,XI
SS		KO D1(L,B1),D2(B2) KO S1(L),S2 KO D1(L1,B1),D2(L2,B2) KO S1(L1),S2(L2)	CLC,ED,EDMK,MVC,MVN,MVZ, NC,OC,TR,TRT,XC AP,CP,DP,MP,MVO,PACK,SP, UNPK,ZAP

Objaśnienia: KO-kod operacji, R1, R2, R3 — numery rejestrów pierwszego, drugiego i trzeciego argumentu, X1, X2 — numery rejestrów indeksowych, B1, B2, — numery rejestrów bazowych, D1, D2 — przesunięcia, L1, L2 — długości argumentów, I2 — bezpośredni argument, S1, S2 — adresy symboliczne.

Jeśli nie używa się rejestru indeksowego należy napisać instrukcję w następujący sposób:

KO R1,D2(Ø,B2),

natomiast gdy nie stosuje się zarówno rejestru X2 jak i B2, instrukcja przyjmuje postać:

KO R1,D2.

Podobnymi zasadami należy się kierować w wypadku instrukcji pozostałych formatów: jawne adresy argumentów D(X,B) oraz D(B) można zastąpić adresami symbolicznymi.

W czasie wykonywania instrukcji mogą wystąpić pewne szczególne sytuacje prowadzące najczęściej do przerwania programu. W rozdziale tym, w tabelkach, zawierających tematycznie zgrupowane instrukcje, będziemy podawać te sytuacje lub nieprawidłowości, używając przy tym następujących oznaczeń:

A — nieprawidłowa adresacja.

Wskazany w instrukcji adres przekracza dopuszczalne granice pamięci. Wykonywanie instrukcji przerywa się.

S — błędna specyfikacja.

Adres danych lub instrukcji nie jest całkowitą wielokrotnością jednostki informacji (półsłowa, słowa, lub podwójnego słowa, w zależności od rodzaju instrukcji).

W arytmetyce dziesiętnej długość dzielnika lub mnożnika przekracza 15 cyfr lub przy mnożeniu i dzieleniu długość pierwszego argumentu jest mniejsza lub równa drugiemu.

W instrukcjach, w których pierwszy argument wymaga pary rejestrów występuje rejestr nieparzysty.

We wszystkich tych wypadkach następuje przerwanie programu.

IF — nadmiar stałoprzecinkowy.

Występuje przeniesienie z najstarszej pozycji liczby stałoprzecinkowej do pozycji znakowej.

IK — nieprawidłowe dzielenie.

Przy dzieleniu stałoprzecinkowym iloraz nie mieści się w rejestrze wynikowym. Dotyczy to również dzielenia przez zero.

D — błąd danych.

Nieprawidłowy kod znaku lub cyfry w argumencie instrukcji arytmetyki dziesiętnej, redagowania i konwersji dwójkowej. Nieprawidłowe nakładanie się pól w instrukcjach arytmetyki dziesiętnej.

DF — przepełnienie dziesiętne.
Wynik nie mieści się w wyznaczonym dla niego polu.
DK — nieprawidłowe dzielenie dziesiętne.
Iloraz przekracza wyznaczone dla niego pole.
P — naruszenie ochrony pamięci.
Klucz ochrony pamięci pola, do którego ma być zapisany (przeczytany) wynik operacji nie jest równy kluczowi ochrony pamięci w PSW.

a. Instrukcje arytmetyczne

W instrukcjach arytmetycznych stałoprzecinkowych jeden argument znajduje się zawsze w rejestrze, natomiast drugi może być również w rejestrze lub w pamięci operacyjnej. W tym ostatnim wypadku wymagana jest odpowiednia adresacja — argument musi być zapisany na granicy słowa lub półsłowa, w zależności od typu konkretnej instrukcji.

Dodawanie

Lista rozkazów zawiera trzy instrukcje dodawania zwykłego i dwie instrukcje dodawania kodów, zwanego też dodawaniem logicznym:

Kod operacji	Symbol instrukcji	Format instrukcji	Nieprawidłowości	Kod warunku
1A	AR	R1,R2	IF	C
5A	A	R1,D2(X2,B2)	IF,P,A,S	C
4A	AH	R1,D2(X2,B2)	IF,P,A,S	C
1E	ALR	R1,R2		C
5E	AL	R1,D2(X2,B2)	P,A,S	C

Dodawanie zwykłe typu „rejestr—rejestr":

AR	R1, R2

Do zawartości rejestru R1 dodaje się zawartość rejestru R2. Wynik pozostaje w rejestrze R1, którego poprzednia wartość zostaje zniszczona.
Instrukcja ta powoduje ustawienie kodu warunku.

PRZYKŁAD 6

POLE NAZWY:	OPERACJA	ARGUMENTY:
1 5	9 10 15	20 25 30 35
E 1	A R	3 , 9

Zawartość rejestrów[2] przed wykonaniem tej instrukcji jest następująca:

R3 (+396)

0000 0000	0000 0000	0000 0001	1000 1100
0 0	0 0	0 1	8 C

R9 (+4001)

0000 0000	0000 0000	0000 1111	1010 0001
0 0	0 0	0 F	A 1

Po wykonaniu instrukcji w rejestrze 3 zostanie zapisana suma:

R3 (+4397)

0000 0000	0000 0000	0001 0001	0010 1101
0 0	0 0	1 1	2 D

Zawartość rejestru 9 nie ulegnie zmianie.
Dodawanie zwykłe typu „rejestr—pamięć":

A	R1, D2 (X2, B2)

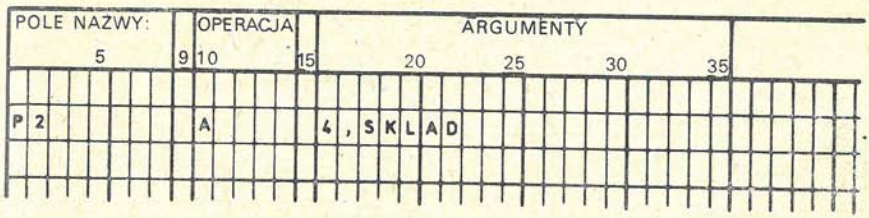
Zawartość słowa w pamięci głównej o adresie obliczonym w następujący sposób:

$$\text{adres} = (X2)+(B2)+D2$$

[2] W przykładach zamieszczonych w tym rozdziale podajemy w ramkach zawartość rejestrów lub pól pamięci w postaci dwójkowej i szesnastkowej albo tylko szesnastkowej (czytelnik łatwo odróżni obie formy zapisu). Jest zasadą, że każda pojedyncza kratka jest obrazem jednego bajtu.

sumuje się z zawartością rejestru R1. Wynik zostaje umieszczony w rejestrze R1. Drugi argument musi być zapisany w pamięci na granicy słowa. Po wykonaniu instrukcji system sprawdza czy nie wystąpiło przepełnienie rejestru.

PRZYKŁAD 7



Do zawartości rejestru 4 dodana zostanie zawartość słowa pamięci z obszaru o nazwie **SKLAD.** Argumenty te przedstawiamy:

R4 (+3361)

0000 0000	0000 0000	0000 1101	0010 0001
0 0	0 0	0 D	2 1

SKLAD (+2240)

...	0000 0000	0000 0000	0000 1000 1100 0000	...
	0 0	0 0	0 8 C 0	

Zawartość rejestru po wykonaniu instrukcji będzie następująca:

R4 (+5601)

0000 0000	0000 0000	0001 0101	1110 0001
0 0	0 0	1 5	E 1

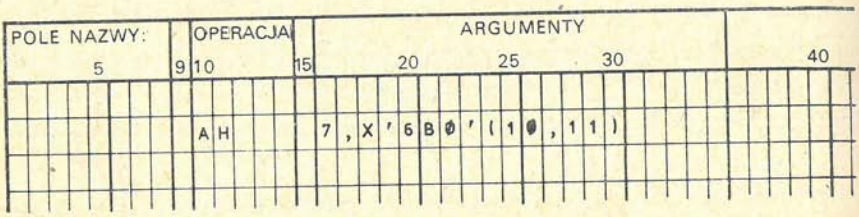
Argument znajdujący się w pamięci nie zmieni swojej wartości. ■

Dodawanie zwykłe półsłowa:

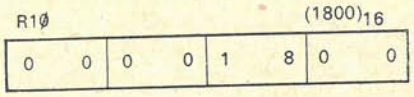
AH	R1, D2(X2, B2)

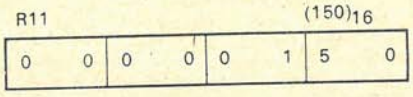
Do liczby zawartej w rejestrze R1 dodaje się dwubajtową liczbę zawartą w pamięci pod adresem wskazanym w części adresowej instrukcji, podobnie jak w instrukcji A. Przed wykonaniem operacji 16-bitowy drugi argument zostanie rozszerzony do 32 bitów przez uzupełnienie do pełnego słowa bitem znaku argumentu. Nie zmienia to zawartości pamięci. Drugi argument musi być zapisany na granicy półsłowa.

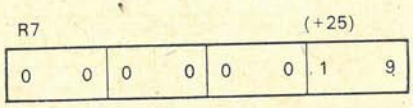
PRZYKŁAD 8



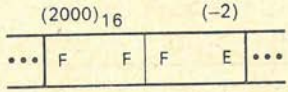
Zawartość rejestrów przed wykonaniem instrukcji:



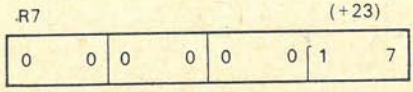




Argument drugi znajduje się w pamięci operacyjnej pod adresem $1800_{16}+150_{16}+6B\varnigma_{16} = 2000_{16}$:



Po wykonaniu instrukcji wynik zostanie umieszczony w rejestrze 7:



W tych instrukcjach **(AR,A,AH)** ustawia się następujący kod warunku:

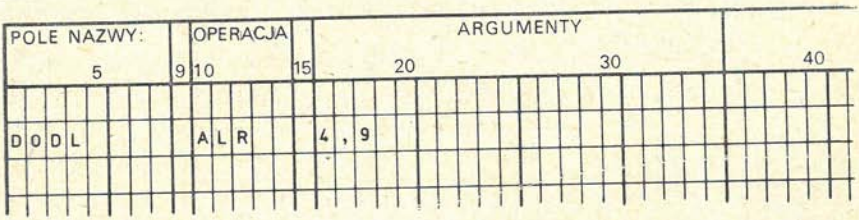
0 — suma równa zeru,
1 — wynik jest ujemny,
2 — wynik jest dodatni,
3 — nastąpił nadmiar rejestru.

Dodawanie kodów „rejestr—rejestr":

ALR	R1,R2

Do zawartości rejestru R1 (pierwszy argument) dodaje się zawartość rejestru R2 (drugi argument). Wynik pozostaje w R1. Udział w dodawaniu biorą wszystkie bity rejestrów łącznie z bitem znaku, zgodnie ze zwykłymi prawami dodawania liczb dwójkowych. W wypadku nadmiaru nie następuje przerwanie programowe, lecz jedynie ustawia się kod warunku, sygnalizujący przeniesienie ze znakowej pozycji.

PRZYKŁAD 9



Zawartości rejestrów występujących w tej instrukcji są następujące:

R4 (–3596)

F	F	F	F	F	1	F	4

R9 (–398541)

F	F	F	9	E	B	3	3

Wynikiem jest kod dwójkowy, znajdujący się w rejestrze 4:

R4 (–402137)

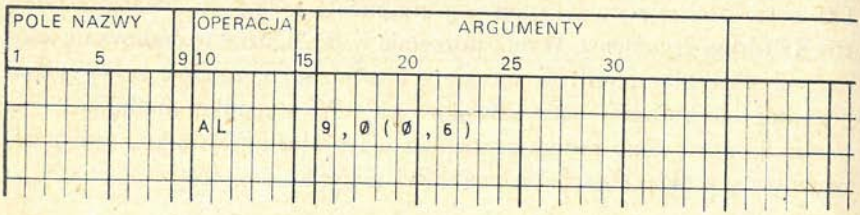
F	F	F	9	D	D	2	7

Dodawanie kodów „rejestr—pamięć":

AL	R1,D2(X2,B2)

Pierwszy argument (R1), znajdujący się w rejestrze, dodawany jest do drugiego argumentu, znajdującego się w pamięci operacyjnej. Wynik pozostaje w rejestrze R1. Drugi argument musi znajdować się w pamięci na granicy słowa. Dodawane są do siebie wszystkie bity argumentów łącznie ze znakiem (argumentów nie traktuje się jako liczby, lecz jako pewne kody). Nadmiar nie powoduje przerwania, lecz ustawienie odpowiedniego kodu warunku.

PRZYKŁAD 10



Argumenty mają przed wykonaniem instrukcji następujące wartości:

R9 (+1895840783)

0111	0001	0000	0000	0011	1100	0000	1111
7	1	0	0	3	C	0	F

Ø(Ø,6) (+255267216)

...	0000	1111	0011	0111	0001	0001	1001	0000	...
	0	F	3	7	1	1	9	0	

Po wykonaniu instrukcji otrzymujemy wynik w rejestrze 9:

R9 (−2143859297)

1000	0000	0011	0111	0100	1101	1001	1111
8	0	3	7	4	D	9	F

W wyniku wykonania instrukcji dodawania kodów ustawia się jeden z następujących kodów warunku:

0 — suma równa zeru (nadmiar nie wystąpił),

1 — suma różna od zera (nadmiaru nie było),

2 — suma równa zeru (wystąpił nadmiar),
3 — suma różna od zera (wystąpił nadmiar).

Odejmowanie

Istnieje pięć typów instrukcji odejmowania:

Kod operacji	Symbol instrukcji	Format instrukcji	Nieprawidłowości	Kod warunku
1B	**SR**	R1,R2	IF	C
5B	**S**	R1,D2(X2,B2)	IF,P,A,S	C
4B	**SH**	R1,D2(X2,B2)	IF,P,A,S	C
1F	**SLR**	R1,R2		C
5F	**SL**	R1,D2(X2,B2)	P,A,S	C

Instrukcje te mają następujące nazwy:
Odejmowanie zwykłe „rejestr—rejestr" **(SR),**
Odejmowanie zwykłe „rejestr—pamięć" **(S),**
Odejmowanie zwykłe półsłowa **(SH),**
Odejmowanie kodów „rejestr—rejestr" **(SLR),**
Odejmowanie kodów „rejestr—pamięć" **(SL).**

W instrukcjach odejmowania od pierwszego argumentu odejmuje się drugi argument, a wynik zostaje zachowany na miejscu pierwszego argumentu. Poza tym instrukcje odejmowania są analogiczne do odpowiednich instrukcji dodawania.

Kod warunku w instrukcjach odejmowania zwykłego **(SR,S,SH)** jest taki sam, jak w instrukcjach dodawania. Natomiast w odejmowaniu kodów istnieje w analogii z odpowiednimi instrukcjami dodawania jeden wyjątek: w wypadku różnicy równej zeru kod warunku jest nieokreślony.

Mnożenie

Lista rozkazów zawiera trzy instrukcje mnożenia stałoprzecinkowego:

Kod operacji	Symbol instrukcji	Format instrukcji	Nieprawidłowości	Kod warunku
1C	**MR**	R1.R2	S	
5C	**M**	R1,D2(X2,B2)	S,P,A	
4C	**MH**	R1,D2(X2,B2)	S,P,A	

Mnożenie „rejestr—rejestr":

MR	R1,R2

Mnożenie wymaga pary rejestrów dla wyniku, ze względu na możliwą dużą wartość iloczynu. R1 wyznacza mnożną. Musi to być rejestr z parzystym numerem. Mnożna natomiast znajduje się w najbliższym nieparzystym rejestrze. Drugi argument to mnożnik, znajdujący się w dowolnym innym rejestrze. Iloczyn pozostaje w parze rejestrów pierwszego argumentu. Jest to liczba stałoprzecinkowa składająca się z bitu znaku i 63 bitów liczby. Znak i bardziej znaczące cyfry znajdują się w parzystym rejestrze, młodsze pozycje iloczynu — w nieparzystym rejestrze.

PRZYKŁAD 11

POLE NAZWY	OPERACJA	ARGUMENTY
1 5	9 10 15	20 30
MNOŻMR	MR	4,9

W tej instrukcji mnożna znajduje się w rejestrze 5, natomiast zawartość rejestru jest dowolna:

R4				? R5				(+3276)	
?	?	?	?	0	0	0	0	C	C

Mnożnik znajduje się w rejestrze 9:

R9				(+991)	
0	0	0	0	D	F

Po wykonaniu instrukcji iloczyn znajduje się w rejestrach 4—5:

R4						R5				(+3246516)	
0	0	0	0	0	0	0	0	3	1	8	9	B	4

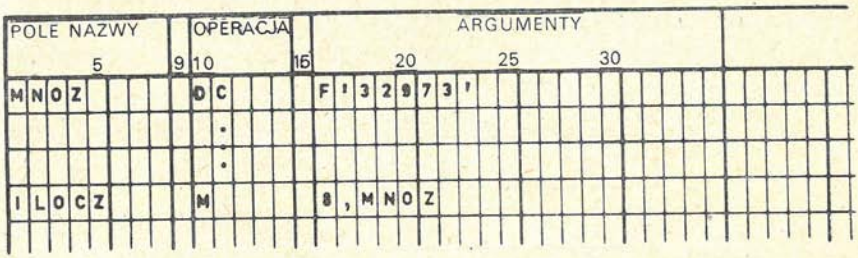
Mnożenie „rejestr—pamięć":

M	R1,D2(x2,B2)

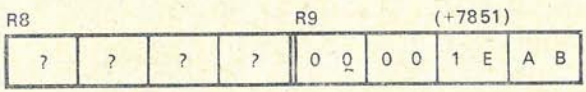
Operacja ta przebiega podobnie jak mnożenie **MR**, z tą różnicą, że drugi argument jest czterobajtowym polem pamięci, umieszczonym na granicy słowa.

PRZYKŁAD 12

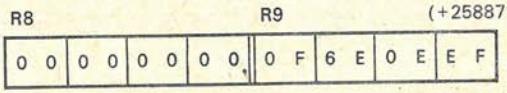
Rozważmy instrukcje:



W tym przykładzie mnożenia mnożna znajduje się w rejestrze 9:



Mnożnik określony instrukcją **DC** znajduje się w pamięci w polu o nazwie **MNOZ.** Wynik zostanie zapisany do rejestru 8 i 9:



Ponieważ w tym wypadku iloczyn mieści się całkowicie w jednym rejestrze, w dalszych obliczeniach można posługiwać się jako wynikiem zawartością rejestru 9. Należy jednak pamiętać, że w ogólnym wypadku iloczyn zajmuje dwa rejestry. ■

Mnożenie półsłowa:

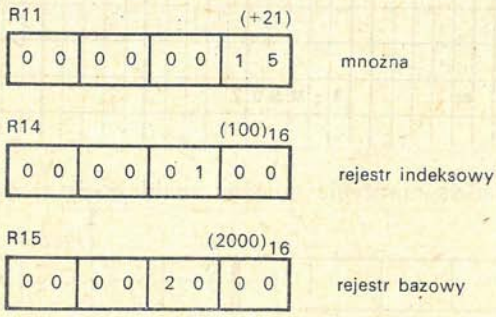
MH	R1,D2(X2,B2)

Za pomocą instrukcji **MH** można pomnożyć liczbę zawartą w rejestrze przez liczbę dwubajtową, przechowywaną w pamięci. Wynik pozostaje w rejestrze. Możliwy jest w tym wypadku nadmiar rejestru, lecz nie powoduje to przerwania programu.

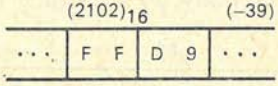
PRZYKŁAD 13

```
POLE NAZWY:   OPERACJA   ARGUMENTY
1     5       9 10   15  20    25    30
              MH       11,2(14,15)
```

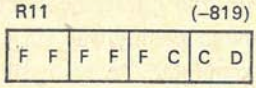
Zawartość rejestrów:



Mnożnik znajduje się w pamięci pod adresem 2102_{16}:



Po wykonaniu mnożenia iloczyn zostanie zapisany w rejestrze 11:



■

Dzielenie

Dwie możliwe instrukcje dzielenia stałoprzecinkowego przedstawia następująca tabliczka:

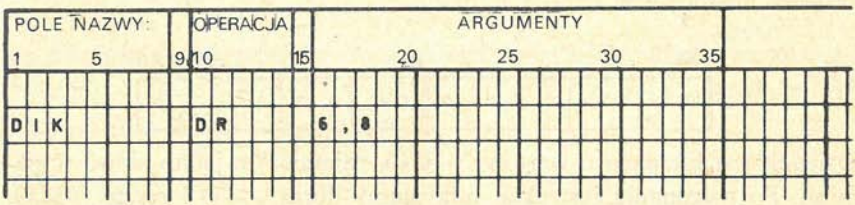
Kod operacji	Symbol instrukcji	Format instrukcji	Nieprawidłowości	Kod warunku
1D	**DR**	R1,R2	IK,S	
5D	**D**	R1,D2(X2,B2)	IK,S,A,P	

Dzielenie „rejestr—rejestr":

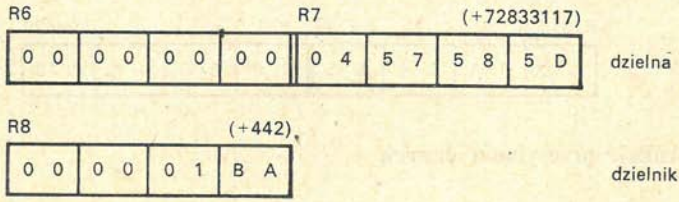
DR	R1, R2

Instrukcja dzielenia **DR** powoduje podzielenie zawartości pary rejestrów, z których pierwszy (R1) musi być parzysty, przez dzielnik znajdujący się w dowolnym innym rejestrze (R2). Dzielną jest więc liczba 64-bitowa ze znakiem. Po zakończeniu instrukcji dzielenia wynik zapisuje się w dwóch rejestrach pierwszego argumentu: część całkowita ilorazu mieści się w rejestrze z numerem nieparzystym, reszta z dzielenia — w rejestrze parzystym. Kod warunku nie ustawia się.

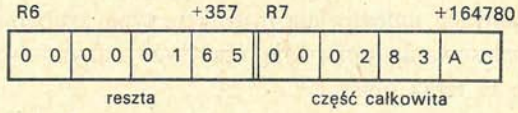
PRZYKŁAD 14



Przypuśćmy, że rejestry zapełnione są w następujący sposób:



Liczba +72933117 zawarta w parze rejestrów 6 i 7 zostanie podzielona przez liczbę +442, znajdującą się w rejestrze 8. W wyniku tej instrukcji część całkowita ilorazu zostanie zapisana w rejestrze 7, natomiast do rejestru 6 wpisze się reszta z dzielenia:



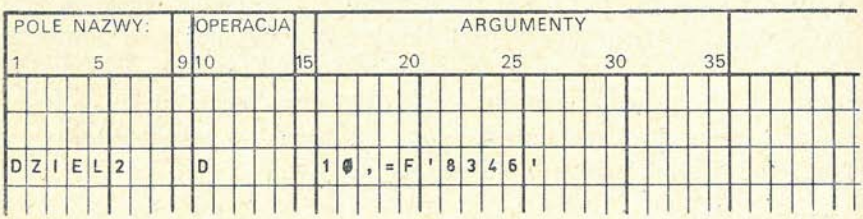
■

Dzielenie „rejestr—pamięć":

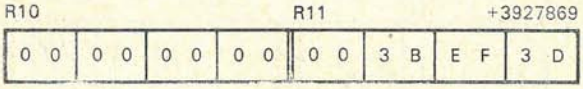
D	R1, D2(X2, B2)

Instrukcja dzielenia **D** różni się od przedstawionej instrukcji **DR** tylko tym, że dzielnik mieści się w słowie pamięci głównej. Wynik również pozostaje w rejestrach.

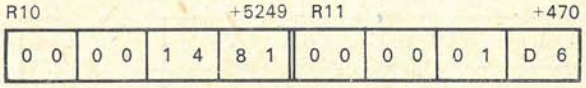
PRZYKŁAD 15



Dzielna mieści się w rejestrach 10 i 11:



Dzielnik jest literalem o wartości +8346, zajmującym jedno słowo w pamięci. Po wykonaniu instrukcji otrzymamy iloraz +470 i resztę +5249, rozmieszczone w rejestrach 10 i 11:



b. Instrukcje przesyłania danych

W programach problemowych powstaje często konieczność przesyłania danych z jednego obszaru pamięci do drugiego. Instrukcje przesyłania powodują przemieszczenie danych z pewnego pola pamięci głównej do pola o innym adresie. Instrukcje ładowania przesyłają dane z pamięci do rejestrów lub z rejestrów do rejestrów. Zapisanie w pamięci danych zawartych w rejestrach umożliwiają instrukcje typu „zapamiętaj". Wreszcie instrukcje przesuwania pozwalają operować danymi wewnątrz rejestrów. Instrukcje te teraz opiszemy.

Instrukcje przesyłania

Kod operacji	Symbol instrukcji	Format assemblerowy	Nieprawidłowości	Kod warunku
92	MVI	D1(B1),I2	P,A	
D2	MVC	D1(L,B1),D2(B2)	P,A	
D1	MVN	D1(L,B1),D2(B2)	P,A	
D3	MVZ	D1(L,B1),D2(B2)	P,A	

Przesyłka bezpośrednia:

MVI	D1(B1),I2

Za pomocą instrukcji **MVI** można przesłać do określonego miejsca pamięci dowolny znak (1 bajt). Bajt przeznaczony do przesyłki podany jest bezpośrednio w instrukcji jako stała.

PRZYKŁAD 16

```
POLE NAZWY      OPERACJA        ARGUMENTY
1    5      9  10      15  20        25        30        35
           MVI     230(0),X'A1'
```

Instrukcja ta spowoduje zapisanie w pamięci głównej pod adresem 230 = $= E6_{16}$ ośmiu bitów 10100001, czyli liczby $A1_{16}$.

Przesyłka znaków

MVC	D1(L,B1),D2(B2)

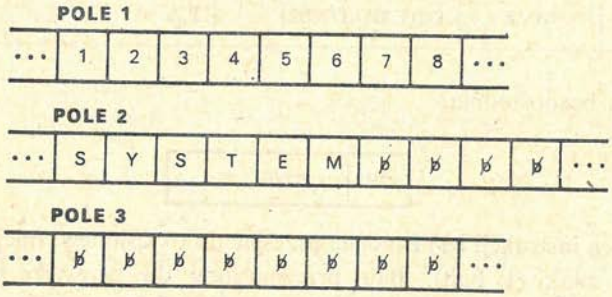
Dowolna dana o długości nie przekraczającej 256 bajtów może być przesłana z jednego pola pamięci (argument drugi) do innego (argument pierwszy) za pomocą instrukcji **MVC**. Liczba bajtów przesłanych w konkretnej operacji określona jest w części adresowej instrukcji (L). Oba pola pamięci biorące udział w operacji mają tę samą długość. Dane przesyła się bajt po bajcie, od lewej strony pola do prawej.

PRZYKŁAD 17

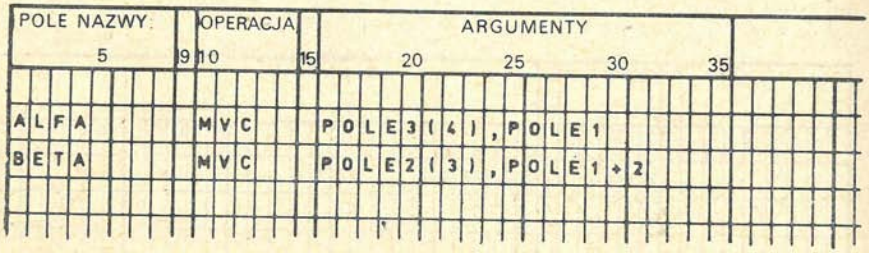
Przypuśćmy, że w programie wystąpiły instrukcje:

POLE1 DC C'12345678'
POLE2 DC CL1 Ø 'SYSTEM'
POLE3 DC 8C'ϕ'

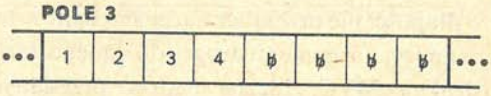
W rezultacie do pamięci zostały wprowadzone stałe:



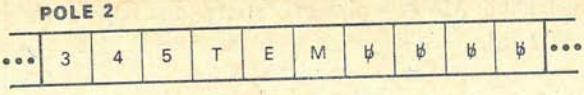
Następnymi instrukcjami niech będą:



Instrukcja **ALFA** spowoduje przesłanie danych z pola pamięci z adresem początkowym określonym przez nazwę **POLE1** do pola o nazwie **POLE3.** Liczbę przesłanych bajtów określa wskaźnik długości pierwszego argumentu (w tym wypadku 4). W rezultacie instrukcja ta nie powoduje żadnych zmian zawartości pola **POLE1,** a obszar pamięci **POLE3** będzie zawierał następujące znaki:



W wyniku instrukcji **BETA** pole długości trzech bajtów o adresie obliczonym jako suma adresu odpowiadającego nazwie **POLE1** i liczby 2 zostanie przesłane do pola **POLE2:**



Przesyłanie cyfr:

MVN	D1(L,B1),D2(B2)

Instrukcja **MVN** powoduje przesłanie 4 prawych bitów każdego bajtu drugiego argumentu na miejsce 4 prawych bitów odpowiednich bajtów w pierwszym argumencie. Lewe połówki bajtów zostają nie zmienione. Obydwa argumenty znajdują się w pamięci głównej.

Prawe cztery bity każdego bajtu w liczbach zapisanych w postaci rozpakowanej oznaczają wartości cyfr. System nie sprawdza jednak, czy są to cyfry od 0 do 9, czy też cyfry szesnastkowe A,...,F.
Maksymalnie można przesłać do 256 bajtów.
Przesyłanie stref:

MVZ	D1(L,B1),D2(B2)

Instrukcja **MVZ** działa podobnie jak instrukcja **MVN**, którą już rozpatrywaliśmy, z tym że powoduje przesłanie lewych czterech bitów każdego bajtu. W liczbach przedstawionych w postaci strefowej bity te noszą właśnie nazwę strefy i równe są 1111. Przesyłanie takich stref jest głównym przeznaczeniem tej instrukcji, ale ponieważ translator nie kontroluje wartości przesyłanych danych można użyć instrukcji **MVZ** do przesyłania dowolnych bitów. Dane przesyła się po jednym półbajcie z lewej strony poczynając. Maksymalnie można przesłać 256 bajtów (a raczej półbajtów).

PRZYKŁAD 18

Przypuśćmy, że w pamięci głównej pod adresem 800_{16} znajduje się pole zawierające:

$(800)_{16}$

1	B	3	D	Q	K	znaki
F1	C2	F3	C4	D8	D2	cyfry szesnastkowe

Niech rejestr 12 zawiera adres 800_{16}:

R12 $(800)_{16}$

0 0	0 0	0 8	0 0

Za pomocą instrukcji **MVZ** wpiszemy strefy do bajtów w pamięci o adresach 801, 802 itd.

POLE NAZWY:	OPERACJA	ARGUMENTY
1 5	9 10 15	20 25 30
	MVZ	Ø(5,12),1(12)

W rezultacie otrzymamy:

$(800)_{16}$

A	2	C	M	Q	K
C1	F2	C3	D4	D8	D2

Instrukcje ładowania

Instrukcje ładowania służą do wprowadzania (ładowania) danych do rejestrów. Ładować do rejestrów można od 1 do 4 bajtów danych, którymi mogą być zarówno liczby, jak i adresy.

Kod operacji	Symbol instrukcji	Format assemblerowy	Nieprawidłowości	Kod warunku
18	**LR**	R1,R2		
58	**L**	R1,D2(X2,B2)	P,A,S	
48	**LH**	R1,D2(X2,B2)	P,A,S	
98	**LM**	R1,R3,D2(B2)	P,A,S	
41	**LA**	R1,D2(X2,B2)		
12	**LTR**	R1,R2		C
43	**IC**	R1 D2(X2,B2)	P,A	
13	**LCR**	R1,R2	IF	C
11	**LNR**	R1,R2		C
10	**LPR**	R1,R2	IF	C

Ładowanie „rejestr—rejestr":

LR	R1, R2

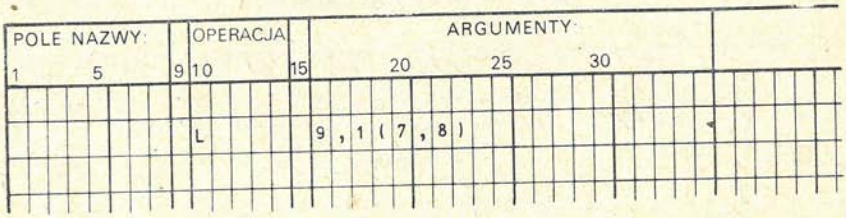
Instrukcja **LR** powoduje przesłanie danych z rejestru zawierającego drugi argument do rejestru z pierwszym argumentem. Po wykonaniu instrukcji zawartość obu rejestrów będzie jednakowa. Instrukcja ta nie wpływa na kod warunku.

Ładowanie „pamięć—pamięć":

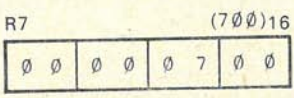
L	R1, D2(X2, B2)

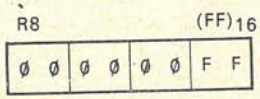
W wyniku instrukcji **L** argument drugi, który jest słowem w pamięci głównej, zostanie przesłany do rejestru R1.

PRZYKŁAD 19

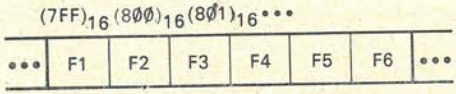


Jeśli przed wykonaniem tej instrukcji rejestry miały wartości:

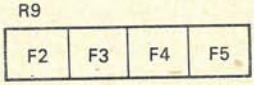




a w pamięci znajdowały się następujące dane:



wówczas po wykonaniu tej instrukcji w rejestrze 9 znajdą się dane przesłane z pamięci:



Oczywiście, jest rzeczą zupełnie nieistotną, jaką zawartość miał rejestr 9 przed wykonaniem instrukcji. ■

Ładowanie półsłowa:

LH	R1,D2(X2,B2)

Instrukcja **LH** stwarza możliwość przesłania danej o długości półsłowa z pamięci do rejestru R1. Drugi argument musi być daną umieszczoną w pamięci na granicy półsłowa. Pobrane z pamięci dwa bajty są rozszerzane w rejestrze do pełnego słowa w ten sposób, że lewa połowa rejestru wypełniana jest znakowym bitem przesyłanego półsłowa. Instrukcji tej, jak również instrukcji ładowania **L**, używa się często do wpisywania stałych do rejestru.

PRZYKŁAD 20

Przypuśćmy, że pole o nazwie AAA ma wartość X'F1FØ'. Zawartość rejestru 13 jest dowolna. Po wykonaniu instrukcji:

POLE NAZWY:	OPERACJA	ARGUMENTY
1 5	9 10 15	20 25 30
	LH	13,AAA

w rejestrze 13 znajdzie się następująca dana przeniesiona z pamięci:

AAA

F 1	F Ø

R13

1111 1111	1111 1111	1111 ØØØ1	1111 ØØØØ
F F	F F	F 1	F Ø

■

Ładowanie grupowe:

LM	R1,R3,D2(B2)

Instrukcja **LM** zapisuje zawartość pola pamięci, o adresie określonym w instrukcji, do szeregu kolejnych rejestrów. Pierwszym z tych rejestrów jest R1, ostatnim R3. Szereg rejestrów tworzony jest cyklicznie, tzn. po rejestrze 15 następuje 0, 1, 2, 3 itd.

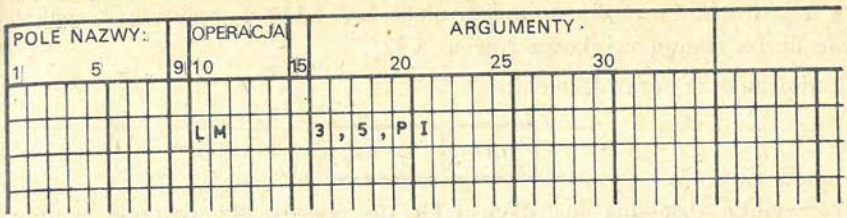
PRZYKŁAD 21

Niech pole pamięci **PI** zawiera następujące dane:

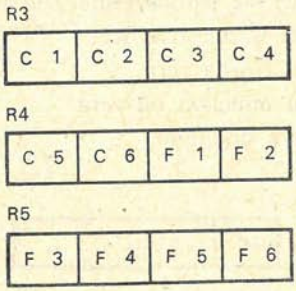
PI

...	C1	C2	C3	C4	C5	C6	F1	F2	F3	F4	F5	F6	...

Instrukcja



spowoduje załadowanie do rejestrów 3, 4, 5 następujących danych:



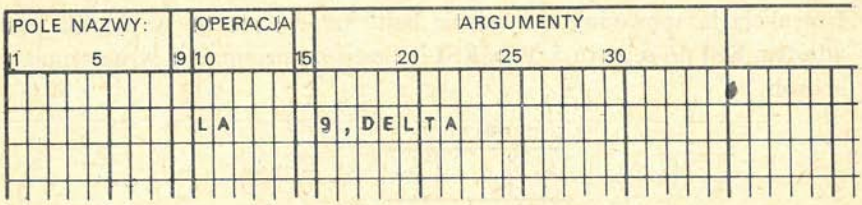
Ładowanie adresu:

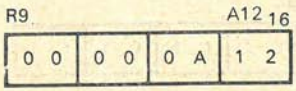
LA	R1,D2(X2,B2)

Za pomocą instrukcji **LA** można załadować do rejestru R1 24-bitowy adres drugiego argumentu. Pozostałe osiem bitów rejestru uzupełnia się zerami.

PRZYKŁAD 22

a. Przypuśćmy, że pole pamięci o nazwie **DELTA** ma adres $A12_{16}$. Liczba ta zostanie zapisana za pomocą tej instrukcji do rejestru 9:





b. Instrukcja **LA** jest bardzo przydatna przy ustawianiu małych liczb w rejestrach. Po wykonaniu instrukcji **LA 4,325** w rejestrze 4 znajdzie się liczba stałoprzecinkowa równa +325. ■

Ładowanie ze sprawdzeniem:

LTR	R1, R2

W wyniku działania instrukcji **LTR** do rejestru R1 zostanie wpisana zawartość rejestru R2. Po wykonaniu instrukcji zawartości obu rejestrów będą identyczne. Ustawi się jednocześnie kod warunku w zależności od znaku nowej zawartości w rejestrze R1:

0 — wartość rejestru równa zeru,
1 — wartość rejestru mniejsza od zera,
2 — liczba w rejestrze dodatnia.

Wstawianie znaku:

IC	R1,D2(X2,B2)

Instrukcja **IC** powoduje wstawienie jednobajtowego argumentu z pamięci głównej do rejestru R1. Wpisany bajt zajmuje osiem najmniej znaczących bitów rejestru, a pozostałe bity nie ulegają zmianie.

PRZYKŁAD 23

POLE NAZWY:	OPERACJA	ARGUMENTY
1 5	9 10	20 25 30 35
W S T	I C	S , K S I

Instrukcja ta spowoduje wstawienie bajtu umieszczonego w pamięci pod adresem **KSI** do rejestru 5. Pole **KSI** i rejestr 5 zapełnione są w następujący sposób:

KSI

...	F F	...

R5

C 1	C 1	C 1	C 1

Po wykonaniu instrukcji w rejestrze 5 znajdzie się następująca liczba:

R5

C 1	C 1	C 1	F F

■

Instrukcje zapisywania w pamięci

Instrukcje tego typu przypominają swym działaniem instrukcje ładowania, z tym że kierunek przesyłania jest przeciwny — z rejestru do pamięci.

Kod operacji	Symbol instrukcji	Format assemblerowy	Nieprawidłowości	Kod warunku
50	**ST**	R1,D2(X2,B2)	P,A,S	
42	**STC**	R1,D2(X2,B2)	P,A	
40	**STH**	R1,D2(X2,B2)	P,A,S	
90	**STM**	R1,R3,D2(B2)	P,A,S	

Instrukcje te umożliwiają zapisywanie w pamięci zawartości rejestrów uniwersalnych. Kod warunku nie ulega przy tym zmianie.

Zapamiętaj:

ST	R1,D2(X2,B2)

Instrukcja **ST** powoduje przeniesienie zawartości rejestru uniwersalnego R1 do pamięci głównej w miejsce wyznaczone przez adres E2 = (X2)+ +(B2)+D2 lub adres symboliczny.

Zapamiętaj znak:

STC	R1,D2(X2,B2)

Do pola pamięci o adresie podanym w instrukcji przesłany zostaje jeden bajt z rejestru R1 (8 najmniej znaczących bitów).

Zapamiętaj półsłowo:

STH	R1,D2(X2,B2)

Prawa połowa rejestru R1 zostaje przeniesiona do pola pamięci o adresie wskazanym w instrukcji.

Zapamiętaj grupę rejestrów:

STM	R1,R3,D2(B2)

Stosując instrukcje **STM** możemy zapamiętać w pamięci głównej zawartość kilku kolejnych rejestrów. Pierwszym rejestrem jest R1, ostatnim rejestr R3. Adres pola pamięci, do którego wpisywane są dane, określany jest w instrukcji jako adres drugiego argumentu.

PRZYKŁAD 24

W rejestrze 11 zapamiętana jest liczba:

R11

F 1	F 2	F 3	F 4

POLE NAZWY: (1–9)	OPERACJA (10–15)	ARGUMENTY (16–)
	DC	F
OMEGA	DC	8C' '
I1	ST	11,OMEGA
I2	STH	11,OMEGA+6
I3	STC	11,OMEGA

Podczas wykonywania tej sekwencji instrukcji zawartość pola OMEGA będzie zmieniać się w następujący sposób:

a) po wykonaniu instrukcji 11:

OMEGA

•••	F 1	F 2	F 3	F 4	4 0	4 0	4 0	4 0	•••

b) po wykonaniu instrukcji 12:

OMEGA

•••	F 1	F 2	F 3	F 4	4 0	4 0	F 3	F 4	•••

c) po wykonaniu instrukcji 13:

OMEGA

•••	F 4	F 2	F 3	F 4	4 0	4 0	F 3	F 4	•••

■

Do przesuwania zawartości rejestrów w prawo lub w lewo służą instrukcje przesuwania. Udział w przesuwaniu biorą wszystkie bity wskazanego w instrukcji rejestru. Wielkość przesunięcia określa adres drugiego argumentu. Wszystkie instrukcje przesuwania mają format **RS**, w którym nie używa się trzeciego argumentu.

Kod operacji	Symbol instrukcji	Format assemblerowy	Nieprawidłowości	Kod warunku
89	**SLL**	R1,D2(B2)		
8D	**SLDL**	R1,D2(B2)	S	
88	**SRL**	R1,D2(B2)		
8C	**SRDL**	R1,D2(B2)	S	
8B	**SLA**	R1,D2(B2)	IF	C
8F	**SLDA**	R1,D2(B2)	S,IF	C
8A	**SRA**	R1,D2(B2)		C
8E	**SRDA**	R1,D2(B2)	S	C

Przesuwanie kodów. Przesuń kod w lewo — **(SLL)**
Przesuń kod podwójny w lewo — **(SLDL)**
Przesuń kod w prawo — **(SRL)**
Przesuń kod podwójny w prawo — **(SRDL).**
Instrukcja **SLL** powoduje przesunięcie 32 bitów rejestru zawierającego pierwszy argument w lewo, a instrukcja **SRL** — w prawo. Podobnie działają instrukcje **SLDL** i **SRDL**, które powodują przesunięcie 64 bitów, odpowiednio w lewo, lub w prawo, czyli w tym wypadku przesunięciu ulega zawartość dwóch sąsiednich rejestrów. Instrukcja wskazuje pierwszy z tych rejestrów i musi być to rejestr o parzystym numerze.

Wszystkie cztery wymienione instrukcje nie wykorzystują wartości drugiego argumentu, używają natomiast jego adresu w celu określenia wielkości przesunięcia, czyli liczby pozycji binarnych, o jaką ma być przesunięta zawartość rejestrów. Liczbą tą jest sześć najmniej znaczących pozycji adresu, obliczonego według normalnych zasad. Bity, które w wyniku przesunięcia przekraczają granicę rejestru, tracą swoją wartość, ale nie powoduje to przerwania programu. Taka sama liczba bitów, jaka opuszcza rejestr, zostaje wpisana z przeciwnej strony na zwolnione pozycje w postaci bitów zerowych.

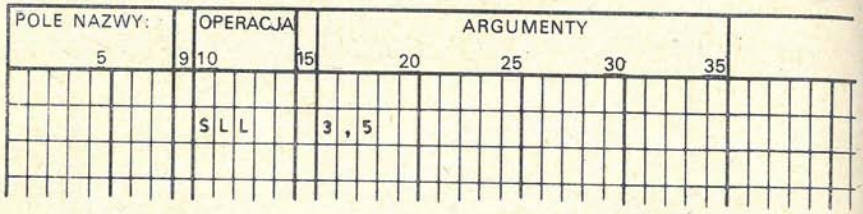
PRZYKŁAD 25

Przypuśćmy, że w rejestrze 3 znajduje się następująca liczba:

R3

8 7	6 5	4 3	2 1
1000 0111	0110 0101	0100 0011	0010 0001

Po wykonaniu rozkazu:



zawartość rejestru 3 przesunie się o pięć bitów w lewo:

R3

E C	A 8	6 4	2 0
1110 1100	1010 1000	0110 0100	0010 0000

10000
(utracone bity)

00000
(dopisane zera)

Za pomocą instrukcji przesuwających można liczbę zapisaną w rejestrze dzielić lub mnożyć przez liczbę równą 2^n, gdzie $n = 1,2,...,64$ — liczba pozycji, o którą należy dokonać przesunięcia. Na przykład w wyniku działania instrukcji

SRL 11,3

zawartość rejestru 11 zostanie podzielona przez 8, ponieważ przesunięcie w prawo o trzy pozycje dwójkowe jest równoznaczne z podzieleniem przez 2^3. Tracimy przy tym wartość reszty, którą są trzy bity wysunięte z rejestru z prawej strony.

Przesuwanie arytmetyczne

Przesuń w lewo arytmetycznie:

SLA	R1,D2(B2)

Przesuń w lewo arytmetycznie podwójnie:

SLDA	R1,D2(B2)

Instrukcje przesuwania arytmetycznego działają podobnie jak instrukcje przesuwania kodów, z tą różnicą, że w przesuwaniu nie bierze udziału bit znakowy. Po wykonaniu tych instrukcji ustawia się kod warunku:

0 — zawartość rejestru równa zeru,
1 — liczba w rejestrze mniejsza od zera,
2 — rejestr zawiera liczbę dodatnią,
3 — podczas przesuwania wystąpił nadmiar.

Przesuń w prawo arytmetycznie:

SRA	R1,D2(B2)

Przesuń w prawo arytmetycznie podwójnie:

SRDA	R1,D2(B2)

Dwie te instrukcje działają według tych samych zasad, lecz przesuwają w prawo. Zwolnione pozycje z lewej strony dopełniane są bitami o wartości takiej samej, jaką ma znak. W rezultacie ustawia się kod warunku:

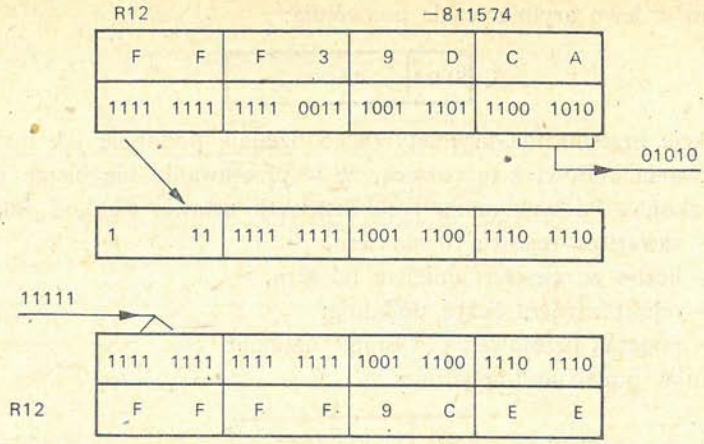
0 — zawartość rejestru po przesunięciu równa zeru,
1 — rejestr zawiera wartość ujemną,
2 — w rejestrze pojawiła się liczba dodatnia.

PRZYKŁAD 26

W rejestrze 12 znajduje się liczba −811574. Należy ją przesunąć o 5 pozycji w prawo. Dokonać tego można za pomocą instrukcji:

POLE NAZWY	OPERACJA	ARGUMENTY
5	9 10 15	20 25 30
	S R A	1 2 , 5

W wyniku tej instrukcji zawartość rejestru 12 przesunie się o pięć pozycji w prawo. Pięć najmniej znaczących bitów tracimy, a na zwolnionych pięć najbardziej znaczących miejsc wpisze się bit znaku (w tym wypadku jedynka).



c. Instrukcje porównania

Zadaniem instrukcji porównania jest ustawienie w **PSW** kodu warunku w zależności od relacji, jaka zachodzi między porównywanymi danymi:

0 — argumenty są równe,
1 — pierwszy argument jest mniejszy od drugiego,
2 — pierwszy argument jest większy od drugiego.

Żadnych zmian wartości argumentów instrukcje te nie powodują.

Kod operacji	Symbol instrukcji	Format assemblerowy	Nieprawidłowości	Kod warunku
15	CLR	R1,R2		C
55	CL	R1,D2(X2,B2)	P,A,S	C
95	CLI	D1(B1),I2	P,A	C
D5	CLC	D1(L,B1),D2(B2)	P,A	C
19	CR	R1,R2		C
59	C	R1,D2(X2,B2)	P,A,S	C
49	CH	R1,D2(X2,B2)	P,A,S	C

Porównywanie kodów „rejestr—rejestr":

CLR	R1,R2

Porównywanie kodów „rejestr—pamięć":

CL	R1,D2(X2,B2)

Instrukcje te porównują pierwszy argument w rejestrze uniwersalnym R1 z zawartością rejestru R2 (instrukcja **CLR**) lub słowa w pamięci (instrukcja **CL**). Porównanie przebiega bit po bicie, od lewej strony do prawej. Jeśli przed porównaniem wszystkich bitów wystąpi nierówność, wówczas przerywa się dalsze sprawdzanie bitów i PSW ustawia się odpowiedni kod warunku.

PRZYKŁAD 27

POLE NAZWY:	OPERACJA	ARGUMENTY
5 9	10 15	20 25 30 35 40
COMPARE	CLR	4,5

W instrukcji tej porównuje się dane zawarte w rejestrach 4 i 5:

R4

F B	4 A	4 A	4 A

R5

F A	4 A	4 A	4 A

Porównanie wtedy bitów zostanie przerwane w ósmym kroku z powodu niezgodności bitów i instrukcja zakończy się ustawieniem w PSW kodu warunku 2, co oznacza, że pierwszy argument jest większy od drugiego. ■

Porównanie bezpośrednie:

CLI	D1(B1),I2

Instrukcja **CLI** umożliwia porównanie jednego bajtu znajdującego się w pamięci pod adresem określonym przez D1(B1) z bajtem zapisanym bezpośrednio w instrukcji jako drugi argument. Bity porównywane są kolejno od lewej strony do prawej. Rezultatem porównania jest ustawienie kodu warunku w PSW. Instrukcję tę wygodnie jest stosować dla sprawdzania stanu jednobajtowych przełączników lub wyszukiwania jednobajtowych symboli sterujących.

PRZYKŁAD 28

Zmienna o długości jednego słowa i nazwie **SWITCH** spełnia w programie rolę pewnego przełącznika. Należy sprawdzić, czy trzeci znak tego słowa jest literą **S.** Zadanie to może być wykonane za pomocą instrukcji **CLI,** w której drugi argument przedstawimy trzema różnymi sposobami:

POLE NAZWY (1–9)	OPERACJA (10–15)	ARGUMENTY (16–35)
	CLI	SWITCH+2,C'S'
	CLI	SWITCH+2,X'E2'
	CLI	SWITCH+2,B'111ØØØ1Ø'

Wynikiem każdej z trzech instrukcji będzie ustawienie kodu warunku 0, jeśli w trzecim bajcie słowa SWITCH pojawi się znak **S.** ■

Porównanie znaków:

CLC	D1(L,B1),D2(B2)

Dwa pola o jednakowej długości znajdujące się w pamięci pod adresami określonymi w instrukcji porównywane są bit po bicie, od lewej strony do prawej. Porównywanie trwa tak długo, dopóki system nie wykryje różnych bitów na tych samych pozycjach lub do końca wyznaczonych pól. W wyniku wykonania instrukcji ustala się kod warunku, podobnie jak w pozostałych instrukcjach porównywania.

PRZYKŁAD 29

W pewnym programie do pamięci głównej wczytuje się karty, których zawartość zapisywana jest każdorazowo w polu o nazwie **ZAPIS.** Kolumny

POLE NAZWY (1–9)	OPERACJA (10–15)	ARGUMENTY (16–30)
ZAPIS	DS	CL80
	CLC	ZAPIS+8(2),=C'RW'

9 i 10 każdej karty zawierają symbol dokumentu. Napiszemy instrukcję, która będzie sprawdzać, czy wczytana karta zawiera dane z dokumentu o symbolu **RW.**

W wyniku działania tej instrukcji **CLC** ustali się kod warunku równy zeru, jeżeli dokument będzie miał symbol RW. W innym wypadku kod warunku będzie miał wartość 1 lub 2.

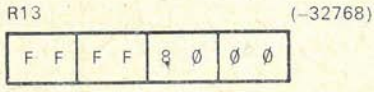
Porównanie arytmetyczne:

CR	R1, R2
C	R1, D2 (X2, B2)
CH	R1, D2 (X2, B2)

Instrukcje **CR,C,CH** porównują arytmetycznie dwie liczby stałoprzecinkowe znajdujące się (w zależności od formatu instrukcji) w rejestrach, rejestrze i słowie pamięci oraz rejestrze i i półsłowie pamięci. Wynikiem porównania jest ustawienie odpowiedniego kodu warunku, podobnie jak przy porównywaniu kodów.

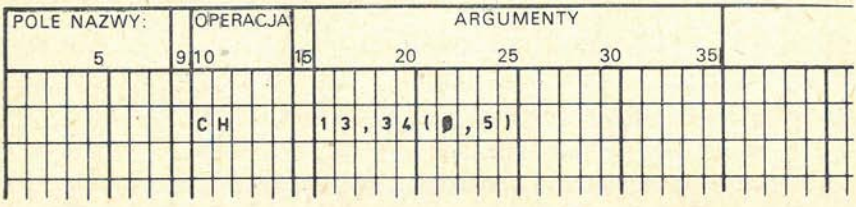
PRZYKŁAD 30

W rejestrze 13 znajduje się liczba:



Rejestr 5 zawiera liczbę 2000_{16}.

W wyniku działania instrukcji:



system porówna 32 bity rejestru 13 z 32 bitami pamięci (półsłowo z adresem $(2022)_{16}$ rozszerzone do pełnego słowa bitami znaku, w tym wypadku jedynkami) i ustawi kod warunku równy 0, ponieważ porównywane liczby są równe.

d. Instrukcje logiczne

Instrukcje logiczne w języku ASSEMBLER tworzy grupa instrukcji realizujących trzy logiczne funkcje:

— koniunkcję (iloczyn logiczny),
— alternatywę (suma logiczna),
— suma modulo 2 (różnica symetryczna).

Każdej z tych funkcji logicznych odpowiada kilka instrukcji różniących się formatem.

Instrukcje iloczynu logicznego

Instrukcje typu iloczyn logiczny (koniunkcja) realizują funkcję logiczną zgodnie z następującą tabelką:

Wartość bitów		Wynik
I argument	II argument	I argument
0	0	0
0	1	0
1	0	0
1	1	1

Funkcję tę spełniają cztery instrukcje:

Kod operacji	Symbol instrukcji	Format assemblerowy	Nieprawidłowości	Kod warunku
14	NR	R1,R2		C
54	N	R1,D2(X2,B2)	A,S,P	C
94	NI	D1(B1),I2	P,A	C
D4	NC	D1(L,B1),D2(B2)	P,A	C

Po wykonaniu instrukcji ustawia się kod warunku 0 lub 1:

0 — wynik zerowy wszystkie bity wyniku równe zeru,
1 — wynik różny od zera przynajmniej jeden z bitów jest jedynką.

Iloczyn logiczny „rejestr—rejestr":

NR	R1,R2

W instrukcji **NR** oba argumenty znajdują się w rejestrach. Wynik koniunkcji zawartości obu tych rejestrów pozostaje w rejestrze R1. Udział w operacji biorą wszystkie 32 bity rejestrów. Drugi argument nazywa się niekiedy maską.

Iloczyn logiczny „rejestr—pamięć":

N	R1,D2(X2,B2)

Pierwszy argument operacji mieści się w rejestrze, drugi w pamięci głównej. Wynikiem działania instrukcji **N** jest koniunkcja obu argumentów. Wynik zostaje wpisany do pierwszego rejestru.

PRZYKŁAD 31

W rejestrze 11 drugi bajt zawiera pewną liczbę *l*. Pozostałe trzy bajty zapełnione są dowolnymi kodami. Należy „oczyścić" rejestr ze zbędnej informacji tak, aby interesująca nas liczba *l* znajdowała się w rejestrze 11 w postaci liczby stałoprzecinkowej.

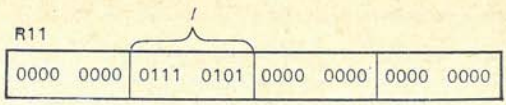
R11 (*l*)

5 8	7 5	A 1	B A
0101 1000	0111 0101	1010 0001	1011 1010

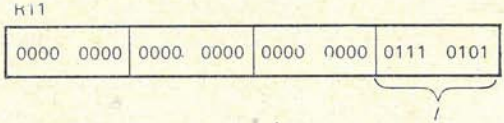
Zadanie wykonamy używając w tym celu instrukcji:

POLE NAZWY	OPERACJA	ARGUMENTY
5 9	10 15	20 25 30
	DS	ØF
MASKA	DC	X'ØØFFØØØØ'
	N	11,MASKA
	SRL	11,16

Po wykonaniu instrukcji **N** rejestr 11 będzie zawierał:



a po ostatniej instrukcji:



Iloczyn logiczny bezpośredni:

NI	D1(B1),I2

Instrukcja **NI** jest bardzo przydatna do ustawiania w programie przełączników bitowych. W instrukcji tej maską jest drugi argument — jeden bajt zadany bezpośrednio w instrukcji, natomiast pierwszy argument znajduje się w pamięci. Wynik — iloczyn logiczny — wpisany zostaje na miejscu argumentu pierwszego.

Jako przykład podamy, że instrukcja **NI ALFA,X'ØØ'** spowoduje wyzerowanie jednego bajtu pamięci o adresie **ALFA.**

Iloczyn logiczny znaków:

NC	D1(L,B1),D2(B2)

Instrukcja **NC** daje w wyniku iloczyn logiczny dwóch pól pamięci o jednakowej długości, nie większej jednak od 256 bajtów. Rezultat operacji pozostaje w polu pamięci pierwszego argumentu.

Instrukcje sumy logicznej

Instrukcje te realizują funkcję logiczną — alternatywę, określoną następującą tabelką:

Wartość bitów		Wynik
I argument	II argument	I argument
0	0	0
0	1	1
1	0	1
1	1	1

Po wykonaniu instrukcji wynik pozostanie w pierwszym argumencie, a kod warunku przyjmuje wartość zgodnie z regułą:

0 — wszystkie bity wyniku równe zeru,

1 — co najmniej jeden bit wyniku różny od zera.

Istnieją cztery rodzaje instrukcji sumy logicznej:

Kod operacji	Symbol instrukcji	Format assemblerowy	Nieprawidłowości	Kod warunku
16	OR	R1,R2		C
56	O	R1,D2(X2,B2)	P,A,S	C
96	OI	D1(B1),I2	P,A	C
D6	OC	D1(L,B1),D2(B2)	P,A	C

Suma logiczna „rejestr—rejestr":

OR	R1,R2

W instrukcji tej obydwa argumenty znajdują się w rejestrach. Suma logiczna zawartości tych rejestrów pozostaje w rejestrze pierwszym. W tym wypadku symbol instrukcji pokrywa się z angielską nazwą tej funkcji (*or* = lub).

Suma logiczna „rejestr—pamięć":

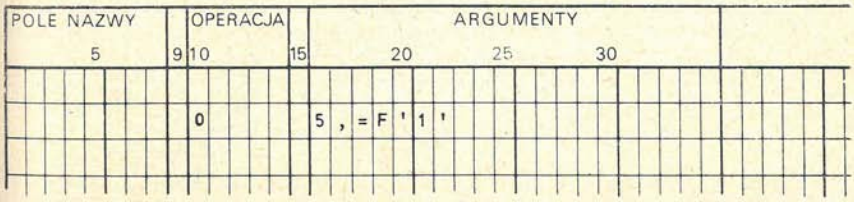
O	R1,D2(X2,B2)

PRZYKŁAD 32

Napiszemy instrukcję, która zrealizuje następujący algorytm:

1) jeśli w rejestrze 5 znajduje się liczba parzysta, wówczas należy zamienić ją na najbliższą większą liczbę nieparzystą,

2) jeżeli w rejestrze 5 jest liczba nieparzysta, wartość jej nie powinna ulec zmianie.



Instrukcje suma modulo dwa (lub inaczej różnica symetryczna lub nierównoważność) realizują natępującą funkcję logiczną:

Wartość bitów		Wynik I argument
I argument	II argument	
0	0	0
0	1	1
1	0	1
1	1	0

Funkcję tę wykonują cztery instrukcje:

Kod operacji	Symbol instrukcji	Format assemblerowy	Nieprawidłowości	Kod warunku
17	XR	R1,R2		C
57	X	R1,D2(X2,B2)	P,A,S	C
97	XI	D1(B1),I2	P,A	C
D7	XC	D1(L,B1),D2(B2)	P,A	C

W rezultacie sumowania modulo dwa pierwszego argumentu z drugim otrzymujemy wynik na miejscu pierwszego argumentu i w zależności od wartości wyniku ustawia się kod warunku:

0 — wszystkie bity wyniku równe zeru,

1 — przynajmniej jeden bit wyniku różny od zera.

Suma modulo 2 „rejestr—rejestr":

XR	R1,R2

Jako przykład rozważmy taką sytuację, w której obydwa argumenty to jeden i ten sam rejestr. W takim wypadku wynikiem działania instrukcji będzie zapisanie w rejestrze samych zer. Na przykład, instrukcja:

XR 12,12

spowoduje wyzerowanie rejestru 12 i ustawienie kodu warunku równego 0.

PRZYKŁAD 33

Początkowe litery alfabetu od A do I przedstawione są w kodzie EBCDIC w następujący sposób: A = C1, B = C2, C = C3 itd. Należy zamienić je na odpowiadające im cyfry 1, 2, 3 itd. Przypuśćmy, że tekst literowy zapisany jest w polu pamięci **TEKST.** Drugi argument — **MASKA** zostanie wprowadzona przez instrukcję **DC.**

TEKST

E	D	C	B	A
C 5	C 4	C 3	C 2	C 1
1100 0101	1100 0100	1100 0011	1100 0010	1100 0001

MASKA

3 Ø	3 Ø	3 Ø	3 Ø	3 Ø
0011 0000	0011 0000	0011 0000	0011 0000	0011 0000

Po wykonaniu instrukcji:

POLE NAZWY: 1–9	OPERACJA 10–15	ARGUMENTY: 16–
MASKA	DC	5X'3Ø'
	XC	TEKST(5),MASKA

w polu **TEKST** otrzymamy wartości:

TEKST

5	4	3	2	1
F 5	F 4	F 3	F 2	F 1
1111 0101	1111 0100	1111 0011	1111 0010	1111 0001

■

Instrukcja Testuj z maską

Do rozdziału tego włączyliśmy instrukcję Testuj z maską, chociaż nie realizuje ona żadnej funkcji logicznej. Jest to uzasadnione tym, że instrukcja ta przeznaczona jest przede wszystkim do sprawdzania przełączników bitowych w programie, co zwykle wykonuje się za pomocą operacji logicznych.

Testuj z maską:

TM	D1(B1),12

Instrukcja **TM** służy do sprawdzania bitów jednobajtowego pola pamięci — pierwszego argumentu. Drugi argument jest jednobajtową maską, zadaną w sposób bezpośredni. Działanie instrukcji polega na porównaniu bitów w bajcie danych z odpowiadającymi im bitami maski. Porównywane są tylko te pozycje, w których bit maski równy jest 1. W wyniku porównania (testowania) ustawiony zostaje kod warunku, natomiast wartość argumentów nie ulegnie zmianie. Kod warunku:

0 — wszystkie porównywane bity równe są zeru lub też zerowa maska,

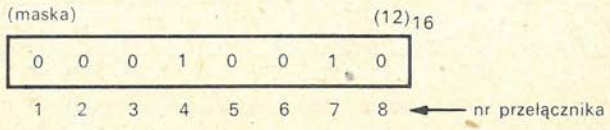
1 — wynik porównania jest mieszany, tzn. wśród analizowanych bitów są zarówno zera jak i jedynki,

3 — wszystkie testowane pozycje równe są jedynce.

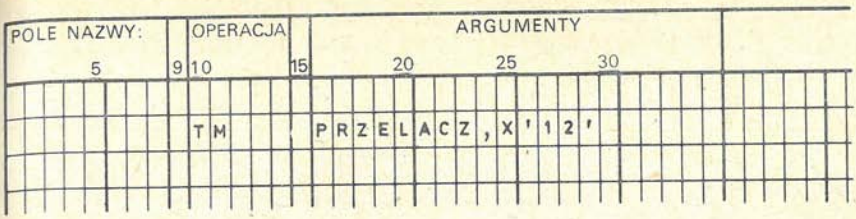
W programie instrukcja **TM** występuje najczęściej w roli przełącznika bitowego, umożliwiającego przekazywanie sterowania do odpowiednich miejsc programu w zależności od stanu bitów przełącznika pierwszego argumentu.

PRZYKŁAD 34

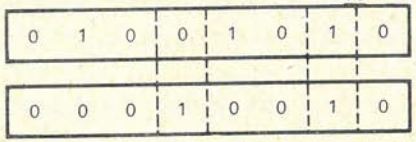
Należy sprawdzić bajt **PRZEŁĄCZ,** składający się z ośmiu bitowych przełączników. Przypuśćmy, że w danym momencie stan przełączników może być przedstawiony liczbą szesnastkową 4A. W przykładzie tym żądamy przetestowania przełączników 4 i 7, a więc maska będzie miała wartość:



Zadanie to można wykonać za pomocą instrukcji **TM**:



Ponieważ przełączniki miały wartość $4A_{16}$, a maska 12_{16}·



w rezultacie wykonania instrukcji ustali się kod warunku 1. ■

e. Instrukcje skoków

Zadaniem instrukcji skoków jest zmiana kolejności wykonywania instrukcji programu. Na ogół jest to podjęcie decyzji co do dalszej realizacji programu na podstawie sytuacji powstałej po wykonaniu poprzedniej instrukcji. Ogólnie można podzielić instrukcje skoków na dwie grupy: skoki warunkowe, wybierające dalszą drogę programu w zależności od kodu warunku w PSW lub zawartości określonego rejestru oraz skoki ze śladem (z powrotem), przekazujące sterowanie do pewnej sekwencji instrukcji i umożliwiające powrót do instrukcji, która skok spowodowała.

Kod operacji	Symbol instrukcji	Podstawowy format assemblerowy	Nieprawidłowości	Kod warunku
07	**BCR**	M1,R2		
47	**BC**	M1,D2(X2,B2)		
46	**BCT**	R1;D2(X2,B2)		
06	**BCTR**	R1,R2		
86	**BXH**	R1,R3,D2(B2)		
87	**BXLE**	R1,R3,D3(B2)		
45	**BAL**	R1,D2(X2,B2)		
05	**BALR**	R1,R2		

Jak już wspominaliśmy (por. rozdz. I) bity 34 i 35 w słowie stanu programu PSW zwane są kodem warunku i mogą przyjmować wartość 00, 01, 10 lub 11 w rezultacie wykonania pewnych instrukcji. Konkretna wartość kodu warunku zależy od rodzaju instrukcji i wartości argumentów.
Skok warunkowy **BC**:

BC	M1,D2(X2,B2)

Instrukcja **BC** powoduje skok w programie do instrukcji umieszczonej pod adresem wskazanym drugim argumentem, jeżeli kod warunku znajdujący się aktualnie w PSW odpowiada masce M1, która jest pierwszym argumentem. Maska może przyjąć cztery podstawowe wartości:

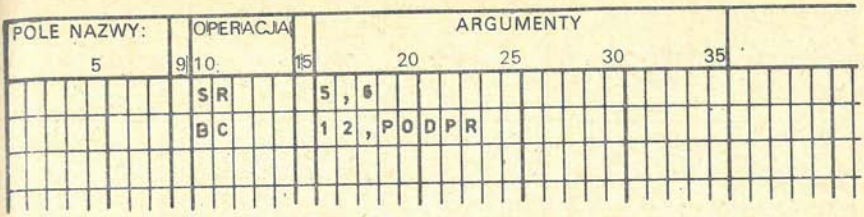
wartość maski	odpowiadający jej kod warunku w PSW
8	00 (0)
4	01 (1)
2	10 (2)
1	11 (3)

W instrukcji zapisuje się maskę w postaci liczby dziesiętnej, natomiast w rozkazie maszynowym maska jest polem czterobitowym, w którym można zapisać liczby od 0000 do 1111. Tak więc, jeśli w PSW ustawiony jest kod warunku 00, a w instrukcji skoku maska ma wartość 8, to zostanie wykonany skok do adresu drugiego argumentu; przy wszystkich innych wartościach kodu warunku program będzie kontynuowany od instrukcji następującej bezpośrednio po **BC**.

Jeżeli chcemy, aby skok nastąpił w bardziej złożonych sytuacjach, tzn. w wypadku wystąpienia jednego z kilku możliwych warunków, maska powinna być odpowiednią sumą wielkości 8-4-2-1. Na przykład, jeśliby programista żądał skoku w wypadku, gdy kod warunku równy jest 00 lub 10, to wtedy maska powinna mieć wartość $8+2 = 10$.

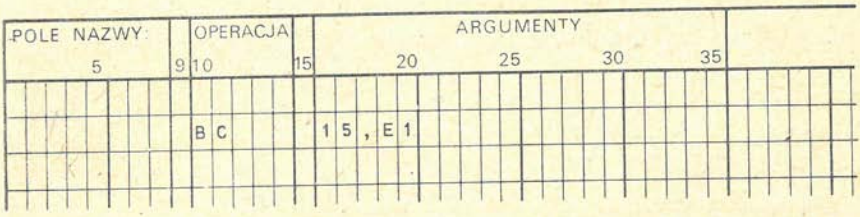
Maska $1111 \equiv 15$ odpowiada sytuacji, gdy chcemy, aby skok nastąpił przy dowolnym kodzie warunku, a więc w każdym wypadku. Jest to oczywiście skok bezwarunkowy. Inny szczególny wypadek to maska równa $0000 \equiv 0$, która odpowiada instrukcji „nic nie rób".

PRZYKŁAD 35



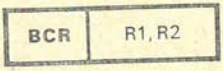
Po wykonaniu instrukcji odejmowania **SR** ustawi się kod warunku 0 lub 1, jeśli wynik jest równy zeru lub mniejszy od zera, albo 2 lub 3, jeśli wynik jest dodatni lub wystąpił nadmiar. W wyniku instrukcji **BC** z maską 12 program będzie kontynuowany od adresu **PODPR,** jeśli wynik odejmowania był równy lub mniejszy od zera; w przeciwnym razie zostanie wykonany następny rozkaz po instrukcji **BC.**

PRZYKŁAD 36



Instrukcja ta jest skokiem bezwarunkowym, ponieważ niezależnie od zaistniałych w programie warunków program przejdzie do wykonywania instrukcji z adresem symbolicznym E1.

Skok warunkowy **BCR:**



Instrukcja **BCR** jest analogiczna do instrukcji **BC** z wyjątkiem formatu; jak zwykle w formacie **RR** drugi argument jest w rejestrze R2, którego zawartość wskazuje adres skoku.

PRZYKŁAD 37

POLE NAZWY	OPERACJA	ARGUMENTY
5 9	10 15	20 25 30 35
	LA	5,PROGR1
	CLI	SYMB,C'A'
	BCR	8,5
NAST	......	

Te trzy instrukcje można opisać słowami: „jeśli w polu pamięci o nazwie **SYMB** przechowuje się znak 'A', wtedy skocz do adresu zawartego w rejestrze 5, czyli adresu **PROGR1.** W przeciwnym wypadku wykonaj instrukcję **NAST.**

Ponieważ obliczanie maski jest nieco kłopotliwe, lista rozkazów została rozszerzona o instrukcje, będące szczególnymi wypadkami instrukcji skoków warunkowych **BC** i **BCR.** Ułatwienie polega na tym, że w kodach tych instrukcji zawarta jest maska w postaci nazwy mnemotechnicznej, w związku z czym w instrukcji podaje się tylko jeden argument — adres skoku. W tablicy 17 podano rozszerzone mnemotechniczne kody instrukcji i ich równoważne instrukcje podstawowe. Tablica zawiera również angielskie nazwy instrukcji, znajomość których może ułatwić zapamiętanie kodów instrukcji.

Skok według licznika:

BCT	R1,D2(X2,B2)

Instrukcja **BCT** realizuje następujący algorytm:

1) zawartość rejestru R1 zmniejsza się o 1,

2) sprawdza się, czy znajdująca się w rejestrze liczba jest równa zeru:

3) jeśli liczba w rejestrze jest różna od zera, następuje skok do podanego w instrukcji adresu,

4) jeśli pierwszy argument równy jest zeru, skoku nie ma i wykonywana będzie najbliższa kolejna instrukcja.

Pierwszy więc argument pełni rolę licznika, zmniejszającego w każdym przejściu swoją wartość o jeden.

Instrukcja ta jest stosowana do organizacji cykli i przeglądania tablic.

TABLICA 17

Rozszerzone mnemotechniczne kody instrukcji skoków

Instrukcja z kodem rozszerzonym	Nazwa instrukcji	Równoważna instrukcja podstawowa
BR R2	Skok bezwarunkowy (RR) *Branch Unconditional*	BCR 15,R2
B s[a]	Skok bezwarunkowy (RX) *Branch Unconditional*	BC 15,s
NOPR R2	Nic nie rób (RR) *No Operation*	BCR ∅,R2
NOP s	Nic nie rób (RX) *No Operation*	BC ∅,s
	Po instrukcjach porównania	
BH s	Skocz przy większym *Branch on High*	BC 2,s
BL s	Skocz przy mniejszym *Branch on Low*	BC 4,s
BE s	Skocz przy równym *Branch on Equal*	BC 8,s
BNH s	Skocz przy nie większym *Branch on Not High*	BC 13,s
BNL s	Skocz przy nie mniejszym *Branch on Not Low*	BC 11,s
BNE s	Skocz przy nie równym *Branch on Not Equal*	BC 7,s
	Po instrukcjach arytmetycznych	
BO s	Skocz przy nadmiarze *Branch on Overflow*	BC 1,s
BP s	Skocz przy plusie *Branch on Plus*	BC 2,s
BM s	Skocz przy minusie *Branch on Minus*	BC 4,s
BZ s	Skocz przy zerze *Branch on Zeros*	BC 8,s
BNO s	Skocz, jeśli nie ma przepełnienia *Branch on Not Overflow*	BC 14,s
BNP s	Skocz, jeśli nie plus *Branch on Not Plus*	BC 13,s
BNM s	Skocz, jeśli nie minus *Branch on Not Minus*	BC 11,s
BNZ s	Skocz, jeśli nie zero Branch, jeśli nie zero	BC 7,s

TABLICA 17 (cd.)

Instrukcja z kodem rozszerzonym	Nazwa instrukcli	Równoważna instrukcja podstawowa
	Po instrukcji *Testuj z maską* (TM)	
BO s	Skocz przy jedynkach *Branch on Ones*	BC 1,s
BM s	Skocz przy mieszanych *Branch on Mixed*	BC 4,s
BZ s	Skocz przy zerach *Branch on Zeros*	BC 8,s
BNO s	Skocz, jeśli nie jedynki *Branch on Not Ones*	BC 14,s
BNM s	Skocz, jeśli nie mieszane *Branch on Not Mixed*	BC 11,s
BNZ s	Skocz, jeśli nie zera *Branch on Not Zeros*	BC 7,s

* s — adres typu D2 (X2,B2) lub adres symboliczny.

PRZYKŁAD 38

Sprawdzić, czy 300 bajtowe pole pamięci o nazwie **TAB** zapełnione jest spacjami (kod spacji X'4Ø'). W wypadku wykrycia innego znaku niż spacja skoczyć do instrukcji z etykietą **ZNAK.** Jeśli pole **TAB** składa się z samych spacji, kontynuować program w zwykłej kolejności.

Zadanie to możemy wykonać za pomocą instrukcji:

```
POLE NAZWY      OPERACJA        ARGUMENTY
     5        9 10      15      20        25        30        35
              LA        4,TAB
              LA        5,300
POR           CLI       0(4),X'40'
              BNE       ZNAK
              AH        4,=H'1'
              BCT       5,POR
              .         .
ZNAK          .         .
```

Instrukcja **CLI** będzie potrzebna tak długo, dopóki rejestr 5 nie przyjmie wartości zerowej (300 przejść). Cykl ten może być przerwany tylko wtedy, gdy badany bajt nie będzie spacją i wówczas instrukcja **BNE** spowoduje skok do miejsca programu oznaczonego etykietą **ZNAK**.

Skok według licznika:

BCTR	R1,R2

Działanie instrukcji **BCTR** nie odbiega od opisanej instrukcji **BCT**. Jak zawsze w formacie RR drugi argument znajduje się w rejestrze, a jego zawartość wskazuje w tym wypadku adres, do którego ma nastąpić skok, gdy licznik zostanie wyzerowany.

Skok gdy indeks większy:

BXH	R1,R3,D2(B2)

Skok gdy indeks mniejszy lub równy:

BXLE	R1,R3,D2(B2)

Instrukcje te używają trzech rejestrów. Ze względów praktycznych wybiera się trzy kolejne rejestry: R1 (nieparzysty), R3 (parzysty) i następny za R3, którego nie wymienia się w instrukcji. Instrukcje te działają następująco:

1) liczbę zapisaną w R1 dodaje się algebraicznie do liczby w R3, wynik umieszczany jest w R1,

2) porównuje się rejestry R1 i rejestr nie ukazany w instrukcji, następny za R3,

3) jeśli zawartość rejestru R1 jest większa (instrukcja **BXH**) albo mniejsza lub równa (instrukcja **BXLE**) wtedy następuje skok do adresu ukazanego w instrukcji,

4) w przeciwnym razie skoku nie będzie i program przechodzi do wykonania kolejnej instrukcji.

Skok ze śladem:

BAL	R1,D2(X2,B2)

Instrukcja **BAL** umożliwia odwołanie się do pewnych sekwencji instrukcji lub podprogramu, a następnie powrót i kontynuację programu. Instrukcja

ta powoduje zapamiętanie w rejestrze pierwszego argumentu bitów 40-63 PSW, zawierających adres następnej do wykonania instrukcji (mówimy, że w R1 pozostaje „ślad"), po czym następuje skok i program jest wykonywany od miejsca wskazanego w instrukcji. Teraz programista może wykorzystać „ślad" i wrócić do miejsca, z którego nastąpił skok. W tym celu wystarczy użyć instrukcji skoku bezwarunkowego, np. **BR** R1, gdzie R1 jest tym samym rejestrem, który został użyty w instrukcji **BAL.**

PRZYKŁAD 39

W pewnym programie programista używa wielokrotnie tej samej sekwencji instrukcji, którą nazwiemy procedurą. Rozpoczyna się ona od etykiety **PROC.** Używając instrukcji **BAL** można każdorazowo w miarę potrzeby odwoływać się do tej procedury. Powrót do punktu wyjścia zapewnia instrukcja **BR.**

(*instrukcja* $n-1$)
BAL 9, PROC
(*instrukcja* $n+1$)

.
.
.

(*instrukcja* $m-1$)
BAL 9, PROC
(*instrukcja* $m+1$)

.
.
.

PROC (*instrukcja r*)

.
.
.

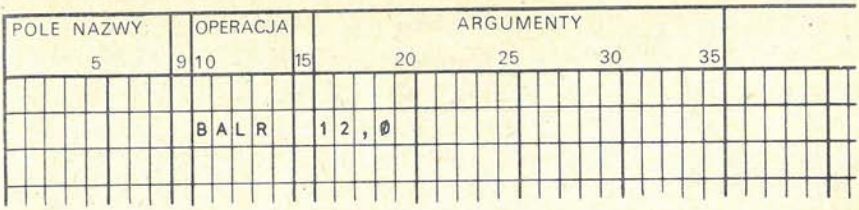
(*instrukcja s*)
BR 9 ■

Skok ze śladem:

BALR	R1,R2

Instrukcja **BALR** jest odpowiednikiem instrukcji **BAL** w formie RR. Różni się więc od instrukcji **BAL** jedynie tym, że drugi argument znajduje się w rejestrze R2 i określa adres skoku.

PRZYKŁAD 40



Instrukcja **BALR** w tej postaci rozpoczyna zwykle program w języku ASSEMBLER. W wypadku jeśli drugi argument jest zerem, nie następuje skok, a jedynym skutkiem instrukcji jest załadowanie do rejestru (w naszym wypadku 12) adresu kolejnej po **BALR** instrukcji. ■

f. Instrukcje konwersji

Dane numeryczne wprowadza się do pamięci w postaci liczb dziesiętnych rozpakowanych. Za pomocą instrukcji **PACK** można je zamienić na liczby dziesiętne spakowane, po czym dokonać konwersji dziesiętno — dwójkowej korzystając z instrukcji **CVB**. Otrzymamy w ten sposób liczby dwójkowe, które mogą być argumentami operacji arytmetycznych stałoprzecinkowych.

Konwersja niezbędna jest również w wypadku wyprowadzania liczb z pamięci. Najpierw należy za pomocą instrukcji **CVD** zamienić liczby dwójkowe, stałoprzecinkowe na liczby dziesiętne spakowane, które można rozpakować używając instrukcji **UNPK**. Liczby w takiej postaci można bez trudu wprowadzić na zewnętrzny nośnik informacji, np. wydrukować.

Formaty liczb przedstawiliśmy w rozdziale I (por. rys. 6), a instrukcje konwersji zawiera następująca tabliczka.

Kod operacji	Symbol instrukcji	Format assemblerowy	Nieprawidłowości	Kod warunku
F2	**PACK**	D1(L1,B1),D2(L2,B2)	P,A	
F3	**UNPK**	D1(L1,B1),D2(L2,B2)	P,A	
4F	**CVB**	R1,D2(X2,B2)	P,A,S,D,IK	
4E	**CVD**	R1,D2(X2,B2)	P,A,S	

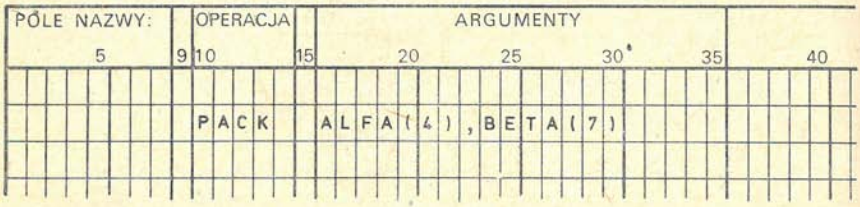
Pakowanie:

PACK	D1(L1,B1),D2(L2,B2)

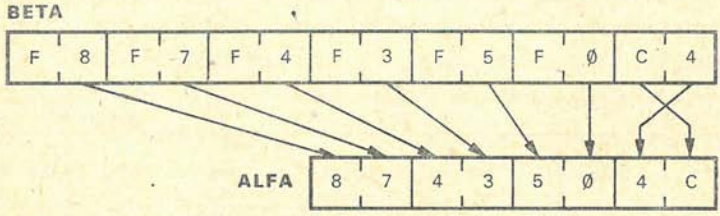
Instrukcja **PACK** przekształca dane drugiego argumentu z postaci strefowej (rozpakowanej) w format spakowany, umieszczając wynik w pamięci według adresu określonego dla pierwszego argumentu. Maksymalna długość każdego z argumentów nie może być większa od 16 bajtów. Pierwszy argument powinien być oczywiście krótszy i mieć długość równą połowie długości drugiego argumentu plus 1 (z zaokrągleniem do najbliższej większej liczby całkowitej). Jeśli pole pierwszego argumentu jest dłuższe niż jest to konieczne dla umieszczenia wyniku, zostaje ono uzupełnione z lewej strony zerami; jeśli wynik nie mieści się w zarezerwowanym dla niego polu, lewe cyfry zostają odrzucone, a prawe, najmniej znaczące, pozostają.

Ponieważ w formacie spakowanym znak znajduje się w prawej połowie najmniej znaczącego bajtu, a w formacie strefowym w lewej połowie, znak liczby jest odpowiednio przenoszony. Pokażemy to na przykładzie.

PRZYKŁAD 41



Działanie tej instrukcji ilustruje schemat:



W wyniku działania tej instrukcji **PACK** powstaje czterobajtowe pole **ALFA**, zawierające dodatnią spakowaną liczbę dziesiętną. ■

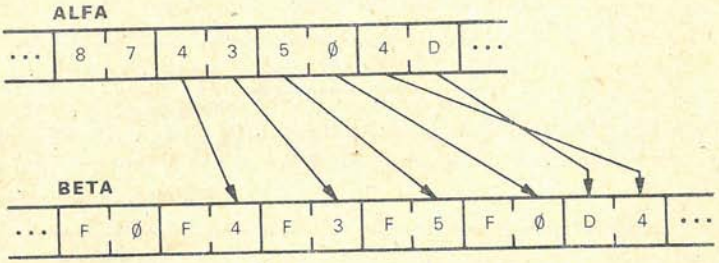
Rozpakowywanie:

UNPK	D1(L1,B1),D2(L2,B2)

Dane w postaci dziesiętnej spakowanej można przekształcić w postać rozpakowaną za pomocą instrukcji **UNPK.** Drugi argument jest liczbą

spakowaną, natomiast wynik zostaje zapisany w polu pamięci określonym jako pierwszy argument. Jeśli długość pierwszego argumentu jest większa niż to wynika z długości liczby spakowanej, wtedy pole uzupełniane jest z lewej strony zerami. Natomiast jeśli wynik nie mieści się w polu pamięci dla niego przeznaczonym, najbardziej znaczące cyfry są odrzucane. Maksymalna długość argumentów nie może przekraczać 16 bajtów.

PRZYKŁAD 42



W wyniku działania tej instrukcji **UNPK** powstaje liczba dziesiętna w kodzie EBCDIC. ■

Konwersja dwójkowa:

CVB	R1,D2(X2,B2)

Za pomocą instrukcji CVB można dokonać konwersji spakowanej liczby dziesiętnej, znajdującej się w polu pamięci określonym jako drugi argument, na liczbę dwójkową, zapisując ją do rejestru R1. Znak liczby dziesiętnej zostaje zachowany w postaci znakowego bitu w liczbie dwójkowej. Liczba dziesiętna mieści się w ośmiobajtowym polu pamięci umieszczonym na granicy podwójnego słowa. Maksymalna liczba, jaka może być przekształcona w liczbę binarną równa jest 2147483647.

PRZYKŁAD 43

Liczbę w postaci strefowej, zajmującą w pamięci jedno słowo o nazwie **DANA**, przekształcimy z postaci dziesiętnej na dwójkową. Przedtem jednak umieścimy ją na granicy podwójnego słowa w polu **DANA1**.

POLE NAZWY: 5 9	OPERACJA 10 15	ARGUMENTY 20 25 30 35 40
DANA1	DS	D
	PACK	DANA1(8),DANA(4)
	CVB	3,DANA1

W rezultacie działań tych instrukcji w rejestrze 3 zostanie umieszczona liczba **DANA** przekształcona w postać binarną. ■

Konwersja dziesiętna:

CVD	R1,D2(X2,B2)

Instrukcja **CVD** przekształca liczbę dwójkową znajdującą się w rejestrze R1 w liczbę dziesiętną spakowaną, umieszczając ją w pamięci pod adresem wskazującym drugi argument. Wynik jest ośmiobajtowym polem na granicy podwójnego słowa.

PRZYKŁAD 44

Instrukcja:

POLE NAZWY: 5 9	OPERACJA 10 15	ARGUMENTY 20 25 30 35
	CVD	11,LSPAK

spowoduje zmianę liczby zawartej w rejestrze 11:

R11

Ø Ø	Ø Ø	6 8	1 F

w liczbę dziesiętną spakowaną:

LSPAK

...	Ø Ø	Ø Ø	Ø Ø	Ø Ø	Ø Ø	2 6	6 5	5 C	...

Należy przy tym pamiętać, aby dla wyniku przeznaczone było pole umieszczone w pamięci na granicy podwójnego słowa, co można osiągnąć np. za pomocą instrukcji:

LSPAK DS D ■

g. Instrukcje arytmetyki dziesiętnej

Dotychczas rozpatrywaliśmy instrukcje arytmetyki stałoprzecinkowej. Dużym udogodnieniem dla programisty są instrukcje arytmetyki dziesiętnej, w których dane przedstawione są jako liczby dziesiętne spakowane ze znakiem. Jakimi instrukcjami posługiwać się? — wybór ten należy do programisty. Chcąc odpowiedzieć na to pytanie, należy wziąć przede wszystkim pod uwagę relację między operacjami wejścia/wyjścia a operacjami obliczeniowymi, jaka występuje w danym programie. Instrukcje arytmetyki dziesiętnej wykonują się w maszynie znacznie dłużej niż instrukcje stałoprzecinkowe, lecz jeśli dane wprowadza się w postaci znakowej i po niewielkiej ilości obliczeń drukuje, należy wówczas raczej stosować arytmetykę dziesiętną.

Instrukcje arytmetyczne

Kod operacji	Symbol instrukcji	Format assemblerowy	Nieprawidłowości	Kod warunku
F8	ZAP	D1(L1,B1),D2(L2,B2)	P,A,D,DF	C
FA	AP	D1(L1,B1),D2(L2,B2)	P,A,D,DF	C
FB	SP	D1(L1,B1),D2(L2,B2)	P,A,D,DF	C
FC	MP	D1(L1,B1),D2(L2,B2)	P,A,D,S	
FD	DP	D1(L1,B1),D2(L2,B2)	P,A,D,S,DK	
F9	CP	D1(L1,B1),D2(L2,B2)	P,A,D	C

Zerowanie z dodawaniem:

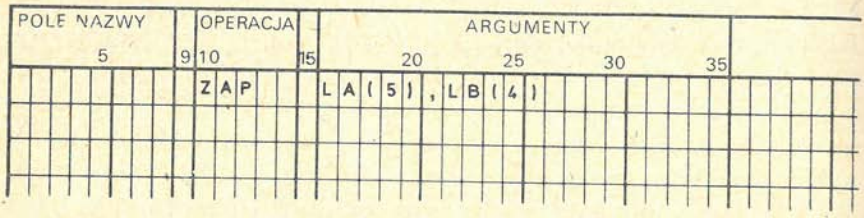
ZAP	D1(L1,B1),D2(L2,B2)

Za pomocą instrukcji **ZAP** można wyzerować dowolne pole pamięci o długości do 16 bajtów, a następnie zapisać w nim liczbę dziesiętną spako-

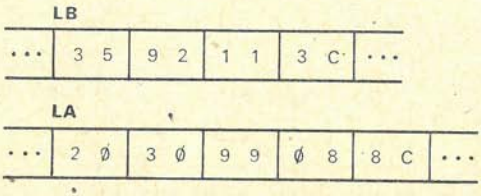
waną. Liczba ta jest drugim argumentem. Zostanie ona przesłana do pola pamięci, zawierającego pierwszy argument, czyli jest to faktycznie dodawanie do zera. Jeśli przesłana liczba nie mieści się w polu przeznaczonym dla pierwszego argumentu, następuje nadmiar dziesiętny. Natomiast gdy krótsze pole przesyłane jest do dłuższego, wówczas to dłuższe uzupełniane jest z lewej strony zerami.

Instrukcja **ZAP** powoduje ustawienie następujących kodów warunku:
0 — pierwszy argument równy jest zeru,
1 — pierwszy argument jest ujemny,
2 — pierwszy argument jest dodatni,
3 — wystąpił nadmiar.

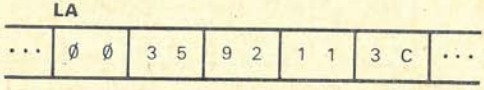
PRZYKŁAD 45



Jeśli w pamięci znajdowały się następujące liczby:

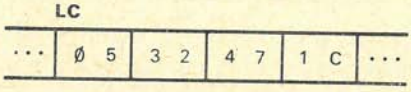


to po wykonaniu tej instrukcji w polu **LA** znajdzie się liczba:

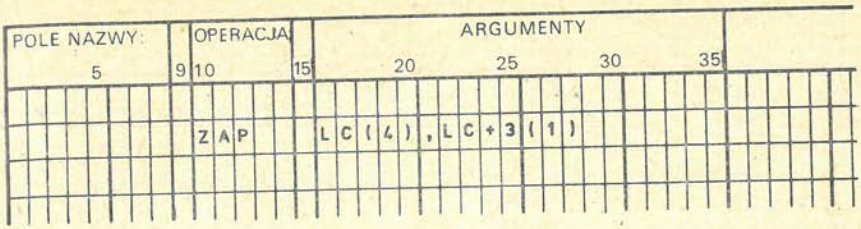


PRZYKŁAD 46

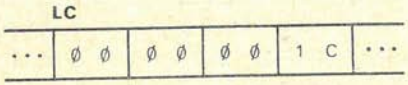
Pole **LC** zawiera liczbę:



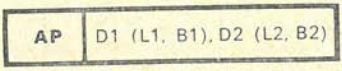
Po wykonaniu instrukcji:



otrzymamy nową zawartość pola **LC**:



Dodawanie dziesiętne:



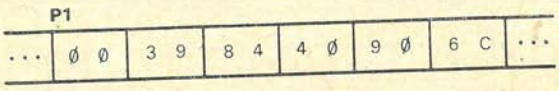
Instrukcja **AP** służy do dodawania algebraicznego dwóch liczb dziesiętnych spakowanych. Suma, również w postaci liczby dziesiętnej spakowanej, pozostaje w polu pamięci przeznaczonym dla pierwszego argumentu. Jeśli suma jest liczbą nie mieszczącą się w polu pierwszego argumentu, następuje nadmiar. Suma równa zeru ma zawsze znak dodatni. Przypominamy, że prawidłowy znak dodatni liczby dziesiętnej spakowanej znajduje się w prawym skrajnym bajcie i może mieć wartość A, C, E lub F, natomiast liczba ujemna — B lub D.

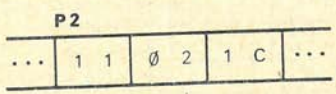
Po wykonaniu instrukcji w PSW ustawi się kod warunku:

0 — suma równa zeru,
1 — suma mniejsza od zera,
2 — suma dodatnia,
3 — nadmiar.

PRZYKŁAD 47

W polach P1 i P2 mieszczą się następujące liczby:





Po wykonaniu instrukcji:

POLE NAZWY:	OPERACJA	ARGUMENTY
5	9 10 15	20 25 30 35
	AP	P1(6),P2+1(2)

pole P1 będzie zawierało liczbę:

P1										
...	ϕ ϕ	3 9	8 4	4 ϕ	9 2	7 C	...			

Zawartość pola P2 nie ulegnie zmianie. W PSW ustali się kod warunku równy 2.

Odejmowanie dziesiętne:

SP	D1(L1,B1),D2(L2,B2)

W wyniku działania instrukcji odejmowania dziesiętnego jest różnica dwóch liczb dziesiętnych spakowanych znajdujących się w pamięci głównej. Odjemna jest pierwszym argumentem, odjemnik drugim. Otrzymana różnica pozostaje w polu pamięci pierwszego argumentu. Ustala się przy tym kod warunku, w taki sam sposób jak przy dodawaniu dziesiętnym.

PRZYKŁAD 48

ALFA									
...	ϕ ϕ	4 2	8 3	7 5	1 C	...			

OMEGA									
...	ϕ ϕ	5 3	7 1	4 4	2 C	...			

Po wykonaniu instrukcji:

POLE NAZWY:	OPERACJA	ARGUMENTY
5	9 10 15	20 25 30 35
	SP	ALFA(5),OMEGA(5)

w polu **ALFA** pozostanie ujemna liczba:

ALFA
••• Ø Ø 1 Ø 8 7 6 9 1 D •••

Mnożenie dziesiętne:

MP	D1(L1,B1),D2(L2,B2)

Za pomocą instrukcji **MP** można pomnożyć dwie liczby dziesiętne spakowane. Mnożnik występuje w instrukcji jako drugi argument, natomiast mnożna jest pierwszym argumentem. Wynik zostaje zapamiętany w pamięci głównej w polu pierwszego argumentu. Mnożnik nie może być dłuższy od 8 bajtów, jak również równy lub większy od długości pola pierwszego argumentu. Długość pola przeznaczonego dla iloczynu można obliczyć jako sumę długości pól mnożnika i mnożnej, co gwarantuje, że nadmiar nie wystąpi.

Mnożenie dziesiętne nie powoduje zmiany kodu warunku.

PRZYKŁAD 49

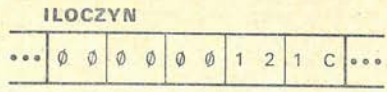
POLE NAZWY:	OPERACJA	ARGUMENTY
5 9	10 15	20 25 30 35
MNOZNIK	DC	PL3'3443'
MNOZNA	DC	PL2'1211'
ILOCZYN	DS	CL5
	ZAP	ILOCZYN(5),MNOZNA(2)
	MP	ILOCZYN(5),MNOZNIK(3)

Przed wykonaniem instrukcji **ZAP** i **MP** biorące udział w instrukcjach pola miały wartość:

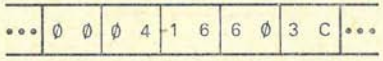
MNOŻNIK
••• Ø 3 4 4 3 C •••

MNOŻNA
••• 1 2 1 C •••

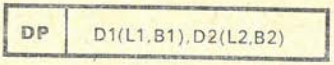
Instrukcja **ZAP** spowodowała odpowiednie umieszczenie mnożnej w polu ILOCZYN:



Po wykonaniu instrukcji mnożenia pole **ILOCZYN** zawiera liczbę:



Dzielenie dziesiętne:



Instrukcja **DP** realizuje dzielenie dwóch liczb dziesiętnych spakowanych. Zarówno dzielna, która jest pierwszym argumentem, jak i dzielnik, który jest drugim argumentem, znajdują się w pamięci głównej. Wynik dzielenia pozostaje na miejscu pierwszego argumentu, przy czym:

— całkowita część ilorazu (o długości obliczonej jako różnica między długością pierwszego i drugiego argumentu) mieści się w najbardziej znaczących bajtach pierwszego argumentu,

— reszta zajmuje mniej znaczące bajty pola pierwszego argumentu (długość reszty równa się długości dzielnika).

Obie części wyniku dzielenia są liczbami dziesiętnymi spakowanymi ze znakiem. Długość pola przeznaczonego na wynik powinna być równa sumie długości dzielnej i dzielnika, natomiast długość dzielnika nie może przewyższać 8 bajtów i nie może być większa lub równa długości pierwszego argumentu.

Kod warunku nie ulega zmianie.

PRZYKŁAD 50

Przypuśćmy, że pola **DZIELNA, DZIELNIK** i **ILORAZ** zawierają następujące liczby:

ILORAZ

...	0 0	0 0	0 0	0 0	0 0	0 C	...

Instrukcja **ZAP** spowoduje wpisanie dzielnej do pola **ILORAZ**, a instrukcja **DP** — dzielenie:

POLE NAZWY	OPERACJA	ARGUMENTY
5	9 10 15	20 25 30 35
	ZAP	ILORAZ(6),DZIELNA(4)
	DP	ILORAZ(6),DZIELNIK(2)

W efekcie otrzymamy wynik w polu **ILORAZ**:

ILORAZ

...	0 0	1 0	8 8	7 C	1 1	0 C	...

część całkowita ilorazu — reszta

Porównanie dziesiętne:

CP	D1(L1,B1),D2(L2,B2)

Porównuje się dwie liczby dziesiętne spakowane. Jeśli liczby te mają różne długości, to krótszą uzupełnia się zerami z lewej strony. Przyjmuje się, że zero dodatnie (+0) równe jest zeru ujemnemu (−0). Wartość obu argumentów nie ulega zmianie, a jedynym wynikiem instrukcji jest ustawienie kodu warunku:

0 — obie liczby są sobie równe,
1 — pierwszy argument jest mniejszy od drugiego,
2 — pierwszy argument jest większy od drugiego.

PRZYKŁAD 51

Dwa pola pamięci: **LA** i **LB** mają następujące wartości:

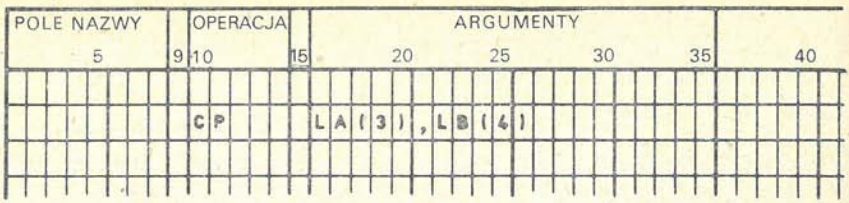
LA

...	4 1	3 5	2 C	...

LB

...	0 0	4 1	3 5	2 F	...

Po wykonaniu instrukcji:



otrzymamy kod warunku równy 0, ponieważ obie liczby są równe (znaki C i F są równoważne w liczbach dziesiętnych spakowanych i oznaczają plus, natomiast liczba w polu **LA** zostanie uzupełniona przed porównaniem jednym zerowym bajtem).

4. Makroinstrukcje

W języku ASSEMBLER można definiować makroinstrukcje, które generują w procesie translacji więcej niż jedną instrukcję maszynową. Makroinstrukcje[3] znacznie rozszerzają możliwości programowania.

Makrodefinicja

Każda makroinstrukcja użyta w programie musi być w tym programie opisana. Opis makroinstrukcji, zwany makrodefinicją, składa się z instrukcji określających sposób przekształcania makroinstrukcji w ciąg rozkazów maszynowych. Każda makrodefinicja musi składać się z następujących instrukcji:

— instrukcja **MACRO**, wskazująca początek makrodefinicji.

— instrukcja prototypu,

— treść makrodefinicji, składająca się z dowolnych instrukcji języka ASSEMBLER oprócz: **END, ICTL, ISEQ, PRINT, MACRO, MEND,**

— instrukcja **MEND**, wskazująca koniec makrodefinicji.

Instrukcje **MACRO** i **MEND** nie mają argumentów i nazw. Pierwszą po instrukcji **MACRO** powinna być instrukcja prototypu. Określa się w niej nazwę makroinstrukcji i parametry formalne. Instrukcja prototypu ma następującą postać:

---

[3] Por. *Programmirowanija na jazykie ASSEMBLERA JS EWM*, Moskwa 1975; *DOS/JS ASSEMBLER* — opis języka.

Nazwa	Operacja	Argumenty
Parametr lub spacje	Symbol (kod operacji makroinstrukcji)	Parametry rozdzielone przecinkami

Parametry formalne, które występują jako argumenty lub jako nazwy instrukcji, mogą być używane w dowolnym innym miejscu makrodefinicji. Podczas przetwarzania programu translator zastępuje parametry formalne parametrami rzeczywistymi, podanymi w makroinstrukcji. Parametry zamieniane są zgodnie ze swoją pozycją w spisie parametrów instrukcji prototypu i w spisie parametrów rzeczywistych w makroinstrukcji. Parametry takie nazywają się parametrami pozycyjnymi. Zapisuje się je jako symbole poprzedzone znakiem &, np., &A, &POLE, &X32.

Parametry formalne, które zamieniane są na wartości rzeczywiste niezależnie od pozycji ich występowania, noszą nazwę parametrów kluczowych. Parametrom tym można nadawać wartości początkowe w samej makrodefinicji. Zapisuje się je podobnie jak parametry pozycyjne, tzn. *&symbol*, a dodatkowo stawia się znak równości i pewną wartość. Wartość tę zapisuje się według takich samych reguł jak i argumenty makroinstrukcji. Przykłady argumentów kluczowych: **&C = X'2'**, **&P3 =**, **&ET = C'SKOK'**.

Parametry można łączyć z innymi parametrami lub znakami. Jeśli znaki łączy się z parametrami, wówczas parametr zapisuje się bezpośrednio za tymi znakami, np. **L&SUMA**. Natomiast w wypadku, gdy chcemy dopisać inne znaki do parametru, należy parametr oddzielić od nich kropką, np. **&MV.IX.**

Argumenty makroinstrukcji

Makroinstrukcja ma budowę analogiczną do instrukcji prototypu. W polu argumentów wymienia się rzeczywiste parametry pozycyjne lub kluczowe. Można pominąć dowolne parametry pozycyjne, ale wówczas w makroinstrukcji należy pozostawić wszystkie przecinki tak, jak gdyby nie brakowało żadnego parametru, np.:

```
MOVE  &P1, &P2, &P3, &P4   (prototyp)
MOVE  POLE1, 320, 'STOP'   (makroinstrukcja)
```

Jeśli chodzi o parametry kluczowe, to wymienia się w makroinstrukcji tylko te, które mają zmieniać standardową wartość podaną za znakiem równości w odpowiednich parametrach kluczowych w makrodefinicji.

Argumentem makroinstrukcji (czyli parametrem rzeczywistym) może być kombinacja dowolnych znaków kodu EBCDIC. Istnieją pewne ograniczenia dotyczące używania apostrofów, znaku & i nawiasów. Argument może zaczynać się i kończyć apostrofem. Obydwa apostrofy wchodzą w tym wypadku do wartości argumentu. Wewnątrz argumentu mogą również występować apostrofy — każda para sąsiednich apostrofów reprezentuje tylko jeden.

W argumentach makroinstrukcji mogą występować nawiasy okrągłe, lecz tylko w parach: każdemu nawiasowi otwierającemu musi towarzyszyć nawias zamykający.

Jeśli chcemy w makroinstrukcji użyć znaku&, należy go zapisać dwukrotnie.

Każda makroinstrukcja powoduje wygenerowanie ciągu instrukcji maszynowych, który nazwiemy rozwinięciem makroinstrukcji. Rozwinięcie makroinstrukcji otrzymuje się przez zastąpienie parametrów formalnych w makrodefinicji parametrami rzeczywistymi podanymi w makroinstrukcji.

PRZYKŁAD 52

W przykładzie tym określimy pewną makroinstrukcję ZAMIEN i podamy jej rozwinięcie.

Nazwa	Operacja	Argumenty
* MAKRODEFINICJA	MACRO	
	ZAMIEN	&P1,&P2,&P3,&PK1 = 1∅,
		&PK2 = 6
	ST&P3	&PK1,&P1
	ST&P3	&PK2,&P2
	L	&PK1,&P2
	L	&PK2,&P1
	MEND	
* MAKROINSTRUKCJA		
	ZAMIEN	POLE,FIELD,PK1 = 5
* ROZWINIĘCIE		
	ST	5,POLE
	ST	6,FIELD
	L	5,FIELD
	L	6,POLE

Parametry systemowe

Oprócz parametrów pozycyjnych i kluczowych, nazywanych ogólnie parametrami stałymi, programista może używać parametrów systemowych o następujących nazwach: **&SYSNDX, &SYSECT** i **&SYSLIST.**

1. Parametr systemowy **&SYSNDX** jest czterocyfrowym numerem porządkowym danej makroinstrukcji. Gdy translator spotyka w programie pierwszą makroinstrukcję z parametrem **&SYSNDX** nadaje mu wartość 0001. Przetwarzanie każdej następnej makroinstrukcji (również tej samej w pętli) powoduje zwiększenie tej wartości o jeden.

2. Parametr systemowy **&SYSECT** równy jest nazwie sekcji programowej zawierającej daną makroinstrukcję.

3. Parametr systemowy **&SYSLIST** pozwala na odwoływanie się do stałego parametru bez użycia jego nazwy. W tym celu parametr **SYSLIST** zapisuje się z indeksem, który wskazuje pozycję parametru rzeczywistego w makroinstrukcji. Przypuśćmy że (dla przykładu) podana jest następująca makroinstrukcja:

MAKE FORM,(AR,A,AL,ALH),,ST

W tym wypadku wartością parametru systemowego **&SYSLIST(4)** jest **ST,** natomiast **&SYSLIST(2,3)** oznacza parametr **AL,** a **&SYSLIST(3)** oznacza trzeci parametr, który jednak został tu opuszczony.

Parametry zmienne

Podczas przetwarzania przez translator danej makroinstrukcji wartość parametrów stałych i systemowych nie ulegnie zmianie. W niektórych wypadkach konieczne jest użycie parametrów zmieniających swoje wartości w toku generacji postaci wynikowej makroinstrukcji. Rolę taką spełniają parametry zmienne. Wszystkie parametry zmienne należy zadeklarować na początku makrodefinicji za pomocą instrukcji **LCLA, LCLC, LCLB.** Pierwsza litera **(L)** oznacza, że chodzi tu o parametry lokalne, tzn. takie, których używa się tylko w obrębie danej makrodefinicji. Istnieje możliwość używania w języku ASSEMBLER parametrów globalnych, które zakresem działania obejmują cały program (odpowiednie instrukcje mają wówczas symbole **GCLA, GCLC, GCLB).**

Instrukcje deklaracji zmiennych parametrów określają:

LCLA — parametry arytmetyczne,
LCLC — parametry znakowe,
LCLB — parametry binarne.

Instrukcje te ustalają nie tylko typ parametru, lecz również jego wartość początkową. Początkową wartością parametru arytmetycznego jest zero, parametru znakowego — puste (brak znaków), binarnego — logiczna wartość zero.

Wartości parametrów zadeklarowanych tymi instrukcjami można zmieniać za pomocą instrukcji **SETA, SETC** lub **SETB.** Instrukcje te mają następującą postać:

parametr-arytmetyczny	**SETA**	*wyrażenie-arytmetyczne*
parametr-znakowy	**SETC**	*wyrażenie-znakowe*
parametr-logiczny	**SETB**	(*wyrażenie-logiczne*)

Działanie wymienionych instrukcji polega na nadaniu parametrom wskazanym z lewej strony kodu instrukcji wartości obliczonej z odpowiedniego wyrażenia.

Wyrażenie arytmetyczne składa się ze składników połączonych znakami: + (dodawanie,) − (odejmowanie), * (mnożenie), / (dzielenie). Składnikami tymi mogą być człony samodefiniujące, parametry lub charakterystyki. Charakterystyką jest liczba całkowita określająca pewną cechę symbolu lub argumentu makroinstrukcji. Na przykład charakterystyka długości jest liczbą równą ilości bajtów pola pamięci, w którym przechowywana jest zmienna o podanej nazwie. Charakterystykę długości zapisuje się jako literę **L** z apostrofem, np. **L'POLE1** określa długość (w bajtach) pola pamięci o nazwie **POLE1.** Jeżeli pole to byłoby poprzednio zdefiniowane za pomocą następującej instrukcji:

```
POLE1 DC F'35'
```

wówczas **L'POLE1** = 4.

Wyrażenie znakowe może składać się z jednego lub kilku składników połączonych kropką. Składnikiem może być w tym wypadku dowolny ciąg znaków ujęty w apostrofy lub charakterystyka typu. Charakterystyka typu, oznaczana przez **T'**, przyjmuje wartość równą jednej literze w zależności od typu parametru lub symbolu, w stosunku do których została użyta. Na przykład jeśli parametr **&P** jest członem samodefiniującym, wówczas **T'&P** równa się **N**, jeśli rozkazem maszynowym charakterystyka typu równa jest **I**, jeśli stałą znakową — **C**, stałą binarną — **B**, makroinstrukcją — **M**, stałą stałoprzecinkową typu F — **F**, spakowaną liczbą dziesiętną — **P**, rozpakowaną liczbą dziesiętną — **Z** itp.

Wyrażenie logiczne stanowi jeden lub kilka składników związanych logicznymi operatorami **AND, OR** lub **NOT.** Składnikiem wyrażenia logicznego mogą być: wartości logiczne 0 lub 1 ujęte w nawiasy, relacja arytmetyczna lub znakowa. Zamiast znaków relacji używać należy oznaczeń: **EQ** (równe), **NE** (nierówne), **LT** (mniejsze), **GT** (większe), **LE** (mniejsze lub równe), **GE** (większe lub równe). Przykład wyrażenia logicznego:

((&PR+1) * 3 GE 15)

PRZYKŁAD 53

Dany przykład ilustruje sposób użycia w makrodefinicji parametrów zmiennych, mających w tym wypadku nazwy **A** i **B**.

```
        MACRO
        MADE    &P1, &P2, &GAMMA, &ALFA = X'4Ø'

        LCLA    &A
        LCLC    &B
&A      SETA    &P1+&P2
&B      SETC    'C''Y'''
        MVI     &A, &ALFA
        CLC     &A, &B
        BE      &GAMMA
        MEND
```

Makroinstrukcja:

```
        MADE    SWITCH,2,JUMP
```

spowoduje wygenerowanie następujących instrukcji maszynowych:

```
        MVI     SWITCH+2,X'4Ø'
        CLC     SWITCH+2,C'Y'
        BE      JUMP
```

■

Instrukcje warunkowe i skoki

Instrukcje skoków umożliwiają sterowanie przebiegiem translacji i generowanie różnych wariantów postaci wynikowych makroinstrukcji w zależności od spełnienia pewnych warunków. Instrukcją skoku warunkowego jest instrukcja **AIF**;

etykieta | *spacje* **AIF** *(wyrażenie-logiczne). etykieta*

Na przykład instrukcja:

AIF (&P1 OR &P2) .A3

oznacza, że nastąpi skok do etykiety .A3 wówczas, gdy wyrażenie logiczne w nawiasach będzie prawdziwe.

Skok bezwarunkowy następuje wskutek instrukcji **AGO**:

etykieta	*spacje*	**AGO**	*etykieta*

Niektóre inne instrukcje makroassemblerowe

Instrukcja **MNOTE** umożliwia wprowadzenie komunikatów w toku generacji instrukcji maszynowych. Ma ona następującą postać:

nazwa	*spacje*	**MNOTE**	*m, 'tekst'*

gdzie *m* — oznacza całkowitą dodatnią liczbę dziesiętną bez znaku ($0 \leqslant m \leqslant 255$), którą używa się jako numeru drukowanego komunikatu. Drukowanym komunikatem jest *'tekst'* podany w instrukcji.

Instrukcja **MEXIT** powoduje zakończenie przetwarzania danej makrodefinicji. Translator przechodzi do przetwarzania instrukcji następującej bezpośrednio po makroinstrukcji odpowiadającej danej makrodefinicji. Instrukcja **MEXIT** nie ma argumentów.

PRZYKŁAD 54

Zadaniem przedstawionej makroinstrukcji **CARRY** jest przeniesienie określonej ilości bajtów z pewnego pola pamięci do innego. Parametr **&TOTAL** określa liczbę przesyłanych bajtów, parametr **&FROM** wyznacza pole, z którego dane mają być przesłane, natomiast parametr **&TO** wskazuje pole, do którego dane są wprowadzane.

```
         MACRO
         CARRY   &TOTAL,&FROM,&TO
         LCLA    &A&B
         AIF     ('T',&TOTAL NE 'N').ERR
&A       SETA    &TOTAL
.REPEAT  AIF     (&A LE 256).EXC
         MVC     &TO+&B.(256),&FROM+&B
&A       SETA    &A — 256
```

```
&B      SETA    &B+256
        AGO     .REPEAT
.EXC    MVC     &TO+&B.(&A),&FROM+&B
        MEXIT
.ERR    MNOTE   7,'BLEDNY PARAMETR TOTAL'
        MEND
```

Przykładowo makroinstrukcja

```
        CARRY   600,POLE1,POLE2
```

spowoduje wygenerowanie następującego ciągu instrukcji maszynowych:

```
        MVC     POLE2(256),POLE1
        MVC     POLE2+256(256),POLE1+256
        MVC     POLE2+512(88),POLE1+512
```

5. Organizacja programu i translacja

Adresowanie

W komputerach Jednolitego Systemu pamięć adresuje się przez podanie rejestru bazy oraz przesunięcia względem tej bazy. Jeśli występuje jeszcze indeksowanie, adres rzeczywisty równy jest sumie zawartości rejestrów bazowego i indeksowego oraz przesunięcia. Programista może jednak zaoszczędzić sobie trudu obliczania adresów rzeczywistych i posługiwać się adresami symbolicznymi. Wówczas należy wskazać assemblerowi, że jeden lub kilka rejestrów przeznacza się na rejestry bazy, a assembler sam będzie zajmował się przypisywaniem rejestrów bazy i obliczaniem przesunięć. W tym celu programista używa instrukcji **USING**. Ma ona następującą postać:

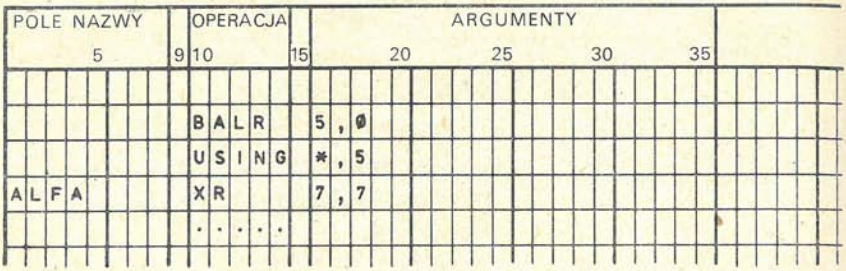
```
USING v,R1,R2,...,Rn
```

Instrukcja ta daje assemblerowi informację o tym, że w rejestrze R_1 znajduje się adres bazy równy v (v — wyrażenie bezwzględne), w rejestrze R_2 wartość $v+4096$, w R_3 wartość $v+8192$ itd. Maksymalna ilość rejestrów, jakiej można użyć w jednej instrukcji **USING** zależy od pojemności pamięci operacyjnej maszyny i wynosi od 5 do 16 rejestrów.

Należy zwrócić uwagę na to, że instrukcja **USING** nie ładuje żadnych liczb do wymienionych rejestrów, lecz tylko wskazuje assemblerowi, jakich rejestrów może używać jako rejestrów bazowych. Rejestry te

należy uprzednio załadować za pomocą innych instrukcji. Z tego powodu program w języku ASSEMBLER zaczyna się zwykle od instrukcji **BALR.** Typowy początek programu przedstawia następujący przykład.

PRZYKŁAD 55



Instrukcja **BALR** powoduje załadowanie do rejestru 5 adresu najbliższej kolejnej instrukcji maszynowej (przy wskazaniu rejestru 0 jako drugiego argumentu skok nie następuje), w tym wypadku instrukcji **ALFA.** Instrukcja **USING** daje assemblerowi informację o tym, że w rejestrze 5 znajduje się bieżąca wartość licznika adresów (gwiazdka reprezentuje położenie pierwszego bajtu pamięci dostępnej w danej chwili).

Przydzielając rejestry bazy należy zachować ostrożność w stosowaniu rejestrów 0,1,13,14 i 15, ponieważ są one używane również przez system operacyjny.

Dzielenie programu na sekcje

Pisząc duże programy w języku ASSEMBLER wygodnie jest podzielić program na sekcje. Każda sekcja może być translowana oddzielnie, a następnie wszystkie sekcje można połączyć w jeden program. Język ASSEMBLER dysponuje środkami programowymi zapewniającymi połączenia między sekcjami, gdyż w programie wielosekcyjnym musi istnieć możliwość przekazywania sterowania z jednej sekcji do drugiej.

W języku ASSEMBLER nie występuje w zasadzie pojęcie programu — istnieje jedynie moduł źródłowy, składający się z jednej lub więcej sekcji programowych. Ponieważ większość problemów można z powodzeniem rozwiązać, pisząc programy w postaci modułu źródłowego, składającego się z jednej sekcji, ograniczymy się wyłącznie do tego wypadku.

Każdej sekcji programowej należy nadać adres początkowy za pomocą instrukcji **START.** Gdy program składa się z jednej sekcji programowej

instrukcję **START** można pominąć. W takim wypadku assembler przyjmuje, że dana jest instrukcja:

spacje **START** *spacje*

i przyjmuje 0 jako umowny adres początkowy. Adres ten, podobnie jak i wszystkie inne adresy nadawane za pomocą zdania **START**, jest adresem względnym. Adres ten może dowolnie zmienić program REDAKTOR. Jeśli jest to adres zerowy, REDAKTOR umieści dany program w pamięci bezpośrednio za programem SUPERVISOR.

Sterowanie translacją

Do sterowania translacją i wydrukiem kontrolnym służy kilkanaście instrukcji — zdań sterujących, które najczęściej nie są generowane w module wynikowym. Najważniejsze z nich omówimy.

Instrukcje sterujące translacją perforuje się na kartach w podobnym formacie, jak instrukcje maszynowe, to jest:

nazwa	symbol instrukcji	argumenty
1 ••• 8	10 ••• 14	16 ••• 71

TITLE — nagłówek.
Instrukcję tę pisze się w postaci:

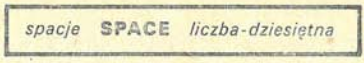
nazwa-lub-spacja **TITLE** *'nagłówek'*

W polu nazwy pisze się zwykle spacje. Można tu również postawić do czterech znaków identyfikacyjnych, które będą perforowane w kolumnach 73-76 kart modułu wynikowego. W polu argumentów można napisać nagłówek o długości do 100 znaków, który będzie wydrukowany na każdej stronie tabulogramu kontrolnego.
EJECT — nowa strona.
Format instrukcji:

spacje **EJECT** *spacje*

Z chwilą pojawienia się instrukcji **EJECT** następna instrukcja będzie drukowana na wydruku kontrolnym od początku strony.
SPACE — odstępy.

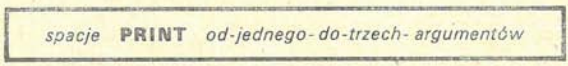
Za pomocą instrukcji **SPACE** o formacie:



można drukować puste wiersze, a więc robić na wydruku odstępy w ilości podanej w argumencie. Brak argumentu powoduje przepuszczenie jednego wiersza.

PRINT — warunki wydruku.

Zadanie **PRINT** ma postać:



Za pomocą instrukcji **PRINT** można zadawać warunki wydruku tekstu programu, używając następujących argumentów:

ON — wydruk kontrolny jest sporządzany,

OFF — nie drukuje się tabulogramu kontrolnego,

GEN — drukuje się wszystkie rozkazy generowane przez makroinstrukcje,

NOGEN — drukuje się makroinstrukcje, bez generowanych przez makroinstrukcje rozkazów.

DATA — drukuje się w całości wszystkie stałe,

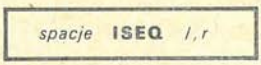
NODATA — drukuje się tylko 8 pierwszych cyfr szesnastkowych wszystkich stałych.

Jeśli instrukcja **PRINT** nie występuje w programie, assembler zachowuje się tak, jak gdyby została podana instrukcja:

PRINT ON,NODATA,GEN

ISEQ — sprawdzanie kolejności kart źródłowych.

Za pomocą instrukcji **ISEQ** można sprawdzać kolejność numeracji kart programu źródłowego. W tym celu w instrukcji, którą zapisuje się w następujący sposób:



podaje się pierwszą kolumnę *l* i ostatnią *r* pola, w którym prowadzona jest numeracja kart. Nie można prowadzić numeracji w polu zadań, tzn. w kolumnach 1—72. Sprawdzanie zaczyna się od następnej karty po karcie **ISEQ** i trwa do momentu napotkania następnej karty **ISEQ**, w której jako argument występują spacje.

PUNCH — perforacja.
Instrukcja ta używana jest w postaci:

spacje **PUNCH** *'dane'*

Dane, do 80 znaków ujętych w apostrofy, są perforowane na karcie i trafiają do modułu wynikowego.
REPRO — reprodukcja.
Instrukcja **REPRO** powoduje, że z następnej karty przenosi się dane bez żadnych zmian na kartę modułu wynikowego:

spacje **REPRO** *spacje*

Za pomocą tych instrukcji można wstawić w odpowiednie miejsca modułu wynikowego dodatkową informację, np. karty sterujące dla programu REDAKTOR.
EQU — ekwiwalentność.
Postać instrukcji:

nazwa **EQU** *wyrażenie*

Działanie instrukcji polega na tym, że najpierw oblicza się wartość wyrażenia i wartość tę nadaje się nazwie podanej w instrukcji.
ORG — ustaw licznik adresów.
Translacji programu dokonuje się w dwóch przebiegach. W pierwszym przebiegu assembler kolejno analizuje instrukcje i stałe. Uruchamia przy tym licznik adresów, którego wartość początkowa równa jest zeru lub ustawiona jest za pomocą instrukcji **START**. Dla każdej etykiety w programie translator określa jej adres w stosunku do początkowej wartości licznika adresów i dane te umieszcza w odpowiednich tablicach. Z każdą instrukcją zawartość licznika adresów zmienia się, w zależności od długości instrukcji, stałej lub pola pamięci. Za pomocą instrukcji **ORG** można w liczniku adresów ustawić nową wartość. Instrukcja występuje w następującej postaci:

spacje **ORG** *spacje-lub-wyrażenie* .

W wyniku działania instrukcji **ORG** licznik zwiększy się o wartość obliczoną z wyrażenia. Jeśli poprzednie zadanie **ORG** zmniejszyło licznik

rozkazów, wówczas użycie instrukcji **ORG** bez argumentu powoduje przywrócenie poprzedniej najwyższej wartości licznika zwiększonej o jeden.
CNOP — warunkowy brak działania.
Instrukcja ta ma postać.

spacje **CNOP** *b,w*

Za pomocą instrukcji **CNOP** można ustawić instrukcje w pamięci na określonej granicy. Jeśli pewne bajty muszą być pominięte, assembler generuje instrukcje „nic nie rób", zmieniając przy tym w żądany sposób zawartość licznika adresów. Argument *b* wskazuje, który bajt słowa lub podwójnego słowa ma być ustawiony w liczniku, a argument *w* określa czy bajt ten znajduje się w słowie ($w = 4$), czy też w podwójnym słowie ($w = 8$). Działanie tej instrukcji ilustruje następujący przykład.

PRZYKŁAD 56

Przypuśćmy, że w pewnym momencie program doprowadził licznik adresów do wartości odpowiadającej granicy podwójnego słowa (co oznacza, że zawiera liczbę równą całkowitej wielokrotności 8). Następnie pojawiają się dwie instrukcje:

```
CNOP  6,8
BALR  3,10
```

W module wynikowym instrukcja **CNOP** zostanie zastąpiona ciągiem instrukcji **BCR Ø,Ø** („nic nie rób"):

```
BCR   Ø,Ø
BCR   Ø,Ø
BCR   Ø,Ø
BALR  3,1Ø
```

W rezultacie instrukcja **BALR** zostanie umieszczona na granicy ostatniego półsłowa podwójnego słowa, a biorąc pod uwagę jej własną długość, spowoduje załadowanie do rejestru 3 liczby odpowiadającej granicy podwójnego słowa. ■
END — koniec translacji.
Instrukcja ta ma postać:

spacje **END** *spacje*

Instrukcja **END** powinna być ostatnią instrukcją programu. Przekazuje ona assemblerowi informację o tym, że nie ma już więcej zdań źródłowych do translacji.

6. Programowanie wejścia/wyjścia

Programowanie operacji we/wy[4] w maszynach Jednolitego Systemu może być dokonywane w trzech poziomach:

1) poziom instrukcji maszynowych,
2) fizyczny poziom sterowania we/wy,
3) logiczny poziom sterowania we/wy.

Pozycja pierwsza dotyczy sterowania wejściem/wyjściem za pomocą instrukcji maszynowych, takich jak **SIO, TIO** itp. Instrukcje te omawialiśmy w rozdziale II. Ponieważ taki sposób programowania jest bardzo skomplikowany i pracochłonny, nie jest on przeznaczony dla użytkownika systemu.

Fizyczny system sterowania wejściem/wyjściem (SSWW) jest zbiorem makroinstrukcji, za pomocą których można łatwo programować operacje we/wy. Fizyczny SSWW wymaga znajomości działania urządzeń zewnętrznych, a w szczególności kodów operacji we/wy, wykonywanych w tych urządzeniach.

W bardziej skomplikowanych wypadkach przetwarzania dużych zbiorów danych o różnych strukturach korzystniej jest posługiwać się logicznym SSWW. Logiczny SSWW przedstawiamy w rozdziale VIII, natomiast w tym rozdziale zajmiemy się wyłącznie fizycznym SSWW.

Zadaniem programisty pracującego na fizycznym poziomie sterowania we/wy jest przygotowanie programu kanałowego, utworzenie bloku sterowania operacjami we/wy (CCB) i odwołanie się do programu SUPERVISOR w celu wykonania operacji we/wy. Czynności te wykonuje się za pomocą instrukcji **CCB, EXCP, WAIT.**

Blok sterowania operacjami we/wy, zwany blokiem CCB, tworzy się za pomocą makroinstrukcji CCB:

[*nazwa*] **CCB SYS***nnn, adres-CCW* [,**X'** *nnn'*] [, *adres bajtu sprecyzowania stanu'*]

[4] Por. C. Germain, *Programming the IBM/360*. Prentice-Hall, Inc., 1967; J.R. Jagielski, *Operowanie danymi w DOS/JS*, „Informatyka" 1974, nr 12; DOS/JS *Organizacje wejście-wyjście;* DOS/JS *Makroinstrukcje wejście-wyjście.*

W tej instrukcji nazwa jest taką samą nazwą symboliczną, która poprzedza makroinstrukcje **EXCP** i **WAIT.** SYS*nnn* wskazuje urządzenie logiczne, w którym wykonuje się operację we/wy. Mogą to być urządzenia SYSRDR, SYSLST, SYSIPT, SYSLOG, SYSPCH, SYSRES, SYSØØØ — SYS244. Danemu urządzeniu logicznemu musi być przydzielone konkretne urządzenie fizyczne za pomocą operatora **ASSGN.** Drugi argument określa adres symboliczny pierwszej instrukcji CCW. Dwa następne argumenty nie są obowiązkowe i oznaczają: X'*nnn*' — liczba szesnastkowa używana dla ustalenia bitów w bajtach łączności (o bajtach łączności mówiliśmy w rozdziale V), natomiast czwarty argument wskazuje symboliczny adres pola pamięci, gdzie system zapisuje bajty sprecyzowania stanu.

Na podstawie parametrów zadanych w makroinstrukcji **CCB** system tworzy tablicę — blok sterowania operacjami we/wy. W większości wypadków wystarczy ograniczyć się do makroinstrukcji **CCB** z dwoma obowiązkowymi parametrami.

Operację we/wy rozpoczyna makroinstrukcja **EXCP** — wykonać rozkaz kanałowy:

[*nazwa*] **EXCP** *nazwa-bloku* CCB

Instrukcja **EXCP** daje SUPERVISOROWI sygnał o konieczności rozpoczęcia operacji we/wy w urządzeniu logicznym podanym w CCB-SUPERVISOR określa fizyczne urządzenie we/wy i ustawia daną operację w kolejce do tego urządzenia (por. rys. 43). Następnie, jeśli nie była podana instrukcja **WAIT,** program problemowy jest wykonywany od następnej instrukcji po **EXCP,** równolegle z operacją we/wy.

Makroinstrukcja **WAIT** (czekać) powoduje wstrzymanie pracy programu aż do czasu zakończenia operacji we/wy:

[*nazwa*] **WAIT** *nazwa-bloku*-CCB

Makroinstrukcja **WAIT** konieczna jest w tych wypadkach, gdy kontynuowanie programu jest możliwe tylko po wprowadzeniu danych niezbędnych dla dalszego przetwarzania.

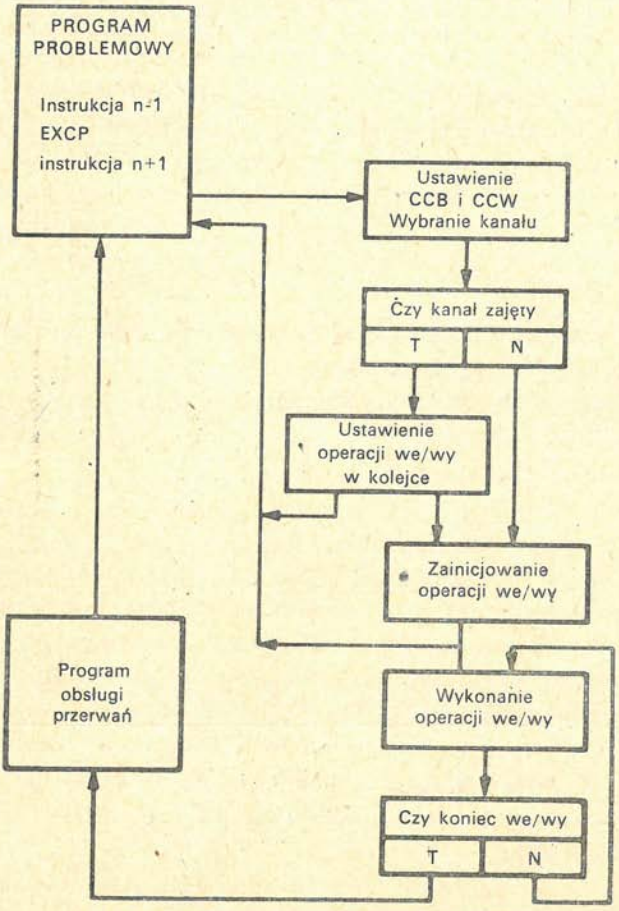
Program kanałowy tworzą instrukcje **CCW.** Instrukcją **CCW** zajmowaliśmy się szczegółowo w rozdziale II. W języku ASSEMBLER CCW można zapisać jako specyficzną stałą w postaci:

[nazwa] CCW, operacja, adres, wskazniki, licznik

Wszystkie cztery argumenty muszą wystąpić w instrukcji. Znaczenie ich jest następujące:

operacja — wartość bezwzględna, oznaczająca kod operacji we/wy dla określonego urządzenia,

adres — wyrażenie bezwzględne lub symboliczne, wyznaczające adres pola danych, do (z) którego mają być przesyłane dane,



Rys. 43. Fizyczny system sterowania we/wy

wskaźniki — wartość bezwzględna określająca sposób realizacji operacji we/wy (łańcuch danych, łańcuch rozkazów itd.),

licznik — wartość bezwzgledna określająca liczbę przesyłanych bajtów

PRZYKŁAD 57

Instrukcja ta dotyczy czytnika kart.

```
POLE NAZWY:    OPERACJA        ARGUMENTY
5          9   10      15  20        25        30        35        40
           CCW             X'02',POLE1,X'00',80
```

Spowoduje ona czytanie karty perforowanej i 80 bajtów danych zostanie wprowadzonych do pamięci głównej pod adres **POLE1**.

Ilustracją stosowania makroinstrukcji fizycznego SSWW, jak też niektórych instrukcji assemblerowych może być następujący, przykładowy, program.

PRZYKŁAD 58

Zadaniem programu **EXAMPLE** jest wczytywanie pliku kart perforowanych zawierających następujące dane:

kolumny: 1 — 8 symbol wyrobu,
9 — 11 spacje,
12 — 31 nazwa wyrobu,
32 — 36 cena,
37 — 40 ilość,
41 — 80 spacje.

Po wczytaniu każdej karty, program oblicza wartość (mnożąc cenę przez ilość) i drukuje zawartość karty łącznie z obliczoną wartością. Po przeczytaniu ostatniej karty, która wyróżnia symbol wyrobu równy 99999999, program drukuje sumę wartości uzyskanych ze wszystkich kart. W programie tym używamy tylko liczb całkowitych. Program w postaci wydruku kontrolnego przedstawiamy na rysunku 44. Wydruk taki można łatwo rozszyfrować, biorąc pod uwagę, że nazwy użyte w nagłówku mają następujące znaczenie:

LOC — względny adres pamięci (pierwsza instrukcja programu ma adres zerowy),

OBJECT CODE — instrukcja w postaci maszynowej,
ADDR1 — adres względny pierwszego argumentu,
ADDR2 — adres względny drugiego argumentu,
STMT — kolejny numer zdania źródłowego,
SOURCE STATEMENT — zdanie źródłowe (ta część wydruku kontrolnego jest tekstem programu przeniesionym bez żadnych zmian z kart).

Program ten nie mógłby być wykonany bez udziału pewnych operatorów sterujących, niezbędnych dla symbolu operacyjnego. Całość składa się z następujących kart:

```
// JOB EXAMPLE
// OPTION LINK
// EXEC ASSEMBLF
   TITLE 'EXAMPLE'
   PRINT NOGEN
/zdania  \
\źródłowe/
// EXEC LNKEDT
// EXEC
  ⟨dane⟩
/*
/&
```

Wydruk kontrolny zawiera więcej informacji, niż udało nam się pokazać na rysunku 44. Po zdaniach źródłowych następują tablice. Dużą pomoc może okazać tablica zawierająca alfabetyczny wykaz wszystkich używanych nazw wraz z ich długością i wartością oraz adresami, dzięki którym łatwo stwierdzić, w jakich zdaniach źródłowych zostały one użyte.

Jeśli translator nie wykryje żadnych błędów formalnych, na wydruku kontrolnym pojawia się zdanie:

NO STATEMENTS FLAGGED IN THIS ASSEMBLY

i rozpoczyna zwykle pracę program REDAKTOR. W razie błędów w programie translator drukuje odpowiednie komunikaty diagnostyczne.

```
**** EXAMPLE ****

LOC OBJECT CODE ADDR1 ADDR2 STMT SOURCE

000000 0530                                    3
000002                                         4
                                               5 READ
                                               9
000016 D507 3082  3146  00084  00148           15
00001C 4780 3060                00062          16
000020 F223 30F9  30A6  000FB  000A8           17
000026 F224 30F6  30A1  000F8  000A3           18
00002C F852 30DE  30F6  000E0  000F8           19
000032 FC52 30DE  30F9  000E0  000FB           20
000038 F3B5 30D2  30DE  000D4  000E0           21
00003E D300 30DD  315A  000DF  0015C           22
000044 FA55 30F0  30DE  000F2  000E0           23
                                               24
                                               28
00005E 47F0 3000                00002          34
000062 F3B5 30E4  30F0  000E6  000F2           35 END
000068 D300 30EF  315A  000F1  0015C           36
                                               37
                                               41
                                               47
000084                                         50 ZAPIS
000084                                         51 SYMBOL
00008C                                         52
00008F                                         53 NAZWA
0000A3                                         54 CENA
0000A8                                         55 ILOSC
0000AC                                         56
0000D4                                         57 WART
0000E0                                         58 WARTP
0000E6                                         59 SUMA
0000F2 00000000000C                            60 SUMAP
0000F8                                         61 CENAP
0000FB                                         62 ILOSCP
                                               63 IN
00010E 0000
000110 0200008400000050                        74 INCCW
                                               75 OUT1
000128 090000842000005C                        86 OUTCCW1
                                               87 OUT2
000140 090000E62000000C                        98 OUTCCW2
                                               99
000148 F9F9F9F9F9F9F9F9                       100
000150 000000FE                               101
000154 00000118                               102
000158 00000130                               103
00015C FF                                     104
```

```
PAGE :
STATEMENT                FDOS/ES V. M 1.3 06/01/76
BALR   3,0
USING  *,3
EXCP   IN                          WCZYTANIE KARTY DANYCH
WAIT   IN
CLC    SYMBOL(8), = 8X'F9'         SPRAWDZENIE CZY OSTATNIA KARTA
BC     8, END
PACK   ILOSCP(3),ILOSC(4)
PACK   CENAP(3),CENA(5)
ZAP    WARTP(6),CENA(3)
MP     WARTP(6),ILOSCP(3)          OBLICZENIE WARTOŚCI
UNPK   WART(12),WARTP(6)
MVZ    WART+11(1), = X'FF'
AP     SUMAP(6),WARTP(6)
AP     SUMAP(6),WARTP(6)           OBLICZENIE BIEŻĄCEJ SUMY
EXCP   OUT1                        WYDRUK KARTY
WAIT   OUT1
BC     15,READ                     SKOK DO "READ"
UNPK   SUMA(12),SUMAP(6)
MVZ    SUMA+11(1), = X'FF'
EXCP   OUT2                        WYDRUK SUMY
WAIT   OUT2
EOJ
DS     CL80
DS     CL8
DS     CL3
DS     CL20
DS     CL5
DS     CL4
DS     CL40
DS     CL12
DS     CL6
DS     CL12
DC     PL6'0'
DS     CL3
DS     CL3
CCB    SYSIPT,INCCW
CCW    X'02',ZAPIS,0,80
CCB    SYSLST,OUTCCW1
CCW    X'09',ZAPIS,X'20',92
CCB    SYSLST,OUTCCW2
CCW    X'09',SUMA,X'20',12
END
       = 8X'F9'
       = A(IN)
       = A(OUT1)
       = A(OUT2)
       = X'FF'
```

VII. PL/1

Język programowania PL/1 (ang. *Programming Language* Nr 1) jest uniwersalnym językiem, opracowanym w latach 1963—1966 przez firmę IBM. Powstał on na podstawie doświadczeń uzyskanych przy opracowywaniu i użytkowaniu języków ALGOL 60, COBOL, FORTRAN, JOVIAL i dlatego zawiera wiele elementów tych języków. Mimo to PL/1 jest zupełnie nowym tworem, mającym wiele nowych właściwości. Celem autorów PL/1 było opracowanie języka, przydatnego do rozwiązywania zadań dowolnego typu, zarówno numerycznych jak i nienumerycznych, który umożliwiłby programiście jak najlepsze wykorzystanie możliwości dużej maszyny cyfrowej, nie uciekając się przy tym do języka maszynowego.

Oczywiście, zawarcie w jednym języku wielu dodatnich i pożytecznych cech zaczerpniętych z różnych języków doprowadziło do znacznego skomplikowania języka jak i jego translatora (dla przykładu, translator języka PL/1 wersji F systemu IBM-360 jest programem składającym się prawie z pół miliona instrukcji, czyli więcej niż translatory języków COBOL, FORTRAN, APL i ALGOL razem wzięte). Jednakże język PL/1 ma tę zaletę, że użytkownik może nie znać całego języka, lecz jedynie pewne jego podzbiory, których opanowanie nie sprawia większych trudności, szczególnie osobom znającym jeden z rozpowszechnionych języków algorytmicznych.

Język PL/1 dysponuje bogatym repertuarem środków do rozwiązywania zagadnień numerycznych, zadań przetwarzania danych, przetwarzania symboli, modelowania, rozwiązywania zadań logicznych, przetwarzania w czasie rzeczywistym, opracowywania oprogramowania podstawowego maszyn cyfrowych. Program problemowy w języku PL/1 może być tak opracowywany, aby jego części mogły być przetwarzane równolegle. Poza tym pewne fragmenty programu mogą być pisane w języku ASSEM-

BLER. Możliwe jest też zadanie odwrotne — program pisany w języku ASSEMBLER może mieć wstawki w języku PL/1.

W rozdziale tym przedstawimy podzbiór języka PL/1, zbliżony do tzw. Subset-PL/1, przystosowany do przetwarzania pod kierunkiem DOS JS[1]. Programy napisane za pomocą tego podzbioru języka PL/1 mogą być realizowane na wszystkich modelach serii RIAD z wyjątkiem R-10.

1. Wprowadzenie

Alfabet języka PL/1

W języku PL/1 używa się 60 znaków (29 liter, 10 cyfr i 21 znaków specjalnych):

A,B,...,Z — duże litery alfabetu łacińskiego,
0,1,...,9 — cyfry dziesiętne,
$ — symbol waluty,
@ — znak handlowy,
# — symbol numeru,
— odstęp (spacja),
= — znak równości,
+ — plus,
– — minus,
* — gwiazdka (znak mnożenia),
/ — znak dzielenia,
(— nawias otwierający,
) — nawias zamykający,
, — przecinek,
. — kropka,
' — apostrof,
% — symbol procentu,
; — średnik,
: — dwukropek,
¬ — symbol negacji (NIE),
& — symbol koniunkcji (I),
| — symbol alternatywy (LUB) -
> — znak większości,

[1] Por. *DOS/JS PL/1-opis języka; DOS/JS PL/I — wstęp do programowania; DOS/JS PL/I w systemie DOS/JS.*

< — znak mniejszości,
— — łącznik (znak podkreślania),
? — znak zapytania.

Trzy znaki specjalne: $, @ i # traktowane są jako litery. Niekiedy używa się skróconej wersji alfabetu, składającego się z 48 znaków.

Kodowanie programów

Pisanie programów w języku PL/1 nie wymaga żadnych formularzy; program pisze się w sposób niepozycyjny. W DOS JS programy przygotowuje się na kartach w dowolnych kolumnach od drugiej do siedemdziesiątej drugiej. Jedno zdanie języka może zajmować kilka kart, albo też na jednej karcie można umieścić więcej niż jedno zdanie. Elementy języka powinny być rozdzielane ogranicznikami, którymi są znaki: spacja, kropka, przecinek, średnik, dwukropek, nawiasy okrągłe, wszystkie operatory i komentarze. Nie wolno więc pisać **FIXEDBINARY** zamiast **FIXED BINARY**. Między elementami języka może wystąpić dowolna ilość spacji.

Elementy programu

Przez *identyfikator* lub nazwę będziemy rozumieli dowolny ciąg liter, cyfr lub łączników rozpoczynający się od litery.

PRZYKŁAD 1

Prawidłowymi nazwami są: ALFA, XYZ, J23, NR_PRACOWNIKA FIXED.

Przykłady nieprawidłowych nazw: 135, _P1, 'NAZWA'. ■

Identyfikatory, które są częściami składowymi języka nazywają się słowami kluczowymi. Słowa kluczowe w języku PL/1 nie są zastrzeżone i programista może używać ich w innym kontekście.

Podstawowym składnikiem programu w języku PL/1 jest *zdanie*. Zdaniem jest ciąg elementów języka zakończony średnikiem, np.

```
DO K = 1 BY 2 TO 20;
ALFA = A * B1;
```

Są to przykłady zdań prostych. Przykładem zdania złożonego może być zdanie typu **IF**:

```
IF X>Y THEN GOTO E1; ELSE GOTO E2;
```

Identyfikatorem zdania jest jedno lub kilka słów kluczowych określających funkcję zdania. Mówimy np. o zdaniu **DO**, zdaniu **IF** itd.

Większymi jednostkami programu są grupy, bloki i procedury, które omówimy szczegółowo dalej.

2. Opisy danych

Dane są to obiekty przetwarzane w toku wykonywania programu. W języku PL/1 rozróżnia się dwa typy danych:

— dane problemowe,
— dane sterujące programem.

Dane problemowe mogą być danymi arytmetycznymi lub danymi tekstowymi, a w szczególności:

— liczby dziesiętne stałoprzecinkowe,	dane arytmetyczne
— liczby dziesiętne zmiennoprzecinkowe,	
— liczby binarne stałoprzecinkowe,	
— liczby binarne zmiennoprzecinkowe,	
— tekst bitowy,	dane tekstowe
— tekst znakowy.	

Ze względu na strukturę dane podzielimy na skalary, tablice i struktury. Skalarem jest stała lub zmienna, której wartościami są pojedyncze stałe. Pojęcia te omówimy w następnych rozdziałach.

Danymi sterującymi są etykiety i wskaźniki.

Dane problemowe i operacje nad nimi

Przed użyciem zmiennych w programie należy je uprzednio opisać za pomocą zdania **DECLARE**. W zdaniu tym, oprócz nazwy zmiennej, występują atrybuty zmiennej, które wskazują podstawę systemu liczenia (**BINARY** lub **DECIMAL**), sposoby reprezentacji liczby w maszynie (**FIXED** lub **FLOAT**), długości danej i tym podobne. Zdanie to ma następującą postać:

DECLARE *nazwa* [*atrybut*] ... [,*nazwa* [*atrybut*] ...] ...;

Zmienne arytmetyczne. Atrybuty zmiennych arytmetycznych mogą być następujące:

a. Atrybut podstawowy systemu liczenia.
Można używać jednego z dwóch następujących atrybutów:
DECIMAL — dziesiętny system liczenia,

BINARY — dwójkowy system liczenia.

b. Atrybut sposobu przedstawienia liczby.

Liczby mogą być zapisane w maszynie w systemie ze stałym przecinkiem lub ze zmiennym przecinkiem:

FIXED — stały przecinek,

FLOAT — zmienny przecinek.

c. Atrybut precyzji.

Atrybut precyzji ma ogólną postać

(*liczba-cyfr* [,*liczba-cyfr-ułamkowych*]).

Dla liczb dziesiętnych stałoprzecinkowych *liczba-cyfr* oznacza ogólną liczbę cyfr łącznie z cyframi po przecinku, których liczbę określa druga część parametru. Liczby stałoprzecinkowe binarne mogą być tylko liczbami całkowitymi, więc atrybut precyzji jest jedną liczbą, określającą ilość bitów, jakie będą brane pod uwagę. Dla liczb zmiennoprzecinkowych atrybut precyzji podaje minimalną liczbę cyfr mantysy, które będą brać udział w przetwarzaniu.

Tablica 18 przedstawia atrybuty danych arytmetycznych.

TABLICA 18

Atrybuty danych arytmetycznych

Sposób przedstawienia	Podstawa liczenia	Precyzja	Maksymalna długość	Domyślny atrybut precyzji
FIXED	DECIMAL	(p [,q])	p = 15 cyfr dziesiętnych q = 15 cyfr dziesiętnych	(5)
FIXED	BINARY	(p)	p = 31 cyfr binarnych	(15)
FLOAT	DECIMAL	(p)	p = 16 cyfr dziesiętnych mantysy	(6)
FLOAT	BINARY	(p)	p = 53 cyfr binarnych mantysy	(21)

PRZYKŁAD 2

DECLARE A FIXED DECIMAL (6,3);

Zdanie to opisuje zmienną **A** jako liczbę dziesiętną stałoprzecinkową, której wartość może zmieniać się w granicach

$$-999{,}999 \leqslant A \leqslant +999{,}999.$$ ■

Zasada domyślności w deklaracjach zmiennych. Jeśli w zdaniu **DECLARE** nie podano wszystkich atrybutów zmiennej, wtedy translator przyjmuje pewne atrybuty „automatycznie". Tak więc, jeśli opuszczono atrybut podstawy systemu liczenia, system przyjmuje atrybut **DECIMAL**. Jeżeli natomiast opuszczono atrybut sposobu przedstawienia liczby, translator postępuje tak jak gdyby podany był atrybut **FLOAT**. W tablicy 18 podano odpowiednie dla tych wypadków domyślne atrybuty precyzji. Jeśli opuszczony jest zarówno atrybut podstawy systemu liczenia jak i sposobu przedstawienia liczby, wtedy translator przyjmuje atrybuty na podstawie wyników analizy nazwy zmiennej. Jeżeli pierwszy znak nazwy zmiennej równy jest **I,J,K,L,M,N**, zmienna otrzymuje atrybuty **FIXED BINARY**, dla wszystkich innych nazw — **FLOAT DECIMAL**. Dla przykładu, zmienna o nazwie **ALFA** otrzyma atrybuty **FLOAT DECIMAL (6)**, zmienna **NUMER** będzie miała atrybuty **FIXED BINARY (15)**.

Stałe arytmetyczne. Stałe przedstawiają pewną wartość liczbową, nie wymagają więc nazw. Translator przypisuje stałym atrybuty w zależności od postaci, w jakiej występują w programie. Rozróżnia się cztery typy stałych arytmetycznych:

— dziesiętne stałoprzecinkowe,
— binarne stałoprzecinkowe,
— dziesiętne zmiennoprzecinkowe,
— binarne zmiennoprzecinkowe.

Stała dziesiętna stałoprzecinkowa składa się z jednej lub kilku cyfr dziesiętnych, ze znakiem lub kropką dziesiętną[2], które nie są jednak obowiązkowe. Jeśli brak kropki, to rozumie się, że postawiona jest ona z prawej strony najmniej znaczącej cyfry. Natomiast w wypadku braku znaku przyjmuje się, że liczba j est dodatnia.

PRZYKŁAD 3

Przykłady stałych dziesiętnych stałoprzecinkowych:

Stała	Przypisane atrybuty
329	**FIXED DECIMAL** (3,0)
+8	**FIXED DECIMAL** (1,0)
+8.0	**FIXED DECIMAL** (2,1)
−0.05	**FIXED DECIMAL** (3,2)
.05	**FIXED DECIMAL** (2,2)
−3.141	**FIXED DECIMAL** (4,3)

[2] Przyjęto, wzorem krajów anglosaskich, używać kropki zamiast przecinka dziesiętnego.

Stałą binarną stałoprzecinkową zapisuje się w postaci ciągu cyfr binarnych, bezpośrednio za którymi stoi litera **B**. Znak może wystąpić, lecz nie jest konieczny. Kropka nie jest dopuszczalna.

PRZYKŁAD 4

Przykłady liczb binarnych stałoprzecinkowych:

Stała	Przypisane atrybuty
+1011B	FIXED BINARY (4)
111B	FIXED BINARY (3)
−1B	FIXED BINARY (1)

■

Stała dziesiętna zmiennoprzecinkowa składa się z dwóch liczb — mantysy i cechy. Mantysa jest stałą dziesiętną stałoprzecinkową. Za mantysą następuje litera **E**, a następnie cecha ze znakiem lub bez znaku. Cecha może być jedno- lub dwucyfrową liczbą dziesiętną.

PRZYKŁAD 5

Przykłady stałych dziesiętnych zmiennoprzecinkowych:

Stała	Przypisane atrybuty	Wartość dziesiętna
31E2	FLOAT DECIMAL (2)	$31 . 10^{2}$
−0.1E-02	FLOAT DECIMAL (2)	$-10E^{-3}$
25E0	FLOAT DECIMAL (2)	25
1.000E-5	FLOAT DECIMAL (5)	$1.000 . 10E^{-5}$

Stałą binarną zmiennoprzecinkową zapisuje się podobnie jak stałą dziesiętną zmiennoprzecinkową, z tym że cecha nie może mieć więcej niż trzy cyfry dziesiętne zakończone literą **B**. Mantysa jest liczbą binarną stałoprzecinkową.

PRZYKŁAD 6

Przykłady stałych binarnych zmiennoprzecinkowych:

Stała	Przypisane atrybuty	Wartość binarna
10111E6B	FLOAT BINARY (5)	$10111 . 2^{6}$
.001101E-4B	FLOAT BINARY (6)	$0.1101 . 2E^{-6}$
−0.1E+25B	FLOAT BINARY (2)	$-0.1 . 2^{25}$

Dane arytmetyczne w postaci znakowej. Dane arytmetyczne mogą występować również w postaci zewnętrznej takiej samej jak znaki alfabetyczne.. Opisuje się je wtedy atrybutem **PICTURE**:

DECLARE *nazwa-zmiennej* **PICTURE** *'specyfikacja-szablonu'*

Przez szablon rozumie się ciąg znaków oznaczający elementy liczby Oto niektóre znaki używane w szablonach:

9 — oznacza, że w tej pozycji danej powinna znajdować się cyfra,

V — wskazuje miejsce dla kropki dziesiętnej, jednakże w pamięci miejsca dla kropki nie rezerwuje; kropka, jak również przecinek w liczbie nie występują,

Z — oznacza, że w danej pozycji powinna być cyfra dziesiętna; jeżeli jednak cyfra ta jest nieznaczącym zerem, zostaje zamieniona na spację,

S — oznacza plus, jeśli arytmetyczna wartość liczby $\geqslant 0$, w przeciwnym wypadku minus,

B — w danej pozycji wstawia się spację,

. — w tej pozycji zostanie wstawiona do liczby kropka,

, — w tej pozycji dana zawierać będzie przecinek.

PRZYKŁAD 7

Przykłady liczb w postaci znakowej:

Wartość liczby	Specyfikacja szablonu	Postać liczby w maszynie
23.351	99V999	23351
0005.1	ZZZZV99	␣␣␣510
−0.092	S9V999	−0092
00030	ZZ999	␣␣030
310	ZZZ	310

Dla skrócenia zapisu można podawać w nawiasach ilość powtarzających się znaków szablonu, np. deklaracja

DECLARE *identyfikator* **PICTURE** '999999.V99';

jest równoważna deklaracji

DECLARE *identyfikator* **PICTURE** '(6)9.V99';

Dane tekstowe. Dane tekstowe mogą być dwóch typów:

— łańcuch znaków alfabetycznych (tekst znakowy),

— łańcuch bitów (tekst bitowy).

Łańcuchem jest dowolny ciąg znaków.

Tekstem znakowym jest dowolny łańcuch znaków wchodzących w skład alfabetu języka PL/1. Zmienną tego typu opisuje się następującym zdaniem:

DECLARE *nazwa* **CHARACTER** *(liczba-całkowita-bez-znaku)*;

Atrybut *liczba-całkowita-bez-znaku* wskazuje ilość znaków w łańcuchu. Długość ta musi mieścić się w granicach 1-255.

PRZYKŁAD 8

DECLARE SYMBOL CHARACTER (13);

W tym wypadku translator zarezerwuje dla zmiennej **SYMBOL** 13 bajtów pamięci.

Ten sam skutek można osiągnąć używając atrybutu **PICTURE '(13)X'**, ponieważ znak X oznacza także dowolny znak alfabetu języka PL/1. ■

Tekst bitowy jest to dowolny ciąg znaków binarnych (0 lub 1). Zmienne typu łańcuch bitów, czyli tekst bitowy, opisuje się następującym zdaniem:

DECLARE *nazwa* **BIT** *(liczba-całkowita-bez-znaku)*;

Liczba-całkowita-bez-znaku określa liczbę bitów w zmiennej. Nie może być ona większa od 64.

PRZYKŁAD 9

DECLARE INDEKS BIT (15);

Zmienna **INDEKS** jest tekstem bitowym o długości 15 znaków binarnych.

■

Stałe tekstowe. Stałe typu tekst znakowy zapisuje się przez ograniczenie dowolnych znaków alfabetu języka PL/1 apostrofami. Znakiem jest również spacja. Dla umieszczenia w stałej apostrofu należy pisać nie jeden, lecz dwa apostrofy obok siebie.

PRZYKŁAD 10

Przykłady stałych tekstowych:

Stałe	Atrybuty przypisane przez translator
'JEZYK PL/1'	CHARACTER (10)
'ALGEBRA BOOLE''A'	CHARACTER (15)

PRZYKŁAD 11

Można używać skróconego zapisu stałych tekstowych. Oto dwa przykłady: (5) '*' równoważne jest '*****', a
(120)' — ' będzie na wydruku linią przez całą szerokość wiersza.
Stałe typu tekst bitowy zapisuje się również w apostrofach, a za drugim apostrofem następuje bezpośrednio litera B.

PRZYKŁAD 12

Przykłady stałych typu tekst bitowy:
'1'B
(15)'Ø'B. ■

Operacje arytmetyczne

W języku PL/1 używa się następujących symboli do oznaczenia operacji arytmetycznych:
+ — dodawanie,
− — odejmowanie,
* — mnożenie,
/ — dzielenie,
** — potęgowanie.
Są to operacje dwuargumentowe. Symbole + i − mogą być używane również jako operatory jednoargumentowe, jeśli występują w charakterze znaków liczb.

Przy wykonywaniu operacji arytmetycznych translator sprawdza, czy liczby biorące udział w operacji mają jednakowe podstawy systemu liczenia, sposoby przedstawienia, formaty i typy wewnętrznej reprezentacji. Jeżeli liczby zapisane są różnymi sposobami, przed wykonaniem operacji następuje przekształcenie danych w celu ich ujednolicenia, zgodnie z podanymi regułami. Programista powinien jednak pamiętać, aby przekształceń

takich było w programie jak najmniej, ponieważ wydłużają one czas przetwarzania.

Zasady przekształcania danych w wyrażeniach arytmetycznych:

1) arytmetyczne dane w postaci znakowej zostają przekształcane w dane dziesiętne stałoprzecinkowe,

2) jeśli argumenty mają różne podstawy systemu liczenia, np. **DECIMAL** i **BINARY** to argument dziesiętny zamieniany zostaje na binarny,

3) jeśli argumenty różnią się tylko atrybutami precyzji, nie ma przekształceń, a operacje wykonuje się według reguł, które będą widoczne na przykładach,

4) jeśli argumenty przedstawione są jako **FLOAT** i **FIXED**, wówczas argument stałoprzecinkowy zamieniany jest na argument zmiennoprzecinkowy. Wyjątkiem jest operacja potęgowania.

Ważną umiejętnością jest właściwe przewidywanie atrybutów wyników operacji. Jeżeli argumenty są liczbami zmiennoprzecinkowymi, to wynik jest również liczbą zmiennoprzecinkową. Gdy argumenty są liczbami stałoprzecinkowymi, wynik jest również liczbą stałoprzecinkową (wyjątkiem może być potęgowanie).

PRZYKŁAD 13

Przyjmiemy następujące oznaczenia:

p — liczba wszystkich cyfr wyniku,
q — liczba cyfr ułamkowych wyniku,
p_1 — liczba wszystkich cyfr argumentu pierwszego,
q_1 — liczba cyfr ułamkowych argumentu pierwszego,
p_2 — liczba wszystkich cyfr argumentu drugiego,
q_2 — liczba cyfr ułamkowych argumentu drugiego.

a) dwie zmienne **A** i **B** zostały opisane w następujący sposób:

```
DECLARE A DECIMAL FIXED (6,4),
        B DECIMAL FIXED (5,3);
```

Jakie atrybuty precyzji miała zmienna **X** = **A**+**B**? Atrybuty sumy liczb dziesiętnych stałoprzecinkowych obliczymy według wzoru:

$$p = \{1+(p_1-q_1,\ p_2-q_2)_{max} + (q_1,q_2)_{max},\ 15\}_{min}$$

$$q = (q_1,\ q_2)_{max}.$$

W naszym przykładzie zmienna **X** powinna być opisana w sposób następujący:

```
DECLARE X DECIMAL FIXED (7,4);
```

b) jakie atrybuty będzie miała zmienna AB = A*B?
Skorzystamy ze wzorów:

$$p = p_1 + p_2 + 1,$$

$$q = q_1 + q_2.$$

Zmienna AB powinna być więc zadeklarowana zdaniem:

DECLARE AB DECIMAL FIXED (12,7); ■

W każdym wypadku należy obliczyć atrybut precyzji wyniku opierając się na znajomości atrybutów składników wyrażenia algebraicznego.

Wyrażenia algebraiczne

Przytaczamy kilka przykładów poprawnych wyrażeń algebraicznych.

```
KWADRAT — A*A
B**2      — 4*A*C
(-B + DELTA) / 2*A
-K + (.019/I)
```

Jeśli nawiasy okrągłe nie określają jednoznacznie porządku wykonywania operacji, w pierwszej kolejności wykonuje się operacje jednoargumentowe (operacji jednoargumentowych nie poprzedza żaden identyfikator, np. +A, −A). W następnej kolejności wykonuje się potęgowanie, potem mnożenie i dzielenie (w kolejności od lewej strony do prawej), wreszcie dodawanie i odejmowanie, również w kolejności od lewej strony do prawej. W związku z tym wyrażenie

```
-32.5+0.01*X/ALFA+.1
```

będzie wykonywane w takiej kolejności, jak gdyby było zapisane w następujący sposób:

```
((-32.5)+((0.01*X) /ALFA))+.1
```

Wyrażenie:

```
XYZ(2+BETA)
```

jest błędne. Przed nawiasem powinien być znak operacji.

W przedstawianym tu podzbiorze PL/1 nie dopuszcza się przekształcenia danych zapisanych w postaci tekstów znakowych w dane arytmetyczne i odwrotnie. Można natomiast wykonywać działania arytmetyczne na zmiennych arytmetycznych znakowych, opisanych atrybutem **PICTURE.**

Teksty znakowe i teksty bitowe mogą brać udział wyłącznie w następujących operacjach:

— operacje podstawienia,
— operacje porównania,
— operacje logiczne,
— operacje konkatenacji.

Operacje niearytmetyczne

Operacje podstawienia. Zmiennej tekstowej może być nadana wartość typu tekst znakowy lub bitowy. Operatorem podstawienia jest znak równości. Na miejsce zmiennej wpisuje się podstawiany łańcuch znaków poczynając od lewej strony. Jeżeli długość zmiennej jest większa od długości podstawianego tekstu, wtedy brakującą resztę zapełnia się spacjami. W przeciwnym wypadku, gdy podstawiany tekst nie mieści się w formacie zmiennej, ucina się nadmiar, nie komunikując przy tym o błędzie.

PRZYKŁAD 14

DECLARE KOLOR CHARACTER (11);

Translator zarezerwuje 11 bajtów dla zmiennej **KOLOR**. Po wykonaniu operacji podstawienia:

KOLOR = 'BIALY';

zarezerwowane pole pamięci zapełni się w następujący sposób:

B	I	A	L	Y	␣	␣	␣	␣	␣	␣

Natomiast po realizacji operacji:

KOLOR = 'BIALO_CZERWONY';

pole pamięci będzie zawierało:

B	I	A	L	O	–	C	Z	E	R	W

Znakiem ␣ oznaczyliśmy tu, zgodnie z poprzednio przyjętą umową, spację.

Jeżeli w operacji podstawienia biorą udział dane arytmetyczne o różnych atrybutach, to wówczas argument lub wartość wyrażenia z prawej strony znaku podstawienia przyjmie atrybuty danej zapisanej z lewej strony równości. Przekształcanie atrybutów odbywa się według tych samych zasad, jak przekształcanie argumentów w wyrażeniach arytmetycznych.

Operacje porównania. Operacje porównania określają następujące operatory:

> większy,
>= większy lub równy,
= równy,
¬= nierówny,
<= mniejszy lub równy,
< mniejszy,
¬> nie większy,
¬< nie mniejszy.

Operatory te łączą dwa argumenty. Wynikiem porównania jest zawsze tekst bitowy o długości jednego bitu. Jeśli wyrażenie porównujące jest prawdziwe, wówczas wynikiem operacji jest '1'B, a jeśli jest fałszywe, wtedy wynik ma wartość 'Ø 'B.

Operacje porównania w zależności od argumentów mogą być:

— algebraiczne,
— tekstowe,
— bitowe.

Jeśli porównuje się argumenty różnych typów, to argument mający typ o niższym priorytecie przekształcany jest na typ o wyższym priorytecie. Priorytety typów są następujące (od najwyższego do najniższego):

— dane arytmetyczne,
— dane arytmetyczne znakowe (określone atrybutem **PICTURE**),
— tekst znakowy,
— tekst bitowy.

Jeśli argumenty różnią się atrybutami, następuje przekształcenie identyczne jak przy działaniach arytmetycznych; natomiast jeżeli argumenty są różnej długości, to krótszy dopełnia się z prawej strony spacjami.

Nie dopuszcza się porównań między danymi arytmetycznymi a danymi tekstowymi.

PRZYKŁAD 15

Przykład operacji porównania dwóch zmiennych, opisanych za pomocą następujących zdań:

```
DECLARE NAZWA_1 CHARACTER (8),
        NAZWA_2 CHARACTER (12);
NAZWA_1 = 'KOWALSKI-JAN';
NAZWA_2 = 'KOWALSKI';
IF NAZWA_1 = NAZWA_2 THEN...
```

W tym wypadku wynikiem porównania będzie '**Ø**'B.

Operacje logiczne określają operatory:

¬ negacja (NIE),
& koniunkcja (I),
| alternatywa (LUB).

Przed wykonaniem operacji logicznej każdy argument przekształcany jest w tekst bitowy. Jeśli długości argumentów są różne, krótszy uzupełniany jest zerami z prawej strony.

PRZYKŁAD 16

W programie następującej operacji logicznej:

'**10011**'B&'**101**'B.

W tym wypadku prawy argument uzupełniany jest przed wykonaniem operacji zerami:

'**10011**'B&'**10100**'B.

Wynikiem operacji jest tekst bitowy '**10000**'B. ■

TABLICA 19

Zasady konkatenacji

Pierwszy argument	Drugi argument		
	Tekst znakowy	Tekst bitowy	Dana arytmetyczna
Tekst znakowy	Wynikiem jest tekst znakowy o długości równej sumie znaków obydwu argumentów.	Przed łączeniem przekształca się tekst bitowy w znakowy. Wynik jest tekstem znakowym.	Przed łączeniem przekształca się dane arytmetyczne w tekst znakowy. Wynikiem jest tekst znakowy.
Tekst bitowy	Przed łączeniem przekształca się tekst bitowy w znakowy. Rezultatem jest tekst znakowy.	Wynikiem jest tekst bitowy o długości równej sumie bitów dwóch argumentów do 64 znaków.	Przed łączeniem przekształca się dane arytmetyczne w tekst bitowy. Wynikiem jest tekst bitowy.
Dane arytmetyczne	Przed łączeniem przekształca się dane arytmetyczne w tekst znakowy. Wynikiem jest tekst znakowy.	Przed łączeniem przekształca się dane arytmetyczne w tekst bitowy. Wynikiem jest tekst bitowy.	Operacja niedozwolona.

Operacja konkatenacji. Konkatenacja polega na łączeniu dwóch tekstów lub dopisywaniu do tekstu danej arytmetycznej bez spacji lub jakiegokolwiek innego znaku łączącego. W języku PL/1 znak konkatenacji oznacza się przez dwie pionowe kreski (dwa symbole alternatywy) ||.

Istnieją cztery możliwe sposoby użycia operacji konkatenacji:

1) tekst znakowy || tekst znakowy,
2) tekst bitowy || tekst bitowy,
3) tekst znakowy || tekst bitowy lub tekst bitowy || tekst znakowy,
4) tekst znakowy || dana arytmetyczna lub tekst bitowy || dana arytmetyczna.

Dane arytmetyczne nie mogą być łączone znakiem konkatenacji między sobą. Zasady łączenia przedstawia tablica 19.

PRZYKŁAD 17

```
DECLARE (Z,Y) CHARACTER (10), X CHARACTER (3);
X ='UNI'; Y = 'WERSYTET'; Z = X||Y;
```

W rezultacie wykonania tych operacji zmienna Z będzie miała wartość 'UNIWERSYTE'. ■

Dane sterujące

Dane sterujące mogą być dwóch typów:

— etykiety,
— wskaźniki.

W tym rozdziale omówimy tylko dane typu etykieta. Przetwarzanie z użyciem wskaźników omówimy dalej.

Etykieta jako stała jest nazwą zakończoną dwukropkiem umieszczonym przed zdaniem, od którego ma być kontynuowane wykonywanie programu.

PRZYKŁAD 18

Przykłady etykiet:

```
LICZ: X = (2*A — 4)/PI;
Q25: IF A>B THEN A = 3.1415;
```

Wykonywaniem programu można również sterować za pomocą zmiennych typu etykieta. Zmienne takie należy *deklarować* z atrybutem **LABEL.** Przed wykonaniem programu należy zmiennej typu etykieta nadać wartość. Jedyną operacją, jaką można wykonać ze zmienną tego typu, jest podstawienie stałej typu etykieta.

PRZYKŁAD 19

```
DECLARE CZYTAJ LABEL;
CZYTAJ = E6;
GOTO CZYTAJ;
      .
      .
      .
E6: A = PI*D*D;
```

W tym fragmencie programu skok nastąpi do zdania oznaczonego etykietą **E6**, ponieważ zmienna etykieta **CZYTAJ** otrzymała wartość **E6**.

Wyrażenia skalarne

Algorytmy obliczania wartości zapisuje się w postaci wyrażeń. Wyrażeniem skalarnym są stałe lub zmienne skalarne połączone operatorami dwuargumentowymi. Wartością wyrażenia skalarnego jest wielkość skalarna. Argumentami wyrażenia skalarnego nie mogą być etykiety. Jeśli argumenty wyrażenia skalarnego mają różne atrybuty, wówczas przed wykonaniem operacji dokonywane jest odpowiednie przekształcenie. Istnieją tu pewne ograniczenia, o których wspominaliśmy mówiąc o wyrażeniach arytmetycznych (każde wyrażenie arytmetyczne jest wyrażeniem skalarnym).

Kolejność obliczania wartości wyrażenia zależy od sposobu użycia nawiasów i priorytetów operatorów. Z dwu sąsiednich operacji jako pierwsza jest wykonywana ta, która zawarta jest w większej liczbie nawiasów. Jeśli obie operacje zamknięte są w takiej samej liczbie nawiasów, o kolejności decyduje priorytet operatora. Operatory mają następujące priorytety (operatory zapisane w jednym wierszu mają taki sam priorytet):

```
¬ + (plus) - (minus) **          najwyższy
*/                                priorytet
+ (dodawanie) - (odejmowanie)          |
                                       |
> >= = ¬= <= < ¬> ¬<                   |
&                                      ↓
|                                najniższy
                                   priorytet
```

Tablice

Oprócz zmiennych prostych można posługiwać się zmiennymi indeksowymi. Zmienna indeksowana jest elementem tablicy. Tablicę tworzy zbiór uporządkowanych danych, przedstawionych w postaci macierzy jedno-,

dwu- lub trójwymiarowych. Każdy element tablicy ma te same atrybuty. Zmienną indeksowaną zapisuje się podając nazwę tablicy, a po niej w nawiasach indeksy oddzielone przecinkami, jeśli tablica jest więcej niż jednowymiarowa.

Tablice opisuje się zdaniem **DECLARE** w następujący sposób:

DECLARE *nazwa-tablicy (rozmiar [,rozmiar] [,rozmiar]) [atrybuty] ;*

W zdaniu tym *rozmiar* oznacza ilość elementów każdego wymiaru tablicy, a *atrybuty* są atrybutami każdego elementu.

PRZYKŁAD 20

DECLARE SPIS (1Ø) FLOAT;

W ten sposób zadeklarowano jednowymiarową tablicę o nazwie **SPIS**, składającą się z elementów:

SPIS (1), SPIS (2),..., SPIS(1Ø).

Każdy element ma atrybut **FLOAT DECIMAL(6)**. Zmienną indeksową można w tym wypadku oznaczyć **SPIS(I)**, gdzie **I** jest dowolną zmienną skalarową. ■

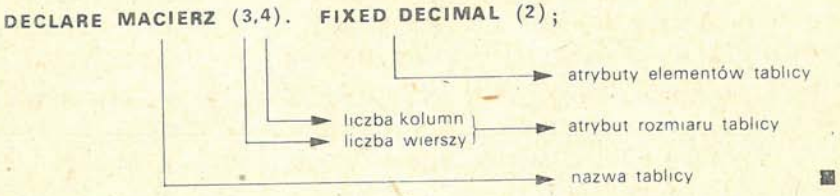
PRZYKŁAD 21

Załóżmy, że w pewnym zadaniu występuje macierz:

$$\text{MACIERZ} = \begin{bmatrix} 1 & 2 & 4 & 2 \\ 3 & 8 & 3 & 8 \\ -6 & 9. & -15 & 0 \end{bmatrix}$$

Taką macierz należy w programie PL/1 opisać następującą deklaracją:

DECLARE MACIERZ (3,4) FIXED DECIMAL (2);



■

Tablica dwuwymiarowa zapamiętywana jest w pamięci wiersz po wierszu. W wypadku tablic trójwymiarowych elementy tablicy rozmieszczane są w ten sposób, aby w pierwszej kolejności zmieniały się indeksy od prawej

strony. Tablica opisana w przykładzie 21 będzie zapamiętana w kolejności: **MACIERZ(1,1), MACIERZ(1,2), MACIERZ(1,3), MACIERZ(1,4), MACIERZ(2,1), ..., MACIERZ(3,3), MACIERZ(3,4).**

Warto zwrócić uwagę na fakt, że translator nie sprawdza, czy użyte przez programistę wskaźniki mieszczą się w deklarowanych granicach.

Każdemu elementowi tablicy można nadać określoną wartość przez podstawienie, np. **MACIERZ(1,2) = 6;** Indeksy mogą być podawane jako liczby całkowite lub jako wyrażenia arytmetyczne. W tym ostatnim wypadku, w celu znalezienia odpowiedniego elementu tablicy na podstawie indeksu, musi być obliczone wyrażenie arytmetyczne. Indeksem jest największa liczba całkowita nie większa od otrzymanego wyniku.

PRZYKŁAD 22

```
K = 8;
L = 0.3;
....
T AB(K*L) = 26;
```

W tym fragmencie programu translator oblicza wartość wyrażenia **K*L = 8*0.3 = 2.4** i jako indeks przyjmuje liczbę 2. W ten sposób drugi element tablicy o nazwie **TAB** otrzyma wartość 26.

Tablica ma nazwę nadaną w instrukcji **DECLARE.** Przez wskazanie nazwy tablicy (bez indeksów) odwołujemy się do wszystkich elementów tablicy jednocześnie. Jeśli natomiast używa się w programie nazwy tablicy z indeksami, wskazuje się w ten sposób jeden element tablicy.

Wyrażenie, w którym co najmniej jeden element jest tablicą, nazywamy *wyrażeniem tablicowym.* Wartością takiego wyrażenia jest również tablica.

PRZYKŁAD 23

Zapis **2*MACIERZ** oznacza, że każdy element tablicy zostanie podwojony (załóżmy, że jest to tablica z przykładu 21). Natomiast wyrażenie **MACIERZ(1,4)+2** oznacza, że do elementu tablicy **MACIERZ(1,4)** zostaje dodana liczba 2. Pierwsze z tych wyrażeń jest wyrażeniem tablicowym, natomiast drugie jest wyrażeniem skalarowym. ■

W stosunku do tablic można używać operacji podstawiania. Operator podstawiania stosuje się w postaci:

$$tablica = \left\{ \begin{matrix} wyrażenie\text{-}tablicowe \\ \\ wyrażenie\text{-}skalarowe \end{matrix} \right\};$$

Tablice wymienione z prawej strony znaku podstawiania muszą być takich samych rozmiarów i mieć takie same graniczne wartości indeksów jak tablica z lewej strony.

PRZYKŁAD 24

Przypuśćmy, że zadeklarowano tablicę;
DECLARE TABLICA(3) FLOAT;
Zdanie:

TABLICA = 9;

jest równoważne zdaniom:

TABLICA(1) = 9;
TABLICA(2) = 9;
TABLICA(3) = 9; ■

PRZYKŁAD 25

Mamy trzy tablice **A, B** i **C.**
DECLARE A(3,2) FLOAT;
B(3,2) FLOAT;
C(3,2) FLOAT;
Zdanie **B = A;** oznacza, że każdemu elementowi tablicy **B** nadaje się odpowiednie wartości z tablicy **A**, tzn.
B(1,1) = A(1,1); B(1,2) = A(1,2) itd.
W wyniku podstawienia **C = −B;** zostaną wszystkim elementom tablicy **C** przypisane wartości elementów tablicy **B** z odwrotnym znakiem. ■

PRZYKŁAD 26

Tablice **A** i **B** składają się z następujących elementów:

$$A = \begin{bmatrix} 3 & 1 \\ 5 & 0 \\ 4 & 2 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 3 \\ 1 & 4 \\ 9 & 5 \end{bmatrix}$$

Po wykonaniu operacji podstawiania:

A = (A+B)2+A(2,1);**

tablica **A** będzie składała się z następujących elementów:

$$A = \begin{bmatrix} 30 & 21 \\ 41 & 57 \\ 210 & 90 \end{bmatrix}$$

Elementy oblicza się w następującej kolejności:
A(1,1), A(1,2), A(2.1) A(2,2), A(3,1), A(3,2) (w toku obliczeń element **A(2,1)** zmienił wartość).

Tablice te można pomnożyć:

C = A*B;

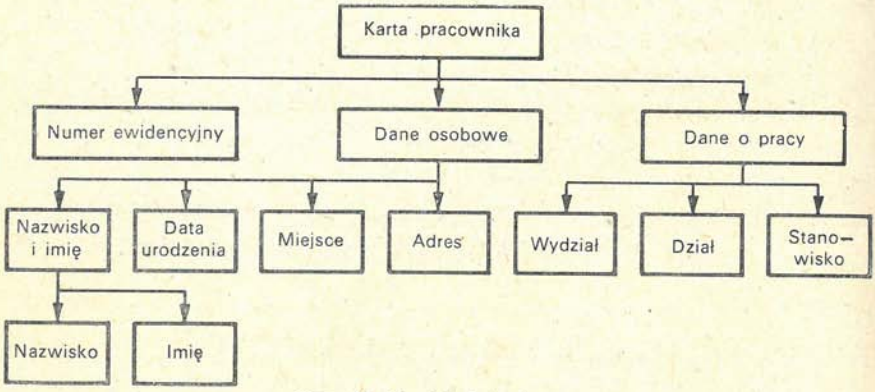
W tym wypadku elementy tablic **A i B**, które mają jednakowe indeksy, będą przemnożone:

C(1,1) = A(1,1)*B(1,1);
C(1,2) = A(1,2)*B(1,2); itd. ■

Operacja A*B nie jest ekwiwalentna mnożeniu macierzy, do jakiego przywykliśmy w algebrze.

Struktury

Struktury można traktować jako rozwinięcie tablic z tym, że każdy z elementów struktury może mieć inne atrybuty.



Rys. 45. Przykład struktury

Zaczniemy od przykładu. Należy zadeklarować zapis zawierający dane o pracowniku przedsiębiorstwa: numer ewidencyjny, dane osobowe, dane o miejscu pracy. Zapis taki może mieć postać hierarchiczną (por. rys. 45). Opis takiej struktury danych jest następujący:

```
DECLARE 1 KARTA_PRACOWNIKA,
          2 NR_EWIDENCYJNY FIXED DECIMAL (6),
          2 DANE_OSOBOWE,
          3 NAZ_IM,
```

```
4 NAZWISKO CHARACTER (2Ø),
4 IMIE CHARACTER (2Ø),
3 DATA_URODZENIA '(6)9',
3 MIEJSCE_URODZENIA CHARACTER (1Ø),
3 ADRES CHARACTER (2Ø),
2 DANE_O_PRACY,
3 WYDZIAL CHARACTER (6),
3 DZIAL,
3 STANOWISKO CHARACTER (5);
```

Reguły deklarowania struktur. Na pierwszym miejscu za słowem kluczowym **DECLARE** następuje zawsze 1, potem spacja i nazwa struktury. Za nazwą struktury występują nazwy podstruktur i elementów podstruktur z odpowiednimi atrybutami. Przed każdym identyfikatorem oznaczającym nazwę podstruktury znajduje się liczba całkowita, która nazywa się numerem poziomu. Za numerem poziomu musi być co najmniej jedna spacja. Elementy struktur lub podstruktury zawarte w strukturze muszą mieć numery poziomu niższe niż numer poziomu starszej struktury, w której są zawarte. W naszym przykładzie starszą strukturą jest **NAZ_IM**, a jest ona jednocześnie podstrukturą struktury **DANE_OSOBOWE**. Elementami podstruktury są zmienne na najniższych poziomach, np. **NAZWISKO, IMIE.**

Ogólna postać zdania **DECLARE** dla struktur jest następująca:

```
DECLARE 1 nazwa , { liczba-całkowita-dodatnia
                  }
    nazwa [atrybut] ••• { [ , liczba-całkowita-dodatnia
                        }
    nazwa [atrybut] ••• ] ••• ;
```

Jeśli *nazwa* jest nazwą podstruktury, to za nią następuje przecinek, jeśli zaś jest elementem, mogą następować za nią atrybuty danych, a za nimi przecinek. Za ostatnim elementem deklaracji następuje średnik.

Numery poziomów nie muszą występować kolejno, mogą być liczbami od 2 do 255. Maksymalna głębokość hierarchiczna struktur wynosi 8. Powtarzające się elementy struktur lub podstruktur można pisać w nawiasach.

PRZYKŁAD 27

```
DECLARE 1 A,
        2 B,
```

3 (C,D,E) CHARACTER(10);

Zapis taki oznacza, że podstruktura **B** składa się z trzech elementów **C, D, E,** każdy z atrybutem **CHARACTER (10).**

Elementami struktur mogą być tablice. Tablice zawarte w strukturze zapamiętywane są w zwykły sposób.

PRZYKŁAD 28

DECLARE 1 AB, 2 C (4,3) DECIMAL FLOAT (5), 2 D (20) FIXED (2); W zdaniu tym zadeklarowano strukturę **AB,** składającą się z dwóch elementów: tablicy **C** i tablicy **D** (zwykle zapisuje się strukturę bardziej przejrzyście, w wielu wierszach, lecz nie jest to konieczne).

Sprecyzowanie nazwy. Przyjmujemy, że dana jest struktura:

```
DECLARE 1 ALFA,
          2 KAPPA,
            3 IMIE (5) CHARACTER (5),
            3 I FIXED DECIMAL (6),
          2 PI CHARACTER (5);
```

Jeśli w programie użyjemy nazwy **ALFA,** będzie to oznaczało, że zwracamy się do wszystkich elementów struktury: **IMIE (1),..., IMIE (5) L, PI.** Przez nazwę **KAPPA** odwołujemy się do sześciu elementów: **IMIE(1),..., IMIE(5), L,** a przez nazwę konkretnego elementu — tylko do tego elementu, na przykład **L.** Jednakże w strukturze mogą wystąpić jednakowe nazwy elementów. Wtedy, chcąc odwołać się do elementu lub podstruktury, stosuje się nazwy złożone. Pokażemy to na przykładzie.

PRZYKŁAD 29

```
DECLARE 1 STRUKTURA,
          2 A,
            3 B FLOAT,
            3 D FLOAT,
          2 E,
            3 D FLOAT,
            3 F FLOAT;
```

W przedstawionej strukturze, chcąc odwołać się do zmiennej **D** należy sprecyzować tę nazwę, gdyż występuje ona dwukrotnie. Sprecyzowanie polega na dodaniu kropki i nazwy starszej podstruktury lub kilku kolejnych podstruktur. W naszym przykładzie mamy **A.D** lub **STRUKTURA.A.D,** a dla drugiej zmiennej **E.D** lub **STRUKTURA.E.D.**

Wyrażenia strukturowe. W wyrażeniach ze strukturami można używać wszystkich operacji dozwolonych w operacjach skalarowych. Wynikiem takiego wyrażenia jest także struktura. Operacje ze strukturami wykonuje się podobnie jak operacje z tablicami. Struktury biorące udział w operacji muszą mieć jednakową budowę, tzn. taki sam układ elementów i rozmiary użytych tablic. Jedynie typy atrybutów mogą być różne, jeżeli dopuszczalne jest odpowiednie przekształcenie.

PRZYKŁAD 30

```
DECLARE 1 STR1,
          2 A,
            3 B1,
            3 B2,
        2 C(2Ø);
DECLARE 1 STR2,
          2 X,
            3 Y1,
            3 Y2,
          2 Z(2Ø);
```

Obie te struktury mają jednakową budowę i mogą być użyte w tym samym wyrażeniu. Wyrażenie **STR1*L**, gdzie **L** jest zmienną skalarną, da w wyniku strukturę o budowie analogicznej **STR1**, z elementami: **B1*L**, **B2*L** i **C*D**.

Wynikiem wyrażenia **STR1*STR2** będzie struktura o elementach: **B1*Y1**, **B2*Y2** i **C*Z**.

Jeśli przekształcenie atrybutów jest konieczne, realizowane jest zgodnie z zasadami przyjętymi dla operacji arytmetycznych.

Operator podstawienia ma dla struktur następującą postać:

struktura = { *wyrażenie-strukturowe* | *wyrażenie-skalarowe* } ;

W ten sposób nadaje się strukturze wartości opisane w prawej stronie równości, w takiej kolejności, jak zadeklarowana jest struktura.

PRZYKŁAD 31

Dla struktur opisanych w ostatnim przykładzie wyrażenie:

```
STR1 = STR2;
```

oznacza, że odpowiednim elementom struktury **STR1** zostaną nadane wartości elementów struktury **STR2.** Zdanie to jest więc równoważne trzem zdaniom:

```
B1 = Y1;
B2 = Y2;
C  = Z;
```

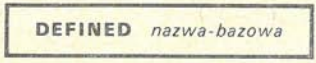
W rezultacie podstawienia:

```
STR1 = A*PI/2;
```

każdy element struktury **STR1** przyjmie wartość obliczoną z wyrażenia **A*PI/2.**

Atrybut DEFINED

Atrybut **DEFINED** występuje w zdaniach **DECLARE** w postaci:



Atrybutem **DEFINED** można zdefiniować nakładanie tekstów na tekst bazowy, określony przez *nazwę-bazową* w zdaniu **DECLARE.** Działanie atrybutu **DEFINED** objaśnimy na przykładzie.

PRZYKŁAD 32

```
DECLARE KARTA CHARACTER (8Ø),
        1 STR1 DEFINED KARTA,
          2 A CHARACTER (65),
          2 B PICTURE '(15)9',
        1 STR2 DEFINED KARTA,
          2 C CHARACTER (3Ø),
          2 D PICTURE '(5)9',
          2 E CHARACTER (45);
```

W przykładzie tym **KARTA** jest nazwą bazową, a **STR1** i **STR2** są nazwami definiowanymi. Przykład ten jest ilustracją typowej sytuacji, gdy wczytujemy dane do pamięci głównej, a następnie zamierzamy używać ich w programie z różnymi nazwami. Karta z danymi zostanie wczytana do pola pamięci o nazwie **KARTA.** Pole to może być traktowane dwojako, dzięki zdefiniowaniu na nim dwóch struktur **STR1** i **STR2.** W pierwszym

wypadku pole **KARTA** zostało podzielone na dwie części o nazwach **A** i **B**, w drugim — na trzy części **C, D** i **E.**

W tablicy 20 przedstawiono zasadę stosowania atrybutu **DEFINED.**

TABLICA 20

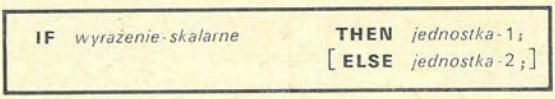
Dopuszczalne sposoby stosowania atrybutu DEFINED

Nazwy definiowane	Nazwy bazowe
Zmienna skalarowa arytmetyczna,	Nieindeksowana zmienna skalarowa arytmetyczna z takimi samymi atrybutami.
Zmienna skalarowa typu etykieta.	Nieindeksowana zmienna skalarowa typu etykieta.
Zmienna skalarowa typu wskaźnik.	Nieindeksowana zmienna skalarowa typu wskaźnik.
Zmienna arytmetyczna znakowa.	Zmienna arytmetyczna znakowa.
Tekst znakowy.	Tekst znakowy.
Tekst znakowy	Struktura z tekstami znakowymi.
Struktura z tekstami znakowymi.	Tekst znakowy.
Tablice z elementami typu zmienne arytmetyczne znakowe lub tekst znakowy.	Tablice z elementami typu zmienne arytmetyczne znakowe lub tekst znakowy.
Struktury z elementami typu zmienna arytmetyczna znakowa, tekst znakowy lub tablica z elementami znakowymi.	Struktury z elementami typu zmienna arytmetyczna znakowa, tekst znakowy lub tablica z elementami znakowymi.
Struktura.	Struktura o identycznej budowie. Elementami struktury mogą być dane arytmetyczne skalarowe, zmienne typu wskaźnik, etykieta lub tekst znakowy.

3. Instrukcje warunkowe i skoki

a. Zdanie IF

Zdanie instrukcji warunkowej **IF** ma następującą postać:



Zdanie to przekazuje sterowanie wykonywaniem programu do jednostki programu *jednostka-1*, jeśli *wyrażenie-skalarne* daje w wyniku wartość '1'B lub wartość, która przekształcona w tekst bitowy, ma chociażby jeden bit równy jedynce; w przeciwnym wypadku wykonywana jest *jednostka-2* lub następne zdanie po zdaniu **IF**, jeżeli części z **ELSE** nie używa się. Jednostką programu *jednostka-1* i *jednostka-2* może być zdanie proste, grupa **DO**, blok, jak również inne zdanie **IF**.

PRZYKŁAD 33

```
IF ROK = 197Ø THEN ROK = 7Ø;
```

Zdanie to spowoduje podstawienie na zmienną **ROK** wartość 70 w wypadku, gdy początkowa wartość tej zmiennej była równa 1970. Dla każdej innej wartości zmiennej **ROK** wartość jej nie ulega zmianie i program będzie kontynuowany od następnego kolejnego zdania. Warto dodać, że w tym przykładzie znak = wystąpił w różnych znaczeniach: pierwszy raz jako operator porównania, drugi raz jako operator podstawienia.

b. Zdanie GOTO

Zdanie **GOTO** jest skokiem bezwarunkowym do miejsca w programie określonego etykietą występującą za słowem kluczowym **GOTO**:

GOTO *etykieta*;

Etykieta występująca w tym zdaniu może być stałą lub zmienną zadeklarowaną z atrybutem **LABEL**. W tym ostatnim wypadku można w zależności od wartości etykiety przekazywać sterowanie do różnych miejsc programu.

Nadmierne stosowanie skoku bezwarunkowego jest szkodliwe: zmniejsza przejrzystość programów i utrudnia ich testowanie. Tak zwane programowanie strukturalne polega m. in. na wyrzeczeniu się instrukcji **GOTO**.

4. Grupy

a. Operator DO jako grupa

Niekiedy powstaje konieczność potraktowania pewnej grupy zdań jako całości. Grupę taką można wydzielić stawiając na początku operator **DO**, a na końcu operator **END;**

PRZYKŁAD 34

```
  IF WARUNEK = Ø THEN GOTO ETYK1;
                 ELSE zdanie-1
     GOTO ETYK2;
ETYK1: zdanie-2
       zdanie-3
       zdanie-4
ETYK2: zdanie-5
       .
       .
       .
```

Ten fragment programu można zapisać za pomocą operatora **DO:**

```
  IF WARUNEK = Ø THEN DO;
     zdanie-2
     zdanie-3
     zdanie-4      END;
               ELSE zdanie-1
     zdanie-5
        .
        .
        .
```

Grupę **DO** tworzy się w programie wtedy, gdy pewien ciąg zdań ma być traktowany jako całość, a więc operator **DO** w parze z **END** jest odpowiednikiem nawiasów w arytmetyce.

b. Operator DO tworzący cykl

Operator **DO** może być użyty do utworzenia cyklu. W takim charakterze operator **DO** może wystąpić w kilku postaciach.

DO WHILE *(wyrażenie)* ;

W tym wypadku operator **DO** powoduje powtórzenie wykonania wszystkich zdań grupy pewną ilość razy, zależną od *wyrażenia* w nawiasach.

Jeśli jest to konieczne, obliczona wartość tego wyrażenia zamieniana jest na tekst bitowy: jeśli chociażby jeden bit tego tekstu ma wartość **'1'B**, to grupa jest wykonywana ponownie; w wypadku przeciwnym następuje przejście do części programu następującej bezpośrednio za daną grupą.

PRZYKŁAD 35

```
X = 1;
DO WHILE (X<1ØØ);
KW = X**2;
SZ = X**3;
X  = X+1;
END;
```

Ten fragment programu jest równoważny sekwencji zdań:

```
  X = 1;
CYKL : KW = X**2;
       SZ = X**3;
       X  = X+1;
       IF X<1ØØ THEN GOTO CYKL;
```

■

Przedstawiając dalsze formy zdania **DO** będziemy używać następujących identyfikatorów:

zmienna — zmienna sterująca operatora **DO**,
wyrażenie-1 — początkowa wartość zmiennej sterującej,
wyrażenie-2 — przyrost (zmiennej sterującej, która po każdorazowym wykonaniu grupy **DO** jest dodawana do zmiennej sterującej),
wyrażenie-3 — końcowa wartość zmiennej sterującej,
wyrażenie-4 — wyrażenie, występujące za słowem **WHILE** jest warunkiem powtórzenia cyklu.

DO *zmienna* = *wyrażenie-1* { **BY** *wyrażenie-2* **TO** *wyrażenie-3* / **TO** *wyrażenie-3* **BY** *wyrażenie-2* } ;

To zdanie DO jest realizowane w następujący sposób: po obliczeniu *wyrażenia*-1 zmienna sterująca przyjmuje wartość początkową, dla której zostaną wykonane wszystkie zdania grupy. Następnie do wartości zmiennej sterującej zostanie dodana wartość *wyrażenia-2* i ponownie wykonane będą zdania grupy. Proces ten powtarza się tak długo, dopóki wartość zmiennej nie przekroczy wartości *wyrażenia-3*.

PRZYKŁAD 36

Wynikiem tej sekwencji zdań będzie macierz jednostkowa o wymiarze N (macierz ta jest tablicą o nazwie M):

```
M = Ø; /* ZEROWANIE ELEMENTÓW MACIERZY */
  DO I = 1 BY 1 TO N;
  M (I,I) = 1;
  END;
```

To samo zdanie **DO** można zapisać w inny sposób:

```
DO I = N BY -1 TO 1;
```

lub

```
DO I = 1 TO N;
```

■

```
DO zmienna = wyrażenie-1 BY wyrażenie-2 WHILE (wyrażenie-4);
```

Grupa zdań będzie wykonywana tak długo, dopóki *wyrażenie-4* będzie dawało jako wynik tekst bitowy, zawierający chociażby jedną jedynkę.

PRZYKŁAD 37

```
DO K = 1Ø BY -2 WHILE(K>1);
        zdanie-1
           .
           .
           .
        zdanie-n
END;
```

W tym wypadku grupa **DO** zostanie wykonana pięć razy, dla K = 10, 8, 6, 4, 2. ■

```
DO zmienna = wyrażenie-1 BY wyrażenie-2
                         TO wyrażenie-3 WHILE (wyrażenie-4);
```

W tej formie operatora **DO** grupa zdań będzie wykonywana tyle razy, dopóki wartość zmiennej sterującej nie osiągnie wartości określonej w *wyrażeniu-3*, lub do momentu, gdy zostanie spełniony warunek zawarty w *wyrażeniu-4*. Fraza **WHILE** działa tak samo, jak w wypadku omówionym wcześniej.

PRZYKŁAD 38

```
DO I = 1 TO N*N+1 BY N WHILE(B>1Ø.5E-8);
A(I) = A(I)+B;
B = B/2;
END;
```

W tym przykładzie zdania zawarte między zdaniem **DO** a zdaniem **END** będą wykonywane cyklicznie. Wyjście z cyklu nastąpi wówczas, jeśli zostanie wypełniony jeden z następujących warunków:

1) **I>N*N+1** lub
2) **B≤1Ø.5E−8.**

DO *zmienna* = *wartość* [, *wartość*]... ;

Wartości, które powinna przyjmować zmienna sterująca, mogą być podane jako lista wartości w operatorze **DO**.

PRZYKŁAD 39

```
DO I = 1, 2, 4, 1Ø, 12, 14;
```

Zdanie to oznacza, że grupa (wszystkie zdania zawarte między zdaniem **DO** a odpowiadającym mu operatorem **END**) zostanie wykonana sześć razy, dla wymienionych wartości **I**.

c. Własności grup i cykli DO

1. Operator **DO** może zawierać tylko wyrażenia skalarowe.

2. Wyrażenia dla początkowych i końcowych wartości oraz przyrostu zmiennej sterującej obliczane są tylko jeden raz i w toku wykonywania operatora DO nie mogą być zmieniane. Możliwa jest natomiast zmiana zmiennej sterującej. Przekroczenie granicznej wartości zmiennej sterującej spowoduje wyjście z cyklu.

3. Możliwe jest wcześniejsze wyjście z cyklu, niż to wynika ze specyfikacji operatora **DO**, za pomocą operatora **GOTO**.

4. Jeśli cykl **DO** zakończył się normalnie, zmienna sterująca ma wartość taką, przy jakiej nastąpiło wyjście z cyklu.

5. Jeżeli cykl został przerwany skokiem **GOTO** lub w wyniku frazy **WHILE**, zmienna sterująca zachowuje wartość taką, jaką miała w danym momencie.

Na przykład po zakończeniu cyklu:

```
DO K = 3 TO 19.25 BY Ø.2;
        .
        .
        .
END;
```

zmienna K będzie równa K = 19.40.

d. Cykle zanurzone

W jednej grupie **DO** można umieszczać inną grupę. W takim wypadku operator **END** grupy wewnętrznej powinien wystąpić wcześniej od operatora **END** grupy zewnętrznej.

PRZYKŁAD 40

```
K = Ø;
DO I = 1 TO 1ØØ;
        .
        .
        .
  DO J = 1 TO 1Ø;
  K = K+1;
        .
        .
        .
  END;
END;
```

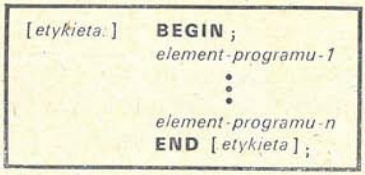
Po wykonaniu tej sekwencji zdań zmienna **K** będzie równa 1000

5. Bloki

Blok jest fragmentem programu, który stanowi pewną funkcjonalną całość. Rozróżnia się dwa rodzaje bloków: blok zwykły (w skrócie będzie my mówili po prostu blok) i blok proceduralny. Blok proceduralny może wystąpić w programie jako procedura lub jako funkcja.

a. Blok zwykły

Blok zwykły ma postać:

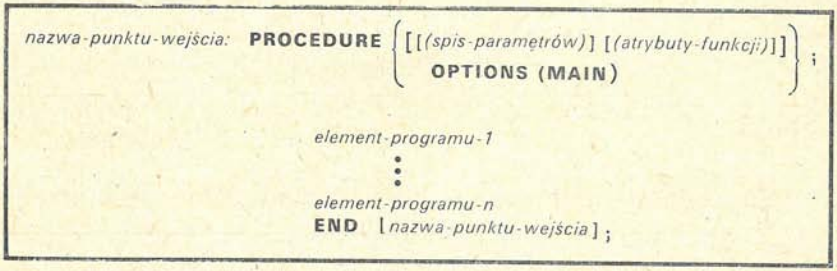


Zdania należące do bloku zawarte są między zdaniami **BEGIN** a **END.** Wszędzie tam, gdzie można zastosować operator grupy **DO,** można w zasadzie użyć bloku **BEGIN.** Zasadnicza różnica między grupą a blokiem polega na tym, że w bloku można deklarować zmienne. Ponieważ blok spełnia pewne dodatkowe funkcje, o których będziemy jeszcze mówić, wszędzie tam, gdzie to jest możliwe należy stosować zdanie **DO** zamiast **BEGIN,** ponieważ przyspiesza to przetwarzanie programu.

Aktywizacja, czyli rozpoczęcie pracy bloku następuje wówczas, gdy w procesie wykonywania programu w zwykłej kolejności wystąpił blok lub gdy instrukcja skoku przekazała sterowanie do zdania **BEGIN.** Ostatnim zdaniem w bloku jest **END,** po wykonaniu którego zostaje wykonane następne zdanie w programie.

b. Blok proceduralny

Blok proceduralny ma następującą postać:



Operatory **PROCEDURE** i **END** ograniczają sekwencję zdań tworząc podprogram. W tym formacie bloku *spis-parametrów* zawiera nazwy zmiennych oddzielone przecinkami, zwane parametrami lub parametrami formalnymi. Przed wykonaniem bloku proceduralnego miejsce parametrów

formalnych zajmą argumenty (lub inaczej parametry aktualne) określane w czasie wywoływania bloku proceduralnego. *Elementy-programu* to zdania, bloki zwykłe lub bloki proceduralne.

Blok proceduralny może być procedurą lub funkcją. Zadaniem procedury jest wykonanie pewnej sekwencji zdań, pewnego podprogramu, który może polegać na obliczeniu wielu różnych wartości. Natomiast zadaniem funkcji jest obliczenie jednej wartości i nadanie jej nazwy funkcji. Nazwa funkcji pokrywa się z *nazwą-punktu-wejścia*. Jeśli blok proceduralny jest funkcją, *atrybuty-funkcji* są atrybutami wartości tej funkcji.

Blok proceduralny może zajmować dowolne miejsce w programie. Gdy występuje w pewnej sekwencji zdań, jest pomijany podczas ich wykonywania. Przed każdym słowem **PROCEDURE** musi występować etykieta, która jest nazwą bloku proceduralnego i jednocześnie pierwotnym punktem wejścia.

Pierwszy blok proceduralny ma postać:

nazwa-punktu-wejścia: **PROCEDURE OPTIONS (MAIN);**

W bloku tym muszą znajdować się wszystkie zdania każdego programu, a więc każdy program w języku PL/1 musi rozpoczynać się tym zdaniem i kończyć zdaniem **END.**

Wywołanie proceduralne. Blok proceduralny może być wykonany wyłącznie przez jego wywołanie. Przez wywołanie określa się punkt wejścia do procedury oraz argumenty, które zostaną przyporządkowane odpowiednim parametrom formalnym. Wywołanie proceduralne może być wywołaniem podprogramowym lub funkcyjnym.

Wywołanie podprogramowe odbywa się za pomocą zdania **CALL,** które ma postać:

CALL *nazwa (spis-srgumentów)*;

W zdaniu tym *nazwa* powinna być *nazwą-punktu-wejścia* wywoływanego bloku. Natomiast *spis-argumentów* tworzą nazwy zmiennych lub wyrażenia, które podczas wywołania procedury aktywizują parametry formalne. Ilość argumentów i ich kolejność musi odpowiadać spisowi parametrów w zdaniu **PROCEDURE.**

Po wywołaniu procedury wykonuje się zawarte w niej zdania, aż do zdania **END,** kończącego procedurę, po czym sterowanie przekazywane jest do następnego zdania po zdaniu **CALL,** które daną procedurę wywołało.

Wykonywanie procedury może zostać zakończone również wskutek zdania **RETURN:**

```
RETURN;
```

Operator **RETURN** powoduje wyjście z bloku proceduralnego i przekazuje sterowanie do następnego zdania po zdaniu wywołującym.

Wywołanie funkcyjne następuje wtedy, gdy w wyrażeniu użyje się nazwy procedury funkcyjnej z podaniem w nawiasach argumentów funkcji. Wówczas na miejsce parametrów wywoływanej funkcji podstawia się argumenty rzeczywiste i sterowanie zostaje przekazane do punktu wejścia tej funkcji. Po wykonaniu obliczeń wartość funkcji przekazywana jest na miejsce wywołania funkcji i program jest kontynuowany. W dalszych obliczeniach wyrażenia bierze już udział wartość funkcji, zastępując jej nazwę.

Wykonywanie funkcji może zostać zakończone tylko w jeden sposób: przez użycie operatora **RETURN**. Zdanie **RETURN** ma w tym wypadku postać:

```
RETURN (wyrażenie);
```

W zdaniu tym *wyrażenie* określa wartość funkcji przesyłaną do punktu wywołania.

Wtórne punkty wejścia. Oprócz podstawowego punktu wejścia do procedury, tzn. przez początek bloku, możliwe są inne punkty wejścia. Muszą być one określone punktem **ENTRY**, który występuje w postaci:

```
nazwa-punktu-wejścia: ENTRY [(spis-parametrów)] [(atrybuty-funkcji)];
```

W zdaniu **ENTRY** *spis-parametrów* i *atrybuty-funkcji* mają takie same wartości w zdaniu **PROCEDURE**.

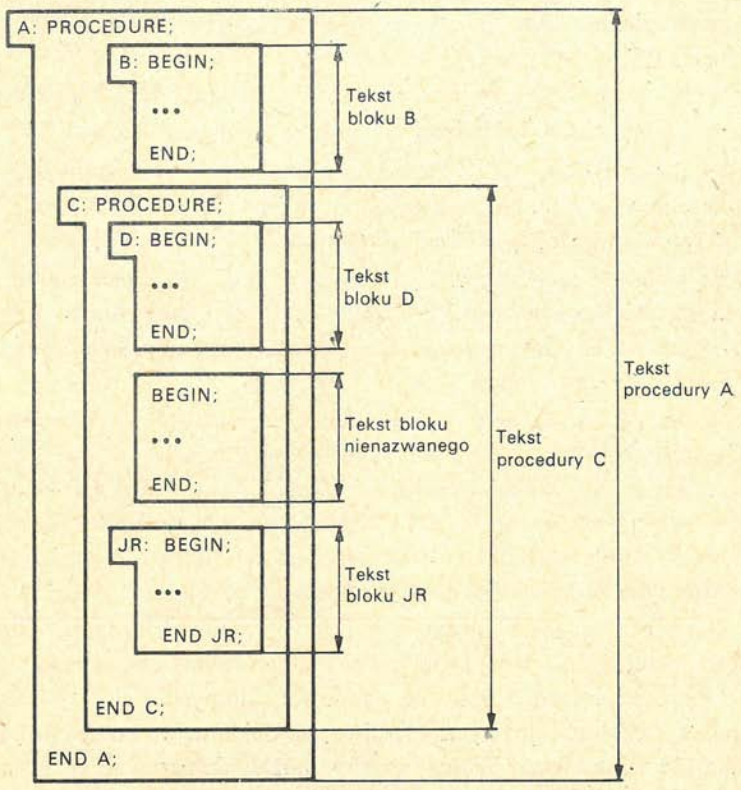
Procedury a bloki. Bloki typu **BEGIN** i **PROCEDURE** spełniają w programie w zasadzie tę samą rolę. Różnice dotyczą dwóch spraw: sposobu przejścia do wykonywania bloku i określenia obiektów, na których blok jest wykonywany.

1. Blok zwykły **BEGIN** wykonywany jest w programie wszędzie tam, gdzie występuje. Blok proceduralny musi być wywołany do wykonania zdaniem **CALL**.

2. Blok **BEGIN** jest wykonywany dla konkretnych wartości zmiennych, dla których go napisano. Nie można podstawiać w miejsce tych zmiennych innych zmiennych. Inaczej jest w blokach proceduralnych: zamiast zmiennych formalnych procedury (parametrów) przy każdym konkretnym wykonaniu można wstawiać inne zmienne (argumenty).

Bloki włożone. W jednym bloku można umieszczać inne bloki. W takim wypadku jeden z bloków jest blokiem zewnętrznym w stosunku do drugiego, wewnętrznego. Blok zwykły musi być zawarty w jakimkolwiek innym bloku.

Dopuszczalna głębokość zanurzenia bloków w rozpatrywanym podzbiorze języka PL/1 wynosi 3. Na rysunku 46 przedstawiono przykład programu składającego się z kilku bloków włożonych.



Rys. 46. Przykład struktury składającej się z wielu bloków

Deklarowanie nazw w blokach. Ponieważ deklarowanie nazw w blokach jest dość skomplikowanym zadaniem, rozpoczniemy od krótkiego podsumowania. Program w PL/1 można rozpatrywać jako ciąg identyfikatorów, stałych i ograniczników. Identyfikatorem jest ciąg liter, cyfr i znaków podkreślenia zaczynający się od litery. Identyfikatory mogą być użyte jako nazwy pewnych obiektów (nazwy zmiennych, nazwy punktów wejścia, nazwy zbiorów danych i tym podobne). Translator wykrywa słowa kluczowe spośród identyfikatorów na podstawie kontekstu. Jeśli identyfikator powinien być użyty jako nazwa określonego obiektu, to trzeba go poprzednio zadeklarować. Typ obiektu określają atrybuty, które mogą być zadane w sposób jawny lub przyjęte przez translator domyślnie.

W jawny sposób nazwa może być zadeklarowana przez:

— zdanie **DECLARE**,

— wystąpienie w charakterze etykiety zdania,

— wystąpienie jako punkt wejścia w zdaniu **PROCEDURE** lub **ENTRY**,

— pojawienie się w spisie parametrów w zdaniu **PROCEDURE**.

Rozpatrzmy nieco dokładniej te sposoby deklaracji nazw.

a. Operator **DECLARE** — w jawny sposób zgłasza identyfikatory jako wewnętrzne nazwy bloku, w którym występuje. Zdanie **DECLARE** może być użyte w dowolnym miejscu programu.

b. Etykieta — identyfikator, oddzielony od początku zdania dwukropkiem, jest jawnie zadeklarowana jako etykieta zdania lub stała typu etykieta. Etykieta jest nazwą wewnętrzną odnośnie bloku, dla którego zdanie zawierające ją jest wewnętrzne.

c. Nazwa punktu wejścia — identyfikator, przed operatorami **PROCEDURE** lub **ENTRY**, oddzielony od słowa kluczowego dwukropkiem. Nazwa może być zadeklarowana w jawny sposób jako nazwa punktu wejścia za pomocą operatora **DECLARE** z atrybutem **ENTRY**. Jeśli rozpatrywany blok jest blokiem wewnętrznym, to nazwy punktów wejścia są wewnętrzne w stosunku do obejmującego go bloku.

d. Parametr. Identyfikatory w spisie parametrów zdania **PROCEDURE** deklaruje się w sposób jawny jako wewnętrzne nazwy danego bloku proceduralnego lub na zasadzie domyślności.

Oprócz deklaracji jawnych możliwe są deklaracje kontekstowe, tzn. wynikające z kontekstu. W ten sposób można deklarować tylko punkty wejścia. Dopuszczalne są dwa wypadki:

— gdy identyfikator następuje zaraz po słowie CALL w zdaniu wywołania procedury,

— gdy identyfikator pojawia się w wyrażeniu przed listą parametrów ujętą w nawiasy (wywołanie funkcyjne).

Obszarem działania nazwy zgłoszonej w ten sposób jest zewnętrzna procedura, w której nazwa ta występuje, z wyjątkiem bloków, w których została zdeklarowana jawnie. Nazwa, która nie została zadeklarowana ani jawnie, ani kontekstowo jest deklarowana domyślnie. Zakresem działania nazwy zadeklarowanej domyślnie jest cała zewnętrzna procedura, w której nazwa ta występuje, z wyjątkiem tych bloków, gdzie została zadeklarowana jawnie.

PRZYKŁAD 42

```
PRO: PROCEDURE;
     DECLARE (K, ALFA, BETA) FIXED BINARY;
     ALFA = Ø;
     BETA = Ø;
        .
        .
        .
     GOTO E1;
        .
        .
        .
     L: GET EDIT (K) (F(5));
        .
        .
        .
     CALL PROC (ALFA);
     E1: ALFA = ALFA+1;
        .
        .
        .
           BL: BEGIN;
               DECLAREK CHARACTER(10);
                  .
                  .
                  .
           CALL PROC(BETA);
           E2: BETA = BETA+1;
```

```
            GOTO L;
            END BL;
              .
              .
              .
      GOTO L;
              .
              .
              .
            PROC: PROCEDURE (I);
                  DECLARE DELTA PICTURE '(5)9';
                    .
                    .
                    .
                  DELTA = I;
                  END PROC;
END PRO;
```

W programie tym użyto następujących nazw:

Nazwa	Typ nazwy	Atrybuty	Obszar działania
PRO	Nazwa punktu wejścia	ENTRY EXTERNAL	PRO
ALFA	Nazwa zmiennej	FIXED BINARY (15)	PRO
BETA	Nazwa zmiennej	FIXED BINARY (15)	PRO
K	Nazwa zmiennej	FIXED BINARY (15)	PRO oprócz BL
L,E1	Stała typu etykieta	LABEL	PRO
BL	Stała typu etykieta	LABEL	PRO
K	Nazwa zmiennej	CHARACTER (10)	BL
E2	Stała typu etykieta	LABEL	BL
PROC	Nazwa punktu wejścia	ENTRY	PRO
I	Parametr	FIXED BINARY (15)	PROC
DELTA	Nazwa zmiennej	PICTURE' (5)9'	PROC

Warto dodać, że nazwa **K** oznacza w tym programie dwie zupełnie różne zmienne, które mają różne obszary działania. Obszarem działania nazwy **E2** jest blok **BL**, a więc przykładowo zdanie **GOTO E2**; umieszczone w bloku **BL** byłoby prawidłowe, natomiast umieszczone na zewnątrz bloku **BL** byłoby błędne.

c. Atrybuty obszaru działania

W przykładzie tym zmienna N w procedurze **P1** i **P2** to dwie różne nazwy, gdyż obszarem działania każdej z tych zmiennych jest tylko blok, w którym zostały zadeklarowane:

```
P1: PROCEDURE;
    DECLARE N FIXED (2);
      .
      .
      .
    END P1;
P2: PROCEDURE;
    DECLARE N FIXED (2);
      .
      .
      .
    END P2;
```

Aby można było traktować nazwę **N** jako nazwę jednej zmiennej, należy rozszerzyć zakres działania zmiennych za pomocą atrybutu **EXTERNAL**. Wszystkie nazwy dotychczas stosowane miały atrybut domyślny **INTERNAL**. Stosując atrybut **EXTERNAL** opisuje się identyfikator jako nazwę zewnętrzną, a więc rozszerza się obszar działania zmiennej. Stąd te dwie procedury należałoby zapisać w ten sposób:

```
P1: PROCEDURE;
    DECLARE N FIXED (2) EXTERNAL;
      .
      .
      .
    END P1;
P2: PROCEDURE;
    DECLARE N FIXED (2) EXTERNAL;
      .
      .
      .
    END P2;
```

Teraz N jest jedną i tą samą zmienną. Pozostałe atrybuty muszą być oczywiście jednakowe.

Atrybuty obszaru działania są przyjmowane domyślnie w sposób:

EXTERNAL — dla nazw punktów wejścia procedury zewnętrznej i nazw kartotek,

INTERNAL — dla nazw punktów wejścia procedury wewnętrznej oraz nazw zmiennych i etykiet.

d. Wykonanie bloku

Aktywizacją bloku nazywamy rozpoczęcie przez translator pewnych czynności pomocniczych bezpośrednio poprzedzających przetwarzanie danego bloku. Zwykły blok aktywizuje się przy kolejnym wykonywaniu zdań programu, gdy sterowanie programu dojdzie do zdania **BEGIN**. Natomiast procedury aktywizuje się przez wywołanie za pomocą operatora **CALL**, a procedury funkcyjne przez użycie nazwy funkcji.

Przy aktywizacji konieczne jest wykonanie takich czynności, jak przydzielenie pamięci, przygotowanie pewnych informacji o stanie programu i tym podobne. W celu wykonania tych czynności translator tworzy dla każdego bloku pewien program, zwany prologiem bloku. Dzieje się to zupełnie bez udziału programisty, ale warto o tym pamiętać i jeśli tylko jest to możliwe unikać stosowania bloków, biorąc pod uwagę stratę czasu na wykonanie prologu.

Po aktywizacji, zdania bloku wykonywane są w zwykły sposób. Aktywny blok może być zakończony w jeden z następujących sposobów:

— przez operator **END**,

— przez operator **RETURN**. Jeżeli wykonywana była funkcja, to pod nazwę jej zostanie podstawiona obliczona wartość,

— przez zdanie **GOTO**, przekazujące sterowanie na zewnątrz bloku.

Po zakończeniu bloku konieczne jest wykonanie również pewnych czynności w rodzaju zwolnienia części pamięci, odtworzenie stanu programu celem jego kontynuacji i tym podobne. Czynności te wykonuje program, zwany epilogiem, co odbywa się podobnie jak w wypadku prologu, również bez udziału programisty.

Przykład 42 ilustruje kolejność wykonywania poszczególnych czynności w programie.

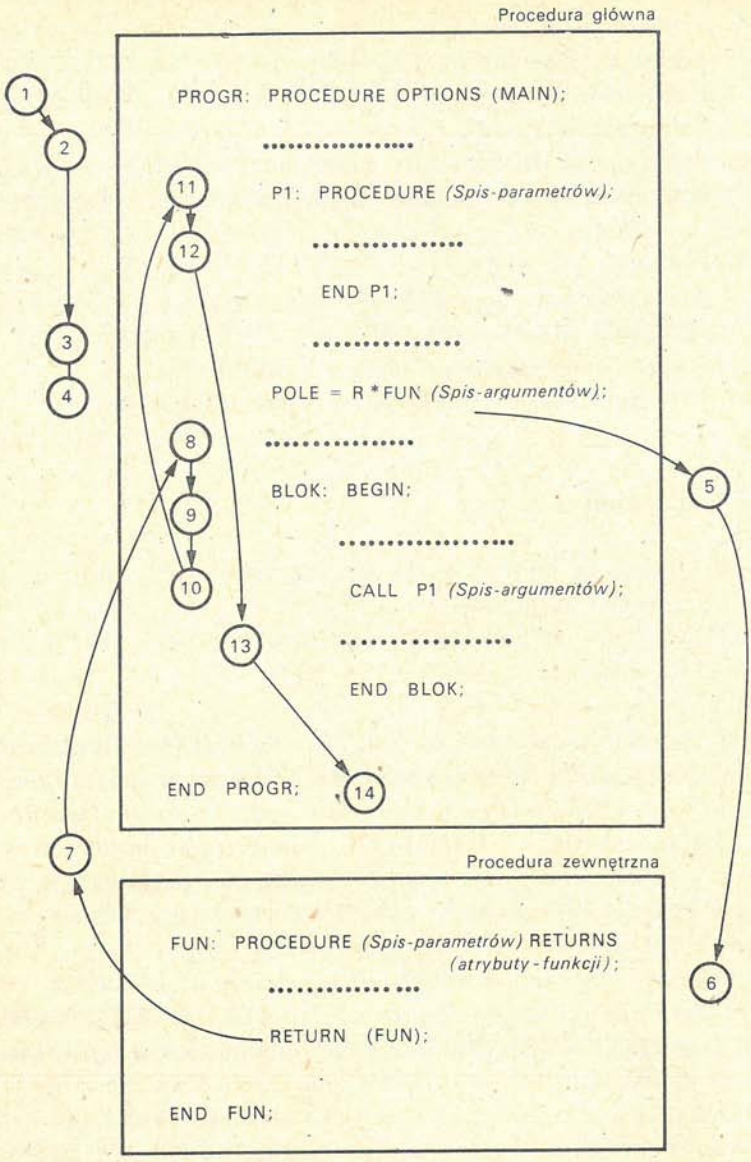
PRZYKŁAD 42

W programie przedstawionym na rysunku 47, poszczególne czynności będą realizowane w następującej kolejności:

1) system operacyjny przekazuje sterowanie procedurze głównej **PROGR**,

2) wykonanie zdań procedury **PROGR**,

3) pominięcie procedury wewnętrznej **P1** i dalsze wykonywanie zdań **PROGR**,



Rys. 47. Program z przykładu 42

4) jedno ze zdań programu **PROGR** zawiera wywołanie funkcyjne, sterowanie zostaje więc przekazane do procedury zewnętrznej,

5) aktywizacja funkcji **FUN**,

6) realizacja zdań funkcji **FUN**,

7) zakończenie wykonywania funkcji **FUN** i powrót z obliczoną wartością do wyrażenia **POLE**,

8) wykonywanie dalszych zdań programu **PROGR**,

9) aktywizacja bloku zwykłego **BLOK** i wykonanie zawartych w nim zdań,

10) wywołanie procedury **P1**,

11) aktywizacja procedury **P1**,

12) wykonanie procedury **P1**,

13) powrót do wykonywania zdań w bloku **BLOK**,

14) zakończenie bloku **BLOK** oraz programu **PROGR**.

6. Klasy pamięci

Wszystkie zmienne opisane w operatorach **DECLARE** należą do jednej z następujących klas pamięci:

1. AUTOMATIC (automatyczna),
2. STATIC (statyczna),
3. BASED (bazowa).

Jeśli zmienna zadeklarowana jest jako **STATIC**, wtedy pamięć dla tej zmiennej wydziela się na początku wykonywania programu i nie zwalnia się jej do zakończenia programu. Natomiast w wypadku, gdy zmienna jest zdeklarowana jako **AUTOMATIC**, pamięć przydzielana jest każdorazowo przy wejściu do procedury i zwalnia się przy wyjściu z niej. Sposób ten jest bardziej praktyczny w wypadku dużej ilości zmiennych w programie, wykorzystywanych tylko epizodycznie.

Zastosowanie atrybutu **BASED** będzie omówione dalej.

Typ klasy pamięci podaje się jako atrybut **AUTOMATIC**, **STATIC** lub **BASED** w zdaniu **DECLARE**. Istnieje ścisły związek między klasami pamięci a atrybutami obszaru działania zmiennych. Zmienne z atrybutami **AUTOMATIC** i **BASED** mogą mieć tylko atrybut **INTERNAL**. Zmienna z atrybutem **STATIC** może być z atrybutem **INTERNAL**, jak również **EXTERNAL**. Można opuścić w zdaniach **DECLARE** słowa **STATIC** lub **AUTOMATIC**. Wówczas rozumie się, że zmienna z atrybutem

EXTERNAL ma atrybut klasy pamięci STATIC, natomiast INTERNAL jest typu AUTOMATIC.
W przykładzie 43 zmienna LICZNIK ma atrybuty STATIC EXTERNAL, dzięki czemu wartość jej pozostaje zachowana po wykonaniu procedur P1 i P2.

PRZYKŁAD 43

W przykładzie tym użyto dla zmiennej LICZNIK atrybutów STATIC oraz EXTERNAL, jest więc to jedna i ta sama zmienna w procedurze głównej i obu procedurach P1 i P2.

```
G_44: PROCEDURE OPTIONS (MAIN);
      DECLARE LICZNIK PICTURE '99' STATIC EXTERNAL;
        .
        .
        .
ET: LICZNIK = 2Ø;
        .
        .
        .
      CALL P1;
        .
        .
        .
      IF LICZNIK 1Ø THEN CALL P2;
                    ELSE GOTO END;
      GOTO ET;
P1: PROCEDURE;
      DECLARE LICZNIK PICTURE '99' STATIC EXTERNAL;
        .
        .
        .
      LICZNIK = LICZNIK-1;
        .
        .
        .
      END P1;
P2: PROCEDURE;
      DECLARE LICZNIK PICTURE '99' STATIC EXTERNAL;
        .
        .
        .
      PUT EDIT (LICZNIK) (F(2));
      END P2;
      END G_44;
END:
```

7. Funkcje standardowe

W języku PL/1 najczęściej używane funkcje opracowano jako funkcje standardowe. Funkcje te znajdują się w bibliotekach systemu operacyjnego i są automatycznie wprowadzane do programu problemowego przez ich wywołanie.

Najważniejsze funkcje standardowe zamieszczono w tablicach 21, 22, 23 i 24.

TABLICA 21

Funkcje matematyczne

Funkcja	Argumenty	Wartość funkcji
ATAN (X)		arctg x (wartość funkcji w radianach)
	$X = 1$	$\pi/2$
ATAND (X)		arctg x (wartość funkcji w stopniach)
	$X = 1$	45°
ATANH (X)	$\|X\| < 1$	arctgh x
	$X = 0$	0
COS (X)	X w radianach	cos x
	$X = 0$	1
COSD (X)	X w stopniach	cos x
	$X = 60$	0.5
COSH (X)		$ch\ x$
	$X = 0$	1
ERF (X)		$\frac{2}{\pi}\int_0^x e^{-t^2}\,dt$
	$X = 1$	0.85
LOG (X)	$X > 0$	$\ln x$
	$X = 2$	0.6931

TABLICA 21 (cd.)

Funkcja	Argumenty	Wartość funkcji
LOG2 (X)	$X>0$	$\log_2 x$
	$X=2$	1
LOG10(X)	$X>0$	lg x
	$X=2$	0.3010
SIN (X)	X w radianach	sin x
	$X=1$	0.8415
SIND (X)	X w stopniach	sin x
	$X=90$	1
SINH (X)		*sh* x
	$X=1$	1.1752
SQRT (X)	$X\geqslant 0$	$\sqrt{x}$
	$X=16$	4
TAN (X)	X w radianach	*tg* x
	$X=1$	1.5574
TAND (X)	X w stopniach	*tg* x
	$X=45$	1
TANH (X)		*th* x
	$X=1$	0.7616

Funkcje matematyczne. Standardowe funkcje matematyczne przedstawia tablica 21. Argumenty tych funkcji powinny być w postaci liczb dziesiętnych zmiennoprzecinkowych. Jeśli tak nie jest, zostaną do takiej postaci doprowadzone. Argumenty mogą być również wyrażeniami skalarowymi lub tablicami. W tym ostatnim wypadku wynikiem funkcji jest tablica o takich samych atrybutach, a funkcja zostanie wykonana odnośnie do każdego elementu tablicy.

PRZYKŁAD 44

Dana jest pięcioelementowa tablica o nazwie **M.**

```
DECLARE M (5) FLOAT,
        A (5) FLOAT;
A = SQRT (M);
```

W wyniku tego zdania otrzymamy tablicę, w której każdy element będzie pierwiastkiem kwadratowym odpowiedniego elementu tablicy M. ■

Pomocnicze funkcje arytmetyczne. Tablica 22 zawiera pomocnicze funkcje arytmetyczne. Argumentami tych funkcji powinny być dane w postaci arytmetycznej, w przeciwnym wypadku przed wywołaniem funkcji będą do takiej postaci doprowadzone.

TABLICA 22

Pomocnicze funkcje arytmetyczne

Funkcja	Argumenty	Wartość funkcji
ABS (X)		|x|
	X = −5	5
BINARY (X[,p[,q]])[a]		Wartość X w postaci dwójkowej w formacie (p, q)
	X = 2 p = 3	010B
CEIL (X)		Najmniejsza liczba całkowita ⩾ X górna granica
	X = 6.03	7
DECIMAL (X[,p[,q]])		Wartość dziesiętna X w formacie (p, q)
	X = 100B, p = 2, q = 1	4.0
FIXED (X [,p [,q]])		Wartość X w postaci stałoprzecinkowej w formacie (p, q)
	X = 2E3, p = 6, q = 2	2000.00
FLOAT (X [,p])		Wartość X w postaci zmiennoprzecinkowej o długości mantysy p
	X = 101B p = 21	Liczba binarna zmiennoprzecinkowa o długości mantysy 21
FLOOR (X)		Największa liczba całkowita ⩽ X (dolna granica)
	X = 2.2	2

TABLICA 22 (cd.)

Funkcja	Argumenty	Wartość funkcji
MAX (X_1, ... X_n)	$n \geqslant 2$	Wartość największego argumentu
	$X_1 = 3$, $X_2 = -1$, $X_3 = 0$	3
MIN (X_1, ..., X_n)	$n \geqslant 2$	Wartość najmniejszego argumentu
	$X_1 = 4$, $X_2 = -1$	-1
MOD (X, Y)		Dodatnia reszta z dzielenia X przez Y
	X = 24, Y = 5	4
PRECISION (X, p[, q])		Wartość X ze wskazaną dokładnością
	X = 6.2, p = 3, q = 2	6.20
ROUND (X, n)		Jeśli X jest liczbą stałoprzecinkową, to X w n-tej pozycji zaokrągla się. Do liczb zmiennoprzecinkowych nie stosuje się
	X = 3.84, n = 1	3.8
TRUNC (X)		CEIL dla X < 0 FLOOR dla X ⩾ 0
	X = −2.4	2

* p, q, l, n — dziesiętne liczby całkowite.

Funkcje przetwarzania tekstów. Funkcje te przedstawia tablica 23. Argumentami funkcji, poza tekstami, mogą być również wyrażenia skalarne i tablice, jeśli w tablicy nie ma na ten temat zastrzeżenia.

Funkcje przetwarzania tablic. Argumentem tych funkcji jest tablica, natomiast wartością jest zawsze wielkość skalarowa.

Funkcje służące do przetwarzania tablic przedstawia tablica 24.

Funkcje dla specjalnych zadań. Pięć funkcji o specjalnym przeznaczeniu przedstawia tablica 25.

TABLICA 23

Funkcje dla przetwarzania tekstów

Funkcja	Argumenty	Wartość funkcji
BIT (X [,*l*])	X — tekst bitowy lub dana arytmetyczna	Wartość X w postaci tekstu bitowego o długości *l*
	X = 5, *l* = 3	'101'B
BOOL (X, Y, *op*)	X, Y — teksty bitowe, krótszy tekst jest wyrównywany zerami z prawej strony do dłuższego *op*-operacja logiczna, może być tekstem bitowym lub znakowym albo wyrażeniem arytmetycznym, które zostanie przekształcone w tekst bitowy o długości 4, a mianowicie 'op_1, op_2, op_3, op_4'	Tekst bitowy o długości dłuższej zmiennej X lub Y, powstały w wyniku zadanej operacji logicznej op: X_i | 0 | 0 | 1 | 1 Y_i | 0 | 1 | 0 | 1 op | op_1 | op_2 | op_3 | op_4
	X = '1100'B Y = '0110'B, op = '0001'B	'0100'B
CHAR (X [,*l*])[a]	X — tekst bitowy, znakowy lub dana arytmetyczna znakowa	Argument zostanie przekształcony w tekst znakowy o długości *l*
	X = '11000001'B	'A'
HIGH (*l*)		Tekst znakowy o długości *l*. Każdy bajt zawiera FF.
	l = 3	FFFFFF
INDEX (X, Y)	X, Y — tekst znakowy, bitowy lub cyfrowy, przekształcony w tekst znakowy	Liczba dwójkowa stałoprzecinkowa 15-bitowa. Wskazuje pozycję, od której tekst Y zaczyna występować w tekście X
	X = 'AAB'	'000000000000010'B
LOW (*l*)		Tekst znakowy o długości l. Każdy znak zawiera 00.
	l = 2	0000
REPEAT (X, *w*)		Tekst otrzymany przez połączenie w-krotne połączenie zmiennej X
	X = 'QS', w = 3	'QSQSQS'

TABLICA 23 (cd.)

Funkcja	Argumenty	Wartość funkcji
SUBSTR (X, *p*, *l*)	X — tekst znakowy, bitowy lub cyfrowy, p-wyrażenie skalarne, które można przekształcić w liczbę całkowitą	Część tekstu X, która zaczyna się z pozycji p i ma długość l.
	X = 'ABCDE', *p* = 2, *l* = 2	'BC'
UNSPEC (X)	X — tekst znakowy lub dana arytmetyczna lub wskaźnik	Tekst bitowy ekwiwalentny wewnętrznemu przedstawieniu argumentu X
	X = −5	'01011101'B

[a] l, w — liczby całkowite dodatnie.

TABLICA 24

Funkcje przetwarzania tablic

Funkcja	Argumenty	Wartość funkcji
ALL(X)	Elementami tablicy X są teksty bitowe; w przeciwnym razie będą w takie przekształcone	Tekst bitowy o długości elementu tablicy X; i-ty bit tekstu równy jest 1, jeśli i-te bity wszystkich elementów X równe są 1, w przeciwnym wypadku równy jest 0.
	X(1) = '101'B X(2) = '001'B	'001'B
ANY(X)	Elementami tablicy X są teksty bitowe; w przeciwnym razie będą w takie przekształcone	Tekst bitowy o długości elementu tablicy; i-ty bit tekst równa się 0, jeśli i-te bity wszystkich elementów równe są 0, w przeciwnym wypadku równy jest 1
	X(1) = '101'B X(2) = '001'B	'101'B
PROD(X)	Tablica X ma elementy typu FLOAT, inaczej zostaną przekształcone w taki typ	Liczba zmiennoprzecinkowa, której wartość jest iloczynem wszystkich elementów tablicy X
	X(1) = 3 X(2) = 1 X(3) = 5	15

TABLICA 24 (cd.)

Funkcja	Argumenty	Wartość funkcji
SUM(X)	Tablica X ma elementy typu FLOAT, inaczej elementy tablicy zostaną przekształcone w taki typ	Liczba zmiennoprzecinkowa, której wartość jest sumą wszystkich elementów tablicy X
	X(1) = 3 X(2) = 1 X(3) = 5	9

8. Operacje wejścia/wyjścia

W rozważanym przez nas podzbiorze języka PL/1 istnieją dwa typy instrukcji we/wy — **GET** i **PUT** oraz **READ** i **WRITE**. Instrukcje **GET** i **PUT** dotyczą przesyłania danych strumieniem, co oznacza, że zbiór danych wejściowych lub wyjściowych traktuje się jako nieprzerwany strumień jednostek danych zapisanych w postaci znakowej. Taki właśnie sposób przetwarzania zbiorów danych będzie przedmiotem tego rozdziału. Przetwarzanie kartotek, w których wyraźnie rozróżnia się zapisy i traktuje je jako jedną całość, omówione zostanie w rozdziale VIII.

a. Operacje wejścia

Przesyłanie danych z urządzenia wejściowego SYSIPT, którym najczęściej jest czytnik kart, do pamięci głównej realizuje się za pomocą zdania **GET**. Postać jego jest następująca:

GET EDIT *(spis-danych)* *(spis-formatów)*;

Słowo **EDIT** oznacza wstępne przetwarzanie danych w toku opera cj wejścia lub wyjścia, np. zmianę formatu liczb.

Spisem danym w zdaniu **GET** mogą być:

— zmienne skalarne,
— zmienne tablicowe,
— zmienne typu struktura.

Spis formatów wskazuje, jak należy interpretować znaki znajdujące się w wejściowym strumieniu. Elementy spisu formatów mogą być następujące :

F(*l*[, *d* [,p]])

Zapis taki oznacza, że *l* znaków powinno być interpretowanych jako liczba stałoprzecinkowa, gdzie

l — wskazuje całkowitą długość liczby,

d — wskazuje pozycję kropki dziesiętnej,

p — współczynnik skali; oznacza, że daną liczbe należy pomnożyć przez 10^p.

Wprowadzane do maszyny liczby powinny mieć następującą postać:

[␣ ...] [+ | −] [*cyfra* ...] [.] [*cyfra* ...] [␣ ...]

PRZYKŁAD 45

Wczytujemy pewne znaki. Wartości, otrzymane w wyniku zastosowania różnych formatów, będą następujące:

Wczytywane znaki	Elementy formatu	Wartość liczby
␣5678	F(5)	5678
␣5678	F(5,1)	567.8
␣5678	F(5,5)	0.05678
␣␣-3.1␣	F(6)	−3.1
␣␣-3.1␣	F(6,2,2)	−310

Jak wynika z tych przykładów, kropka dziesiętna wczytywana razem z liczbą unieważnia położenie kropki zadane w elemencie formatu (znak ␣ oznacza spację).

X(*l*)

Format ten oznacza, że przy wprowadzaniu danych *l* kolejnych znaków należy opuścić.

E(*l*,*d*)

Format typu E opisuje liczby zmiennoprzecinkowe. Liczby zmiennoprzecinkowe wprowadza się w postaci mantysy, za którą następuje litera E, a następnie wykładnik. W podanym formacie *l* oznacza całkowitą liczbę znaków w liczbie, łącznie ze znakiem i literą E, *d* oznacza pozycję kropki dziesiętnej w mantysie. Jeżeli jednak kropka podana jest w sposób jawny, specyfikacja *d* jest pominięta.

Liczby zmiennoprzecinkowe powinny być przygotowane na wejściowym nośniku informacji jako następujący ciąg znaków:

[␣ ...] [+ | −] [*mantysa* ...] E [+ | −] [*cyfra* ...] [␣ ...]

Mantysa jest ciągiem słów, w którym może wystąpić również kropka dziesiętna.

PRZYKŁAD 46

Na karcie wyperforowano liczbę w postaci ⌀⌀-41E+Ø2⌀. W rezultacie zastosowania formatu **E(1Ø̄,3)** do pamięci będzie wprowadzona liczba −4.100. ■

A(*l*)

Format **A** oznacza, że sekwencja *l* znaków ma być rozpatrywana jako tekst znakowy.

B(*l*)

W formacie **B** *l* kolejnych znaków będzie uważanych za tekst bitowy.

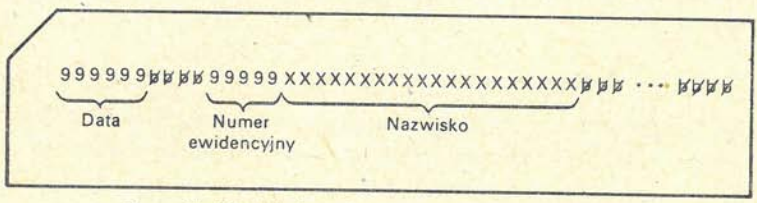
PRZYKŁAD 47

```
DECLARE A PICTURE '(8)9',
        B (4) CHARACTER (5),
        C FIXED (4,1);
GET EDIT (A,B(2), C) (F(8), A(5), F(4,1));
```

W przykładzie tym na zmienną **A** zostanie wczytana liczba całkowita stałoprzecinkowa ośmiocyfrowa, na zmienną indeksowaną **B(2)** zostanie wprowadzony tekst pięcioznakowy i wreszcie na zmienną **C** — liczba stałoprzecinkowa z kropką dziesiętną ustaloną po pierwszej cyfrze z prawej strony. ■

PRZYKŁAD 48

W pewnym systemie przygotowuje się karty perforowane z danymi w następującym układzie (por. rys. 48).



Rys. 48. Układ danych na karcie z przykładu 48

Kartę taką można wczytać używając następujących zdań:

```
DECLARE OBSZAR CHARACTER (8Ø),
        1 POZ DEFINED OBSZAR,
        2 DATA PICTURE '(6)9',
        2 F1 CHARACTER (4),
        2 NR PICTURE '(5)9',
```

```
        2 NAZWISKO CHARACTER (2Ø),
        2 F2 CHARACTER (45);
GET EDIT (OBSZAR) (A(8Ø));
```

Ten sam efekt osiągnąć można również w inny sposób:

```
DECLARE DATA FIXED (6),
        NR FIXED (6),
        NAZWISKO CHARACTER (2Ø);
GET EDIT (DATA,NR,NAZWISKO) (F(6),X(4),F(5),A(2Ø));
```

■

b. Operacje wyjścia

Wyprowadzenie danych z pamięci głównej na drukarkę można zrealizować za pomocą zdania **PUT:**

PUT EDIT *(spis-danych)* *(spis-formatów)*;

Elementami spisu danych mogą być dowolne zmienne skalarowe, tablicowe lub struktury oddzielone przecinkami oraz stałe. W spisie formatów podaje się odpowiednio dla każdej zmiennej postać wydruku. W zdaniu **PUT** w spisie formatów można użyć tych samych formatów co i w zdaniu **GET**, tzn. **F,B,E,A.** Oprócz tego używa się następujących elementów sterujących:

SKIP(*l*), gdzie $l = 0, 1, 2$ lub 3.

Za pomocą elementu **SKIP** można opuścić l wierszy i drukować bieżący wiersz od początku. Przy **SKIP(Ø)** nie następuje przejście do nowego wiersza, co umożliwia wielokrotne drukowanie na tym samym wierszu np. w celu podkreślenia tekstu. **SKIP (1)** spowoduje drukowanie od nowego wiersza. W tym wypadku można pisać po prostu **SKIP.**

PAGE

Element formatu **PAGE** spowoduje drukowanie następnego wiersza od początku nowej strony.

LINE(*l*), gdzie $1 \leqslant l \leqslant 256$.

Element **LINE** stosuje się, aby wskazać wiersz, od którego należy drukować dalszy tekst. Pierwszy wiersz na stronie ma $l = 1$.

COLUMN(*s*), gdzie $1 \leqslant s \leqslant 256$.

Za pomocą elementu **COLUMN** można wskazać pozycję wiersza s, od której powinien być drukowany dalszy tekst. Opuszczone pozycje zostają wypełnione spacjami. W wypadku gdy wskazana pozycja s jest już wy-

prowadzona, proces zapełniania bufora drukarki zostaje zakończony, drukuje się cały wiersz i bufor zaczyna zapełniać się od pozycji *s*.

PRZYKŁAD 49

Nagłówek wydruku można wyprowadzić za pomocą zdania:
PUT EDIT ('PRZYKLADOWY TYTUL') (SKIP,A);
Zdanie to spowoduje wydruk tekstu **PRZYKLADOWY TYTUL** od początku wiersza. Jeśli nagłówek ten miałby znajdować się mniej więcej pośrodku wiersza i rozpoczynać nową stronę, należałoby napisać w programie:

```
PUT PAGE;
PUT EDIT ('PRZYKLADOWY TYTUL') (SKIP, COLUMN(55), A);
PUT EDIT ((120) '—') (SKIP,A);
```

Ostatnie zdanie spowoduje wydrukowanie poziomej linii. ■

PRZYKŁAD 50

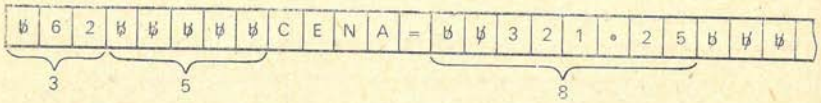
Przypuśćmy, że w pewnym programie wystąpiły zmienne **NR,C,SUMA**, którym nadano wartości:

```
NR = 62;
C = 321.25;
SUMA = -0.032;
```

Zastosujemy różne elementy formatu w zdaniach **PUT** i przedstawimy wydrukowany z ich pomocą wiersz.

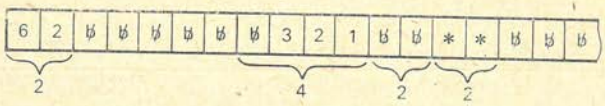
```
PUT EDIT (NR,'CENA =', C) (SKIP,F(3),X(5),A,F,(8,2));
```

W rezultacie tego zdania zostanie wydrukowany wiersz:



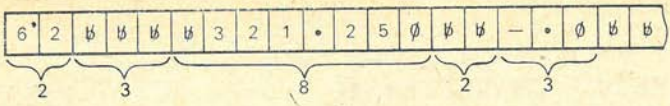

```
PUT EDIT (NR,C) (SKIP,F(2), COLUMN(8),F(4),X(2),F(2));
```

W tym wypadku otrzymamy wiersz:



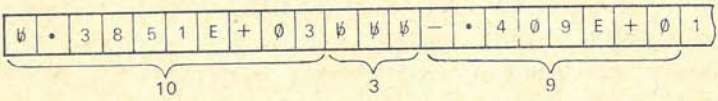
Gwiazdki pojawiają się zamiast liczby w sytuacji, gdy wartość liczby przekracza dopuszczalną wartość wynikającą z podanego formatu.

```
PUT EDIT (NR,C,SUMA) (SKIP,F(2),X(3),F(8,3),X(2),F(3,1));
```



PRZYKŁAD 51

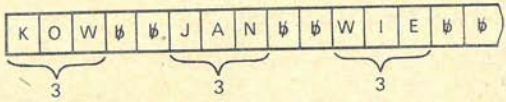
Przyjmijmy, że **A** i **B** liczbami zmiennoprzecinkowymi o wartościach:
A = 385.17,
B = −4.09.
W wyniku zdania **PUT:**
PUT EDIT (A,B) (SKIP,E(1Ø,4),X(3),E(9,3));
zostanie wydrukowany wiersz:



W przykładzie 51 drukowano stałą tekstową. W tym wypadku jednostką formatu jest litera **A**. Podobnie można pisać, gdy długość tekstu, który chcemy wydrukować, jest zgodna z długością w zdaniu **DECLARE**. W ogólnym jednak wypadku element formatu dla tekstu ma postać: **A(1)**, gdzie *l* oznacza długość tekstu.
Jeżeli liczba elementów danych jest mniejsza od ilości elementów w spisie formatów, wówczas zbędne elementy formatów ignoruje się. Natomiast w sytuacji przeciwnej, gdy wyczerpie się cała lista formatów, translator działa cyklicznie i przydziela ponownie te same formaty poczynając od początku spisu formatów.

PRZYKŁAD 52

Przypuśćmy, że w pewnym programie zmienne tekstowe mają wartość:
K = 'KOWALSKI';
J = 'JANKOWSKI';
W = 'WIERZBICKI';
W wyniku zdania:
PUT EDIT (K,J,W) (A(3),X(2));
zostanie wydrukowany tekst:



W następnym przykładzie pokażemy, w jaki sposób można użyć grupy **DO** do wyprowadzenia tablicy.

PRZYKŁAD 53

Załóżmy, że w programie wystąpiły następujące zdania:

```
DECLARE M(3) CHARACTER (4),
        H(5,3) CHARACTER (1);
M = 'ABCD';
H = 'A';
```

Użyjemy zdania PUT z operatorem DO na kilka różnych sposobów:

```
PUT EDIT ((M(I) DO I = 1 TO 3)) (A);
```

W wyniku tego zdania zostanie wydrukowany tekst:

ABCDABCDABCD

```
PUT EDIT ((M(I) DO I = 1 TO 3)) (SKIP,A);
```

Rezultatem będzie wydruk wierszy:

ABCD
ABCD
ABCD

```
DO I = 1 TO 3;
PUT EDIT ((H(I,K) DO K = I TO 5)) (A,X(2));
PUT SKIP;
END;
```

Wynikiem tej sekwencji zdań będzie wydruk:

A␣␣A␣␣A␣␣A␣␣A
A␣␣A␣␣A␣␣A␣␣A
A␣␣A␣␣A␣␣A␣␣A

Chcąc wydrukować strukturę, możemy to uczynić podając w spisie danych nazwę elementów struktury z odpowiednimi elementami formatu. Całą strukturę można wydrukować używając funkcji STRING i atrybutu PACKED, jak to pokazano w przykładzie 54.

PRZYKŁAD 54

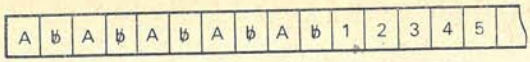
W pewnym programie zadeklarowano strukturę ST:

```
DECLARE 1 ST PACKED,
          2 TAB(5) CHARACTER(2),
          2 TX  PICTURE '(5)9';
```

W wyniku zdań:

```
M = 'A';
TX = 12345; PUT EDIT (STRING (ST)) (A);
```

zostanie wydrukowana cała struktura ST:



c. Zdanie DISPLAY

Krótkie komunikaty lub polecenia dla operatora można wyprowadzać na monitor. Do tego celu służy operator **DISPLAY**, którego używa się w zdaniach następującej postaci:

DISPLAY *(wyrażenie-skalarowe)* **REPLY** *(zmienna-tekstowa)* ;

Wyrażenie skalarowe zostaje zamienione w tekst znakowy o długości do 72 znaków i wyprowadzane jest na monitorze.

Jeśli w zdaniu tym nie występuje słowo **REPLY**, program nie jest przerywany. Atrybut **REPLY** powoduje wstrzymanie pracy programu do czasu, gdy operator maszyny nie napisze na monitorze tekstu (wymaganej odpowiedzi) i przez odpowiedni przycisk zawiadomi o zakończeniu komunikacji z maszyną. Tekst ten zostanie przypisany zmiennej tekstowej, podanej w zdaniu **DISPLAY** po słowie **REPLY**.

PRZYKŁAD 55

W programie tym sterowanie zostanie skierowane do trzech różnych miejsc w zależności od decyzji operatora:

```
PROCES: PROCEDURE OPTIONS (MAIN);
        DECLARE WARIANT CHARACTER (1);
        .
        .
        .
E: DISPLAY ('PODAJ NR WARIANTU') REPLY (WARIANT);
   IF WARIANT = '1' THEN GOTO WAR1;
   IF WARIANT = '2' THEN GOTO WAR2;
   IF WARIANT = '3' THEN GOTO WAR3;
   DISPLAY ('PODALES BLEDNY NR, POWTORZ');
   GOTO E; END;
   .
   .
   .
   WAR1: ...
   WAR2: ...
   WAR3: ...
   END PROCES;
```

■

9. Wskaźniki i zmienne bazowe

W punkcie tym przedstawimy środki języka PL/1 DOS JS, umożliwiające przetwarzanie list. Środkami tymi są wskaźniki i zmienne bazowe.

Wskaźnikiem będziemy nazywać zmienną, której wartością jest adres w pamięci głównej. Wskaźniki należy deklarować w programie jawnie, za pomocą atrybutu **POINTER:**

DECLARE *nazwa-zmiennej* **POINTER;**

Jedyne operacje, w jakich może wziąć udział wskaźnik, to operacje porównania, a więc „=" lub „¬=". Wskaźnikowi można przypisywać jakąś wartość wyłącznie za pomocą funkcji wbudowanych:

ADDR, NULL lub za pomocą pseudozmiennej **UNSPEC.**

TABLICA 25

Funkcje specjalnych zadań

Funkcja	Argumenty	Wartość funkcji
ADDR(X)	X—dowolna zmienna	Wartość skalarowa, która jest adresem zmiennej X w pamięci
	X = LICZ	Wartość wskaźnika określającego miejsce zmiennej LICZ w pamięci
DATE		Tekst znakowy oznaczający aktualną datę w postaci: rrmmdd, gdzie rr—rok, mm—miesiąc, dd—dzień
NULL		Wartość wskaźnika równa zeru
STRING	X—struktura z atrybutem PACKED, zawierająca tylko tekst znakowy lub cyfrowy	Tekst znakowy otrzymany przez konkatenację wszystkich elementów struktury
TIME		Tekst znakowy, oznaczający czas dnia w postaci hhmmssttt, gdzie hh—godziny, mm—minuty, ss—sekundy, ttt—milisekundy

Wbudowana funkcja **ADDR(x)** działa w ten sposób, że wartością jej jest adres zmiennej występującej jako argument funkcji. Argumentem funkcji może być dowolna zmienna skalarna, zmienna indeksowana, elementy struktury, tablica lub struktura.

Funkcja **NULL** nadaje wskaźnikom wartość zerową. Nie jest to konkretny adres pamięci, lecz sposób zainicjowania wskaźnika. Tę wartość wskaźnika można wykorzystać dla porównania z innymi wartościami.

PRZYKŁAD 56

Zmienną **WS** opisano jako wskaźnik:

```
DECLARE WS POINTER,
        DOW PICTURE '999';
```

Zdanie

```
WS = ADDR (DOW);
```

spowoduje, że wskaźnik **WS** będzie równy adresowi zmiennej **DOW**. Natomiast zdanie:

```
IF WS = NULL THEN GOTO ET;
```

może spowodować przekazanie sterowania do etykiety ET, jeżeli wskaźnik ma wartość zerową. ■

Wskaźników używa się wraz ze zmiennymi bazowymi. Zmienną bazową deklaruje się w następujący sposób:

DECLARE *zmienna-bazowa* **BASED** *(wskaźnik)*;

Zmienna bazowa nie może być zmienną typu wskaźnik. Przed użyciem zmiennej bazowej w programie konieczne jest określenie związanego z nią wskaźnika. Jeżeli zmienna bazowa jest w instrukcji, wówczas odwołuje się ona do adresu pamięci, określonego przez związany z nią wskaźnik.

PRZYKŁAD 57

```
DECLARE (A,B) CHARACTER (80),
        P POINTER,
        C CHARACTER (80) BASED (P);
P = ADDR (A); /* WSKAŹNIK P OTRZYMA WARTOŚĆ RÓWNĄ ADRESOWI ZMIENNEJ A*/
              /* ZMIENNA C UMIESZCZONA BĘDZIE W OBSZARZE PAMIĘCI ZMIENNE, A */
```

Zmiennych bazowych można używać m.in. do przetwarzania zapisów różnych typów, modelowania tablic, których elementy są strukturami, dynamicznego przydzielania pamięci, przetwarzania danych w buforze.

10. Sytuacje wyjątkowe

W toku wykonywania programu może wystąpić wyjątkowa sytuacja, zaaranżowana przez programistę lub spowodowana błędem. Może to być, dla przykładu, osiągnięcie końca czytanej kartoteki na taśmie magnetycznej czy też błąd polegający na dzieleniu przez zero.

Pojawiające się sytuacje wyjątkowe powodują z reguły przerwanie programu. W języku PL/1 istnieją środki umożliwiające w takich wypadkach interwencję programisty, który może sam zaproponować dalsze postępowanie albo zdać się na standardową reakcję systemu.

a. Zdanie ON

Za pomocą zdania **ON** można włączyć *sytuację*, to jest „uczulić" program na pewne wyjątkowe warunki. W wypadku powstania takiej sytuacji, zdanie **ON** wskazuje działanie w powstałych warunkach. Działaniem tym może być przekazanie sterowania programem do pewnego własnego podprogramu lub ograniczenie się do standardowej reakcji systemu.

Sytuacje **ON** bywają dwóch typów: sytuacje obliczeniowe, wynikające w procesie obliczeń i podstawiania wartości, oraz sytuacje wejścia/wyjścia. Sytuacjami obliczeniowymi są:

CONVERSION	— nieprawidłowe przekształcenie danych,
FIXEDOVERFLOW	— nadmiar stałoprzecinkowy,
OVERFLOW	— nadmiar zmiennoprzecinkowy,
UNDERFLOW	— utrata wartości liczby zmiennoprzecinkowej,
ZERODIVIDE	— dzielenie przez zero,
SIZE	— utrata znaczących cyfr.

Sytuacje obliczeniowe przedstawione są szczegółowo w tablicy 26. Sposób użycia zdania **ON** pokażemy na przykładzie.

TABLICA 26

Sytuacje obliczeniowe

Sytuacja	Przyczyna	Działanie standardowe systemu	Specyfikacja działania
CONVERSION	Próba wykonania niedozwolonego przekształcenia danych tekstowych	Sytuacja ERROR i przerwanie programu	Operator GOTO; dana zachowuje swoją poprzednią wartość
FIXEDOVERFLOW	Utrata znaczących miejsc w liczbach stałoprzecinkowych: wynik operacji arytmetycznych przewyższa maksimum 15 cyfr dziesiętnych lub 31 binarnych. Nie występuje w GET.	jw.	1–operator GOTO; dana zachowuje swoją poprzednią wartość. 2–operator pusty; kontynuowanie obliczeń z otrzymaną wartością danej, mającą odcięte znaczące pozycje z lewej strony
OVERFLOW	Nadmiar liczby zmiennoprzecinkowej (maksymalna wartość $7.237 \cdot 10^{75}$)	jw.	1–operator GOTO; dana zachowuje swoją poprzednią wartość. 2–pusty operator; kontynuacja programu z nieokreśloną wartością danej
SIZE	W operacji podstawienia dana nie mieści się w wyznaczonym polu	jw.	1–operator GOTO; dana zachowuje swoją poprzednią wartość. 2–operator pusty; traci się nadmiarowe cyfry. W zdaniu PUT przy formacie F wszystkie pozycje zapełniane są gwiazdkami.
UNDERFLOW	Liczba zmiennoprzecinkowa otrzymuje wartość mniejszą od minimalnej ($5.4 \cdot 10^{-78}$)	Kontynuacja programu z daną o wartości zero	1–operator GOTO; dana zachowuje swoją poprzednią wartość. 2–operator pusty; standardowe działanie systemu

TABLICA 26 (cd.)

Sytuacja	Przyczyna	Działanie standardowe systemu	Specyfikacja działania
ZERODIVIDE	Próba dzielenia przez zero danych stało- i zmienno-przecinkowych	Sytuacja ERROR i przerwanie programu	1–operator GOTO; dana zachowuje swoją poprzednią wartość. 2–operator pusty; przy dzieleniu zmiennoprzecinkowym lub binarnym stałoprzecinkowym wynik dzielenia przyjmuje wartość dzielnej, przy dzieleniu dziesiętnym wynik nieokreślony

PRZYKŁAD 58

```
PG_58: PROCEDURE OPTIONS (MAIN);
       DECLARE (A,B) FIXED (4);
       ON ZERODIVIDE GOTO E2;
E1:    GET EDIT (A,B) (F(4));
       PUT EDIT (A/B) (SKIP,E(1Ø,5));
       GOTO E1;
E2:    PUT EDIT ('DZIELENIE PRZEZ ZERO') (SKIP,A);
       END;
```

Program ten wczytuje w pętli liczby **A** i **B** oraz drukuje ich iloraz. Powtarza się to tak długo, dopóki **B** jest różne od zera. W wypadku, gdy $B = 0$, próba dzielenia przez zero spowoduje, dzięki zdaniu **ON**, przekazanie sterowania do zdania z etykietą E2 i zostanie wydrukowany tekst 'DZIELENIE PRZEZ ZERO'.

W takim samym programie bez zdania **ON** w wypadku próby dzielenia przez zero nastąpi standardowe działanie systemu, a mianowicie przerwanie programowe i zakończenie programu. ■

Sytuacje pojawiające się w czasie wykonywania operacji wejścia i wyjścia są przedstawione w rozdziale VIII.

Oprócz wymienionych istnieje sytuacja **ERROR**, którą należy traktować jako pewną zbiorczą sytuację wszystkich stanów wyjątkowych systemu, których nie można rozpoznać. Terminu „sytuacja ERROR" używa się również na określenie ogólnego stanu powstałego w wyniku wystą-

pienia dowolnej sytuacji przewidzianej w PL/1. Sytuacja ta jest częścią standardowego działania systemu.

Przerwanie programowe z powodu wyjątkowej sytuacji może nastąpić tylko wtedy, gdy sytuacja jest włączona. Wszystkie sytuacje oprócz **SIZE** są włączane automatycznie, wraz z rozpoczęciem wykonywania głównej procedury. Niezależnie od tego, sytuację obliczeniową można włączyć lub wyłączyć za pomocą przedrostka sytuacyjnego. Przedrostek sytuacyjny pisze się w nawiasach na początku zdania, po których następuje dwukropek, np.:

(NOZERODIVIDE): ETYK: IF A = B THEN GOTO DR;

Można używać następujących przedrostków sytuacyjnych odpowiednio włączających i wyłączających:

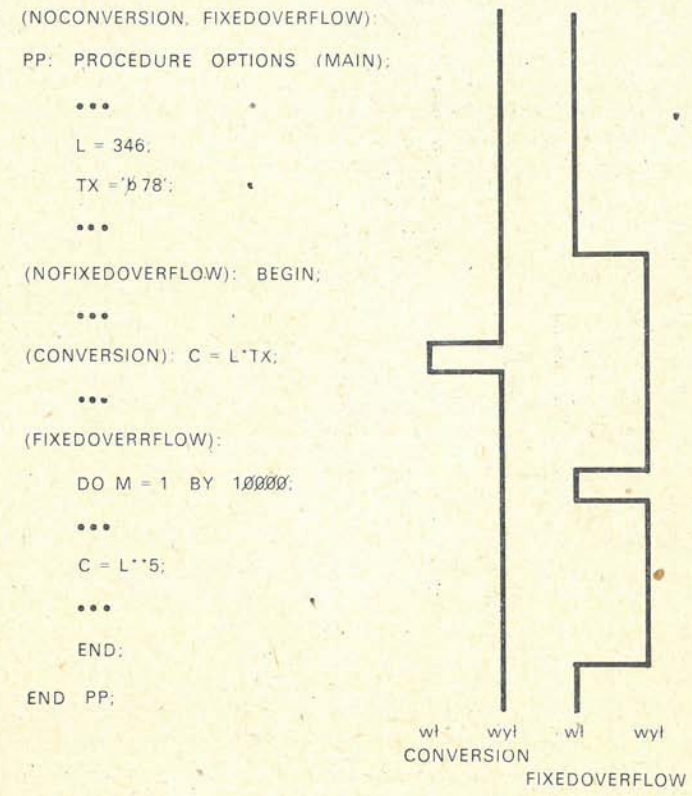
CONVERSION	**NOCONVERSION**
FIXEDOVERFLOW	**NOFIXEDOVERFLOW**
OVERFLOW	**NOOVERFLOW**
UNDERFLOW	**NOUNDERFLOW**
ZERODIVIDE	**NOZERODIVIDE**
SIZE	**NOSIZE**

W jednym przedrostku sytuacyjnym dozwolone jest użycie kilku sytuacji oddzielonych przecinkami. Przedrostki sytuacyjne można stawiać w zasadzie przed wszystkimi zdaniami oprócz **DECLARE** i **ENTRY**, lecz celowe jest ich użycie przede wszystkim przed zdaniami: **PROCEDURE**, **BEGIN**, cykl **DO**, **IF**, operator podstawienia, **GET**, **PUT**, **CALL**, **RETURN**. W czterech pierwszych wypadkach obszarem działania jest odpowiednio cały blok **PROCEDURE**, blok **BEGIN** lub wyrażenia następujące bezpośrednio po **DO** i **IF**. Oczywiście inny przedrostek wewnątrz bloku może zmienić sytuację. W pozostałych wypadkach przyrostek sytuacyjny dotyczy tylko zdań, przed którymi stoi.

Na rysunku 49 przedstawiono program, na którym pokazano zakresy działania sytuacji **CONVERSION** i **FIXEDOVERFLOW**. W programie tym sytuacja **CONVERSION** jest wyłączona w całej procedurze głównej z wyjątkiem jednego zdania: **C = L* TX;** Ponieważ przy wykonywaniu tego zdania wystąpi sytuacja **CONVERSION** z powodu próby niedopuszczalnego przekształcenia wewnętrznego danej znakowej w daną arytmetyczną, działaniem standardowym systemu będzie przerwanie programu. Natomiast, jeśli sytuacja taka wystąpi w pozostałych zdaniach programu,

dane zachowają swoje wartości, a program będzie kontynuowany od następnego zdania za zdaniem, które wywołało sytuację.

Sytuacja **FIXEDOVERFLOW** jest włączona w głównej procedurze **PP** z wyjątkiem bloku **BEGIN**. Z kolei wyjątkiem w bloku **BEGIN** jest zdanie **DO**, gdzie sytuacja **FIXEDOVERFLOW** jest znów włączona.



Rys. 49. Przykładowy program obrazujący obszary działania przedrostków sytuacyjnych

Z tego więc powodu, jeśli zmienna **M** w cyklu **DO** będzie bez przeszkód wzrastać, wówczas przy przekroczeniu maksymalnej wartości, jaką może przyjąć zmienna stałoprzecinkowa, to jest $2^{+31}-1$, nastąpi przerwanie programowe.

Za pomocą zdania **ON** można ustalić działanie różniące się od stan-

dardowego przerwania programu i w określony sposób kontynuować program, jak to pokazano w przykładzie 58. Zdanie **ON** może być w programie umieszczone w dowolnym miejscu, z tym jednakże zastrzeżeniem, iż działaniem swym nie obejmuje zdań poprzedzających je. Ogólna postać zdania **ON** jest następująca:

ON *sytuacja specyfikacja-działania* ·

Sytuacją może być jedna z sytuacji obliczeniowych, wejścia/wyjścia lub **ERROR**. Z reguły jako specyfikacja działania występuje operator **GOTO**, wskazujący dalsze działanie w wypadku pojawienia się wymienionej w zdaniu sytuacji. We wszystkich sytuacjach oprócz **CONVERSION**, **ENDFILE** i **KEY** specyfikacja działania może być pominięta, inaczej mówiąc, może wystąpić operator pusty. W takim wypadku nastąpi przerwanie i sterowanie zostanie przekazane do miejsca, które wywołało sytuację.

Jeśli sytuacja jest wyłączona, program jest kontynuowany, przy czym zdanie, które spowodowało daną sytuację wyjątkową, jest dokończone jako puste.

b. Zdanie SIGNAL

Za pomocą zdania **SIGNAL** można symulować pojawienie się sytuacji wyjątkowej. Jest to szczególnie przydatne w czasie testowania programu. Zdanie **SIGNAL** pisze się w następujący sposób:

SIGNAL *sytuacja* ;

W szczególnym wypadku zdanie:

SIGNAL ERROR;

spowoduje przerwanie programu i takie działanie, jak gdyby wystąpiło zdanie **SIGNAL** z sytuacją aktualną dla danego miejsca programu.

Wyjątkiem jest zdanie:

SIGNAL UNDERFLOW;

które nie powoduje przerwania programu. Drugim wyjątkiem jest zdanie:

SIGNAL ENDPADE *identyfikator*;

gdzie identyfikator jest nazwą kartoteki z atrybutem **PRINT**. W tym wypadku nastąpi tylko przejście do nowej strony i program będzie kontynuowany.

VIII. Organizacja i przetwarzanie kartotek

W systemach elektronicznego przetwarzania danych mamy najczęściej do czynienia z danymi zorganizowanymi w pewne większe zbiory, zwane *kartotekami*. Kartoteka jest grupą danych dotyczących określonego problemu i traktowana jest w procesie przetwarzania danych jako wyodrębniona całość. Kartoteka składa się z *zapisów* (ang. *record*). Zapis zdefiniujemy jako grupę znaków lub słów w pamięci komputera lub na nośniku danych, odpowiadającą najczęściej zawartości jednego dokumentu źródłowego[1]. Problem organizacji kartotek i ich przetwarzania jest centralnym zagadnieniem w przetwarzaniu danych. W Jednolitym Systemie EMC istnieje bogaty repertuar środków służących do tego celu.

1. Organizacja kartotek

Każdy zapis, z logicznego punktu widzenia, składa się z pozycji. Pozycja jest najmniejszą jednostką logiczną, której można przypisywać wartości i jaką można identyfikować za pomocą nazw. Pewne pozycje w zapisie mają szczególne znaczenie, gdyż służą do identyfikacji tego zapisu. Będziemy nazywali je kluczami.

Z punktu widzenia formalnego (fizycznego), zapis jest pewnym blokiem danych, reprezentowanych w pamięci przez bity, bajty, słowa. Dane łączy się w bloki celem zwiększenia efektywności transmisji danych.

Kartotekę tworzą zapisy zorganizowane w pewien logiczny sposób, co umożliwia odpowiednie ich przetwarzanie. System operacyjny DOS JS zapewnia programiście trzy standardowe rodzaje organizacji kartotek[2]: sekwencyjną, indeksowo-sekwencyjną i losową. Wybór jednej z tych me-

[1] Definicje zgodne z Polską Normą 71/T-01016.

[2] Por. *Sistiema IBM/360. Wwiedienije w zapominajuszczije ustrojstwa priamogo dostupa i mietody organizacyi dannych*, Statistika, Moskwa 1974; DOS/JS *Organizacja wejście-wyjście*.

tod zależny jest od charakteru przetwarzania (częstotliwości wprowadzania zmian, zakresu wykorzystania kartoteki, rozmiaru kartoteki, wymaganego czasu dostępu itd.), jak również od rodzaju posiadanych urządzeń zewnętrznych.

Przed omówieniem tych podstawowych struktur kartotek, rozpatrzymy sposoby przedstawiania zapisów.

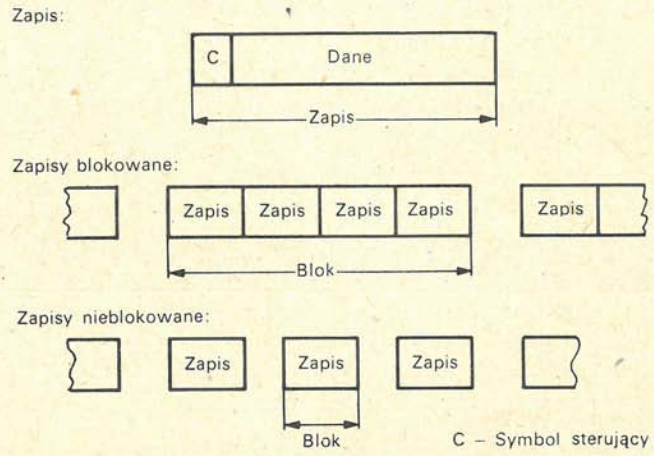
a. Formaty zapisów

Każdy zapis ma jeden z trzech formatów:
F — zapis stałej długości,
V — zapis zmiennej długości,
U — zapis nieokreślonej długości.

Wybór formatu zapisu zależeć będzie od samej natury kartoteki, a także od typu stosowanych urządzeń zewnętrznych.

Zapisy stałej długości **(F)**. Zapisy formatu **F** mają niezmienną długość w całej kartotece. Zapisy te mogą być blokowane lub nieblokowane. Każdy blok składa się z dowolnej, lecz stałej liczby zapisów (być może z wyjątkiem ostatniego bloku, który może być niepełny). Przy formacie **F** nieblokowanym każdy blok składa się z jednego zapisu.

Zapisy stałej długości przedstawia rysunek 50. Znak sterujący służy do sterowania wydrukiem lub wyborem magazynka w perforatorze kart. Nie jest obowiązkowy.

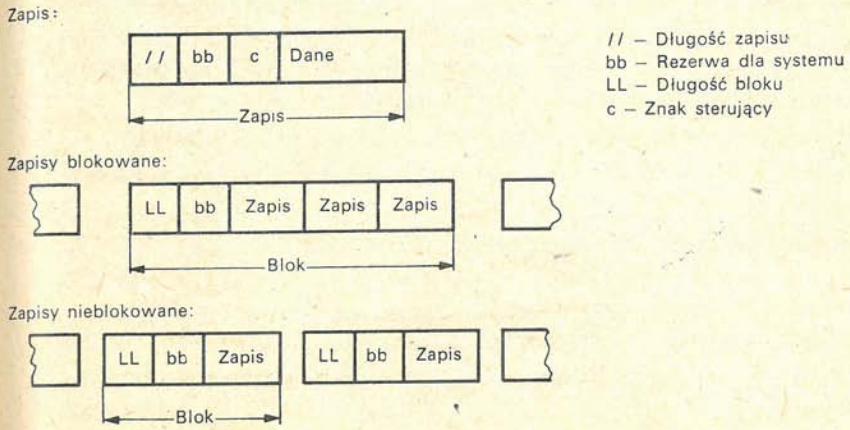


Rys. 50. Zapis formatu F

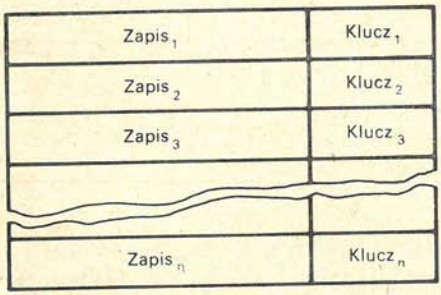
Zapisy zmiennej długości **V**. W formacie **V** zapisy mają różne długości, w związku z czym, w każdym z nich musi być podana jego długość. Jeśli zapisy te są blokowane, wtedy każdy blok musi być poprzedzony daną określającą długość bloku. Rysunek 51 przedstawia zapisy formatu **V**.

Każdy więc zapis poprzedzony jest dodatkowymi znakami. Może im towarzyszyć również symbol sterujący. Znaki te nie są jednak perforowane lub drukowane.

Zapisy nieokreślonej długości **U**. Jeśli z góry nie można przewidzieć długości zapisu, wówczas nie można użyć zarówno formatu **F**, jak i **V**. W takim wypadku korzysta się z formatu **U**. Każdy zapis jest traktowany wówczas jako oddzielny blok. Dozwolone jest stosowanie znaku sterującego.



Rys. 51. Zapisy formatu V



Rys. 52. Kartoteka sekwencyjna

b. Kartoteki sekwencyjne

Jest to najbardziej rozpowszechniona metoda organizacji zbiorów danych, wzorowana na organizacji kartotek tradycyjnych, ręcznych. Zapisy wprowadzane są na nośnik informacji w kolejności ich napływania (sekwencyjnie) i w takiej samej kolejności mogą być odczytane. Sekwencję, czyli uporządkowanie zapisów w zbiorze określają klucze, które mogą w jawnej postaci nie występować, lecz są najczęściej elementami zapisu. Dla przykładu, zapisując na taśmę magnetyczną kartotekę osobową pracowników, otrzymamy zbiór sekwencyjny, którego kluczem może być numer kolejny pracownika.

Zaletą struktury sekwencyjnej jest prostota i oszczędność pamięci, ponieważ organizacja taka nie wymaga żadnych dodatkowych wskaźników. Utrudnione jest natomiast wyszukiwanie zapisów, ponieważ należy przy tym przeglądać zapisy poprzedzające szukaną pozycję, a w najbardziej niesprzyjającym wypadku nawet cały zbiór.

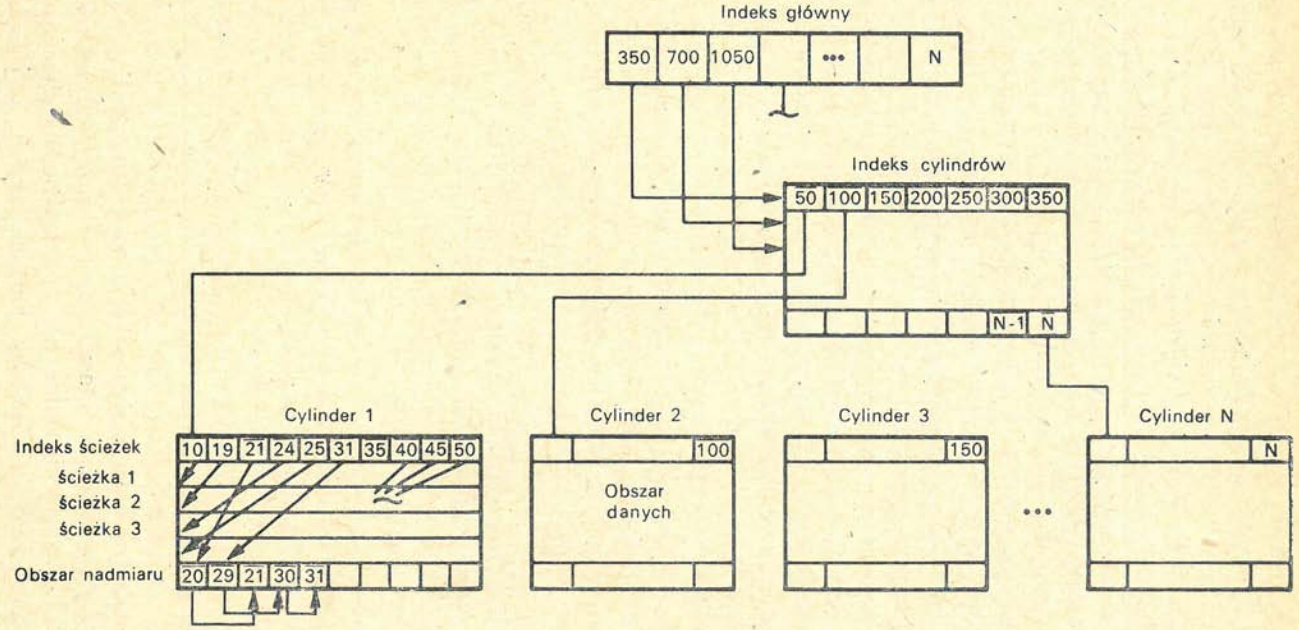
Stosowanie kartotek sekwencyjnych jest korzystne wtedy, gdy w procesie przetwarzania udział w przetwarzaniu bierze większość zapisów kartoteki.

c. Kartoteki indeksowo-sekwencyjne

W zbiorach indeksowo-sekwencyjnych zapisy uporządkowane są według wzrastającej wartości kluczy. Klucze są polami o długości od 1 do 256 bajtów. Każdy zapis poprzedzony jest kluczem. Organizacja indeksowo-sekwencyjna została stworzona dla pamięci dyskowych. Zbiory są zorganizowane sekwencyjnie, a jednocześnie, dzięki możliwości bezpośredniego dostępu do danych, charakteryzującej pamięć dyskową, oraz dzięki kluczom identyfikującym zapisy, możliwe jest przetwarzanie wyrywkowe. Oczywiście, możliwe jest również przetwarzanie sekwencyjne. Kartoteka zapisana jest w trzech obszarach pamięci:

— obszar główny,
— obszar nadmiaru,
— obszar indeksów.

Obszar główny zajmuje wszystkie ścieżki jednego lub kilku cylindrów. Pierwszy cylinder (cylinder 0) jest zarezerwowany dla etykiet. Do obszaru głównego wprowadza się zapisy podczas zakładania kartoteki lub przy jej reorganizacji.



Rys. 53. Struktura indeksów kartoteki imdeksowo-sekwencyjnej

Podczas zakładania kartoteki indeksowo-sekwencyjnej system operacyjny tworzy na podstawie uprzednio posortowanych kluczy co najmniej dwa rodzaje indeksów: indeks cylindrów i indeks ścieżek (por. rys. 53). Indeks cylindrów jest jeden i obejmuje cały zbiór. Składa się on z zapisów, które wskazują indeksy ścieżek i najwyższe klucze cylindrów przechowujących kartotekę. Indeks ścieżek zapisany jest zawsze na pierwszej ścieżce cylindra obszaru głównego. Kolejne zapisy indeksu ścieżek zawierają na przemian normalne i nadmiarowe adresy wejść.

Dodatkowo może być jeszcze utworzony trzeci indeks, indeks główny. Ma to sens w wypadku dużych kartotek, dla których indeks cylindrów zajmuje wiele ścieżek, celem przyspieszenia wyszukiwania odpowiednich pozycji w indeksie cylindrów.

Wszystkie indeksy utworzone są z zapisów składających się z dwóch obszarów: obszaru klucza i obszaru danych. Pierwszy zawiera najwyższy klucz na danej ścieżce (cylindrze), drugi — adres ścieżki.

Bezpośrednio po założeniu kartoteki indeksowo-sekwencyjnej wszystkie zapisy, uporządkowane sekwencyjnie według kluczy, zapisywane są w obszarze danych cylindrów. W czasie przetwarzania kartoteki sekwencyjność ta, w sensie fizycznym, ulega zwykle naruszeniu. Dzieje się tak wówczas, gdy np. wprowadza się nowy zapis do kartoteki. Wówczas, aby uniknąć przesuwania wszystkich zapisów z kluczami o wartości większej od klucza wstawianego zapisu, korzysta się z obszaru nadmiaru.

Wstawiany nowy zapis wpisuje się w odpowiednie miejsce na ścieżce, zgodnie z jego kluczem, a przesunięciu ulegają tylko te zapisy, których klucz jest wyższy i które znajdują się na danej ścieżce. Zapisy, które nie mieszczą się na tej ścieżce, są zapamiętywane w obszarze nadmiaru, przy odpowiedniej korekcji indeksu ścieżek.

Indeksowo-sekwencyjna organizacja kartotek łączy w sobie zalety przetwarzania sekwencyjnego, jak i wyrywkowego, tzn. umożliwia efektywne jednorazowe przetwarzanie większej ilości zapisów, a jednocześnie zapewnia szybki dostęp do pojedynczych zapisów według ich kluczy.

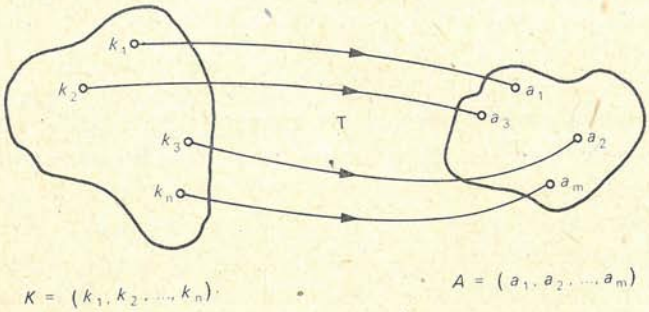
d. Kartoteki losowe

Jeśli kartoteka prowadzona jest w pamięci z bezpośrednim dostępem, można ją zorganizować w sposób zupełnie różny od sekwencyjnego. Ponieważ każdy zapis umieszczony jest w pamięci pod określonym adresem, wystarczy znać zależności między kluczem zapisu a jego adresem.

Taka funkcja może być podana w postaci tabelki, w której każdej wartości klucza przyporządkowany jest adres zapisu. Tabelkę można przechowywać wraz z kartoteką w pamięci zewnętrznej, a na czas przetwarzania kartoteki wczytywać do pamięci operacyjnej. Sposób ten jest jednak nieefektywny, gdyż w wypadku dużych kartotek tabelka taka zajmie dodatkowy obszar pamięci, a oprócz tego system będzie tracił stosunkowo dużo czasu na jej przeszukiwanie.

Znacznie bardziej efektywna jest metoda tzw. *pamięci rozproszonej* (ang. *scatter storage*), w której adres każdego zapisu znajduje się przez pewne przekształcenie, zwykle matematyczne, klucza tego zapisu. Musi więc być określony algorytm odwzorowujący zbiór kluczy K w zbiór adresów A (por. rys. 54).

Dla uproszczenia możemy przyjąć, że pamięć podzielona jest na obszary jednakowej wielkości, zwane działkami. W każdej działce można umieścić jeden zapis, a lokalizację działki określa jej adres początkowy. Stosując proste przeliczenie można posługiwać się adresami względnymi, tzn. liczbami naturalnymi 1, 2, ..., n, którymi ponumerujemy działki. Ilość działek n musi być oczywiście równa lub większa od ilości m zapisów (kluczy) w kartotece.



Rys. 54. Przekształcenie kluczy w adresy

Jeżeli znajdziemy więc taką funkcję T, która odwzorowuje zbiór kluczy K w zbiór adresów pamięci A w sposób jednoznaczny, wówczas przetwarzanie kartoteki będzie bardzo proste i efektywne: chcąc znaleźć zapis o zadanym kluczu k_i obliczamy odpowiadający mu adres a_i i za pomocą jednego dostępu znajdujemy w pamięci dyskowej żądany zapis. Mówiąc o efektywności przetwarzania mamy na myśli czas przetwarzania, którego miarą jest przede wszystkim ilość dostępów do pamięci zewnętrznej,

ponieważ można w przybliżeniu przyjąć, że we współczesnych komputerach czas trwania operacji w procesorze jest znikomo mały w porównaniu z czasem dostępu w pamięciach zewnętrznych, wymagających przesunięć mechanicznych.

Niestety, nie udaje się osiągnąć praktycznie tak doskonałego odwzorowania kluczy w adresy. Często zdarza się, że dwa lub więcej różnych kluczy zostaje odwzorowanych w ten sam adres. Mówimy wówczas o kolizjach lub synonimach. Pojawienie się kolizji nie przekreśla użyteczności metody organizacji kartoteki na zasadzie pamięci rozproszonej, ponieważ istnieją skuteczne metody rozwiązywania problemu synonimów. Przedstawimy kilka metod odwzorowania kluczy w adresy oraz sposoby postępowania z kolizjami.

Spośród wielu metod odwzorowania kluczy w adresy[3] warto zainteresować się przede wszystkim metodą dzielenia, ze względu na jej prostotę i skuteczność. W niektórych wypadkach lepsze wyniki może dać jednak inna metoda.

Metoda dzielenia. Dla danego zapisu z kluczem k_i obliczamy odpowiadający mu adres a_i w ten sposób, że dzielimy wartość klucza przez największy adres obszaru pamięci przeznaczonego dla danej kartoteki. Otrzymana reszta z tego dzielenia jest adresem, a więc

$$a_i = (k_i) \bmod n.$$

Jeżeli n jest liczbą pierwszą, otrzymujemy stosunkowo mało kolizji.

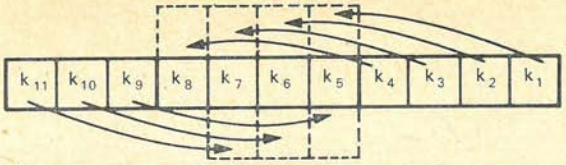
Metoda randomizacji. Do wyznaczania adresów można stosować generatory liczb pseudolosowych. W takim wypadku wejściem generatora jest klucz, wyjściem liczba pseudolosowa, traktowana jako adres. Jednym ze sposobów otrzymywania liczb pseudolosowych może być podnoszenie klucza do kwadratu i wycinanie z wyniku środkowej części, która będzie adresem.

Niekiedy mianem randomizacji określa się ogólnie proces odwzorowania kluczy w adresy.

Metoda składania. Inną, bardzo prostą, metodę ilustruje rysunek 55. Klucz dzieli się na jednakowe części (ostatnia część może być krótsza), które można traktować jako niezależne liczby i sumować.

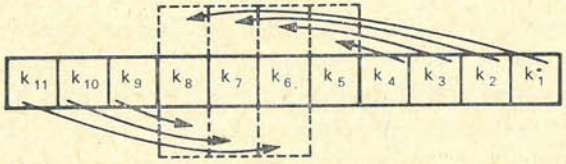
W metodach tych klucz niekoniecznie musi być liczbą. W wypadku kluczy alfanumerycznych, przed odwzorowywaniem należy go doprowadzić do postaci numerycznej, zamieniając litery na cyfry, np. w następujący sposób: A ≡ 11, B ≡ 12, B ≡ 13 itd.

[3] Por. R. Jagielski, *Raspriedielennaja pamiat'*, „Uprawlajuszczije sistiemy i maszyny" 1976, nr 5.



a)

$$a_i = (k_{i4}k_{i3}k_{i2}k_{i1} + k_{i8}k_{i7}k_{i6}k_{i5} + k_{i11}k_{i10}k_{i9}) \ [\mathrm{mod}\ n]$$



b)

$$a_i = (k_{i8}k_{i7}k_{i6}k_{i5} + k_{i1}k_{i2}k_{i3}k_{i4} + k_{i9}k_{i10}k_{i11}0) \ [\mathrm{mod}\ n]$$

Rys. 55. Dwa warianty metody składania

Kolizje są nieuniknione. Na przykład kolejny wprowadzany do pamięci zapis nie może zająć działki o obliczonym dla danego wypadku adresie, ponieważ działka ta jest już zajęta. Wybiera się wtedy, ogólnie rzecz biorąc, jeden z następujących sposobów postępowania:

1) zapisy-synonimy umieszcza się w dodatkowej pamięci, tworząc listę synonimów; szukając np. określonego zapisu w pamięci, należy całą taką listę sekwencyjnie przeszukać,

2) zapisy-synonimy lokuje się w nie zajętych działkach podstawowego obszaru pamięci wyznaczonego dla kartoteki.

Druga metoda jest bardziej interesująca, gdyż nie wymaga dodatkowego obszaru pamięci. Wypadek ten obejmuje kilka różnych strategii. Najprostsza polega na szukaniu najbliższej nie zajętej działki (przy zakładaniu kartoteki) zwiększając kolejno adres o jeden. Metoda ta jest jednak mało efektywna, gdyż zapisy nie są w pamięci rozproszone równomiernie, lecz skupiają się w ciągłe grupy. Powoduje to wzrost czasu wyszukiwania. Jednym ze skutecznych sposobów jest algorytm zaproponowany przez Daya:

(a) ustaw $j = -q$,

(b) odwzoruj klucz w adres początkowy h_o,

(c) jeśli działka h_o nie jest zajęta przetwarzaj (tzn. zapamiętaj zapis, jeżeli jest to zakładanie kartoteki, lub czytaj znaleziony zapis, przy normalnej eksploatacji kartoteki), stop; w przeciwnym razie $j = j+2$,

(d) ustaw $h_i = (h_{i-1} + |j|)$ [mod q],

(e) jeżeli $j<q$ wróć do (c); w przeciwnym wypadku pamięć jest już w całości przeszukana — stop[4].

W algorytmie tym rozmiar pamięci q musi być liczbą pierwszą typu $q = 4n+3$ (n — liczba naturalna).

Należy pamiętać, że ogólna efektywność przedstawionych metod stosowania pamięci rozproszonej zależna jest od stopnia zapełnienia pamięci. Można wprowadzić współczynnik zapełnienia pamięci α równy stosunkowi ilości zapisów w kartotece do ilości działek przeznaczonych dla przechowywania całej kartoteki. Przy $\alpha = 0{,}8$ uzyskuje się zadowalające wyniki (w tym wypadku nie wykorzystuje się w 20% pamięci), natomiast przy większych α efektywność przetwarzania zdecydowanie maleje.

Główną zaletą kartotek zorganizowanych losowo jest mały czas dostępu do dowolnego zapisu, w związku z czym, organizacja taka znajduje zastosowanie przede wszystkim w systemach o wyrywkowym trybie przetwarzania. Nie bez znaczenia jest oczywiście fakt, że kartoteki losowe nie wymagają sortowania.

2. Logiczny System Sterowania Wejściem/Wyjściem

Logiczny System Sterowania Wejściem/Wyjściem (SSWW) jest zespołem środków ułatwiających programiście pisanie programów dla języka ASSEMBLER, przetwarzających duże zbiory danych. Składa się on z modułów operujących zbiorami danych zdefiniowanych w programie problemowym. Dane o zbiorach (kartotekach) umieszcza programista w makroinstrukcjach typu **DTF** (ang. *Define The File*). Logiczny SSWW stanowi wyższy poziom programowania w porównaniu z fizycznym SSWW. Programista zwolniony jest z obowiązku pisania uciążliwych programów kanałowych. Przesyłanie danych realizowane jest za pomocą programów fizycznego SSWW (por. rys. 56), lecz odbywa się to bez udziału programisty.

W przetwarzaniu na poziomie logicznego SSWW biorą udział takie dane, jak:

— nazwa kartoteki,

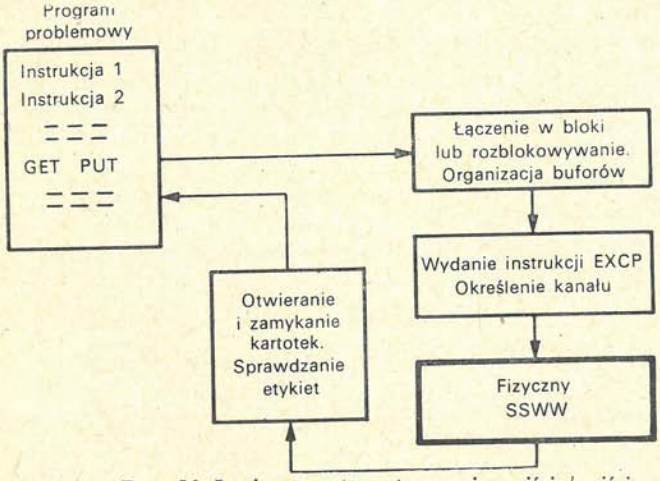
— typ urządzenia zewnętrznego,

— organizacja kartoteki,

[4] A.C. Day, *Full Table Quadratic Searching for Scatter Storage*, „Communications of the ACM" 1970, nr 8

— metoda dostępu do danych,
— format zapisów,
— obszary we/wy,
— etykiety zbiorów itd.

Informacje te podawane są w postaci parametrów instrukcji logicznego SSWW. Wszystkie funkcje przesyłania danych uruchamiane są za pomocą makroinstrukcji **PUT, GET, READ** lub **WRITE.**



Rys. 56. Logiczny system sterowania wejścia/wyjścia

a. Definiowanie kartotek

Użytkownik musi opisać w programie używaną przez siebie kartotekę za pomocą jednej z następujących instrukcji DTF:

DTFCD — zdefiniować kartotekę na kartach perforowanych,
DTFCN — zdefiniować kartotekę na monitorze,
DTFDI — zdefiniować kartotekę niezależną od urządzenia,
DTFMT — zdefiniować kartotekę na taśmie magnetycznej,
DTFSD — zdefiniować kartotekę sekwencyjną na dyskach,
DTFPT — zdefiniować kartotekę na taśmie perforowanej,
DTFPR — zdefiniować kartotekę na drukarce,
DTFIS — zdefiniować kartotekę indeksowo-sekwencyjną,
DTFDA — zdefiniować kartotekę losową.

Siedem pierwszych instrukcji dotyczy kartotek sekwencyjnych.

Na rysunkach od 57 do 63 przedstawiono makroinstrukcje wraz ze wszystkimi możliwymi parametrami (przyjęto przy tym następujące

oznaczenia: *xxxx*.. — oznacza nazwę symboliczną, *nnn*... — oznacza liczbę; w przykładach podajemy maksymalną liczbę znaków). Nie wszystkie wymienione parametry muszą występować w realnej instrukcji **DTF**. Niektóre są wręcz sprzeczne i nie mogą jednocześnie występować. Wybór parametrów zależeć będzie od wymagań użytkownika oraz systemu.

Podajemy alfabetyczny spis parametrów stosowanych w makroinstrukcjach **DTF** wraz z krótkim opisem każdego parametru.

AFTER = YES

Parametr ten powinien być podany wówczas, gdy jakiekolwiek zapisy mają być umieszczone w kartotece poza zapisem, określonym podczas zakładania kartoteki jako ostatni zapis.

BLKSIZE = *nnnnn*

Parametr określający długość obszaru we/wy wskazanego przez **IOAREA1**. W makroinstrukcjach:

DTFCD — jeśli parametr jest opuszczony, system przyjmuje 80,

DTFCN — maksymalnie 256 bajtów,

DTFMT — maksymalnie 32 767 bajtów,

DTFPR — jeśli parametr opuszczony, system przyjmuje 121.

CKPTREC = YES

Parametr konieczny, jeśli kartoteka wejściowa zawiera zapisy punktów kontrolnych.

CONTROL = YES

Parametr ten występuje wówczas, gdy używa się makroinstrukcji CNTRL. Natomiast jeśli stosuje się parametr **CTLCHR**, należy parametr **CONTROL** opuścić.

CRDERR = RETRY

Jeżeli parametr ten jest podany, wtedy w wypadku przekłamania perforatora systemu powtórnie perforuje kartę.

CTLCHR = {**YES** | **ASA**}

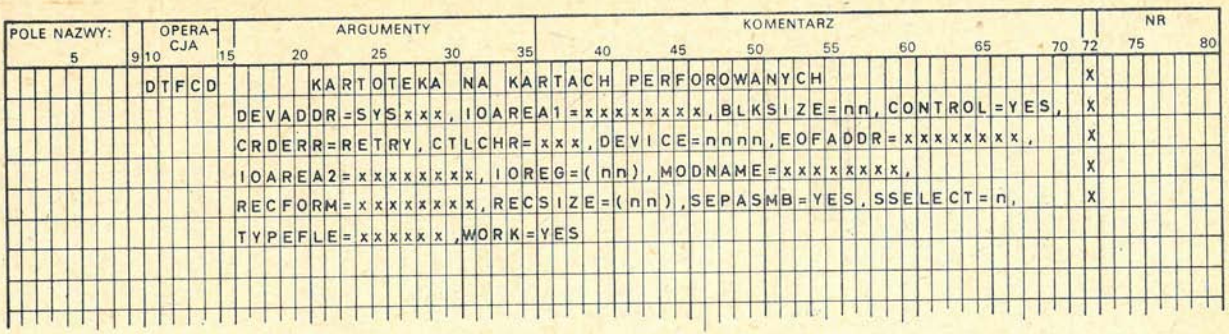
Parametr należy umieścić w makroinstrukcji **DTF** wówczas, gdy pierwszy znak zapisu jest symbolem sterującym. **YES** lub **ASA** oznaczają typ symbolu sterującego.

CYLOFL = *nn*

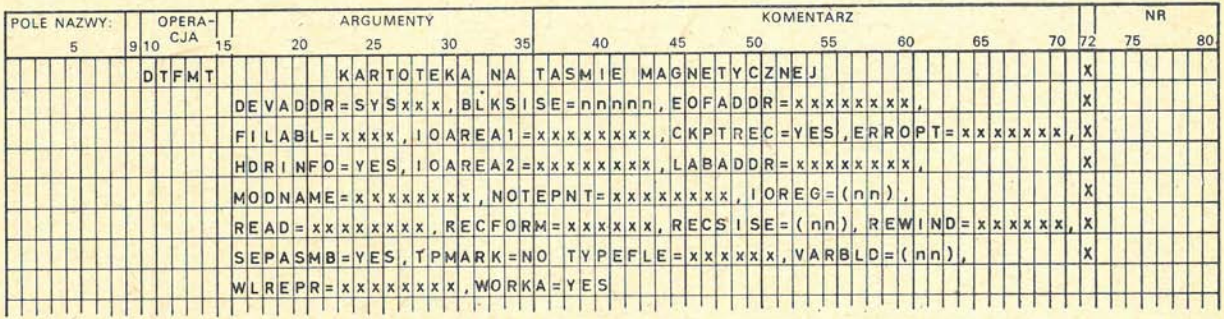
Parametr ten podaje liczbę ścieżek zarezerwowanych dla obszaru nadmiaru cylindra kartoteki indeksowo-sekwencyjnej.

DELETEL = NO

Jeżeli w kartotekach roboczych po instrukcji **CLOSE** etykiety taśm nie powinny ulec zniszczeniu, należy podać ten parametr.



Rys. 57. Makroinstrukcja DTF kartoteki na kartach perforowanych



Rys. 58. Makroinstrukcja DTF dla kartoteki na taśmie magnetycznej

```
POLE NAZWY:   OPERACJA   ARGUMENTY                                   KOMENTARZ                                   NR
5             9 10   15  20        25        30        35  40        45        50        55        60        65  70 72  75  80
              DTFCN          KARTOTEKA NA MONITORZE                                                              X
                  DEVADDR=SYSxxx,IOAREA1=xxxxxxxx,BLKSIZE=nnn,                                                    X
                  RECSIZE=(nn),RECFORM=xxxxxx,TYPEFLE=xxxxxx,WORKA=YES

              DIFDI          KARTOTEKA NIEZALEZNA OD URZADZENIA                                                  X
                  DEVADDR=SYSxxx,EOFADDR=xxxxxxxx,ERROPT=xxxxxxxx,                                               X
                  IOAREA1=xxxxxxxx,IOAREA2=xxxxxxxx,MODNAME=xxxxxxxx                                             X
                  RECSIZE=nnn,SEPASMB=YES,WLRERR=xxxxxxxx,IOREG=(nn)

              DTFPR          KARTOTEKA NA DRUKARCE                                                               X
                  DEVADDR=SYSxxx,IOAREA1=xxxxxxxx,BLKSIZE=nnn,CONTROL=YES,                                       X
                  CTLCHR=xxx,DEVICE=xxxx,IOAREA2=xxxxxxxx,IOREG=(nn),                                            X
                  MODNAME=xxxxxxxx,PRINTOV=YES,RECFORM=xxxxxx                                                    X
                  RECSIZE=(nn),SEPASMB=YES,UCS=xxx,WORKA=YES
```

Rys. 59. Makroinstrukcja: a) DTFCN — dla kartoteki na monitorze, b) DTFDI — kartoteki niezależnej od urządzenia, c) DTFPR — kartoteki na drukarce

DEVADDR = {**SYSPIT** |**SYSPCH** |**SYSRDR** |**SYSLST** |**SYSLNK** |**SYS** *xxx*}

Nazwa urządzenia logicznego przechowującego kartotekę. Konkretna wartość tego parametru zależy od typu kartoteki i musi być zgodna z informacją podaną w operatorze **ASSGN**.

DEVICE = *nnnn*

Typ urządzenia zewnętrznego określonego w parametrze **DEVADDR**. Jest to liczba czterocyfrowa, zgodna ze standardowymi oznaczeniami Jednolitego Systemu.

DSKXTNT = *n*

Parametr ten podaje liczbę segmentów na dysku, określonych przez operator sterujący **EXTEND**.

EOFADDR =*xxxxxxxx*

Parametr określający symboliczny adres podprogramu użytkownika, do którego system przekazuje sterowanie w wypadku stwierdzenia końca kartoteki (koniec kartoteki oznaczony jest operatorem /*).

ERRBYTE = *xxxxxxxx*

Parametr ten jest konieczny wtedy, gdy użytkownik życzy sobie, aby system wydawał dwa bajty zawierające informacje o stanie systemu i przekłamaniach. Bajty te są dostępne dla programu problemowego po wydaniu makroinstrukcji **WAITF**.

ERROPT = {**IGNORE** |**SKIP** |*xxxxxxxx*}

Parametr ten wskazuje rodzaj działania w wypadku wykrycia przekłamania w bloku danych, a więc:

IGNORE — przekłamanie ignoruje się,

SKIP — błędny blok nie jest przetwarzany, system przechodzi do przetwarzania następnego bloku,

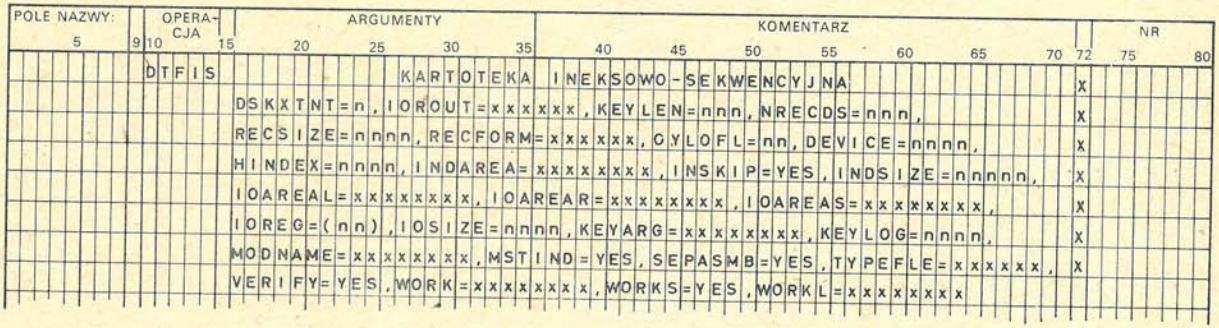
xxxxxxxx — przez podanie nazwy podprogramu użytkownika sterowanie przejmie program problemowy, który obsłuży przekłamanie.

FILABL = {**STD** |**NO** |**NSTD**}

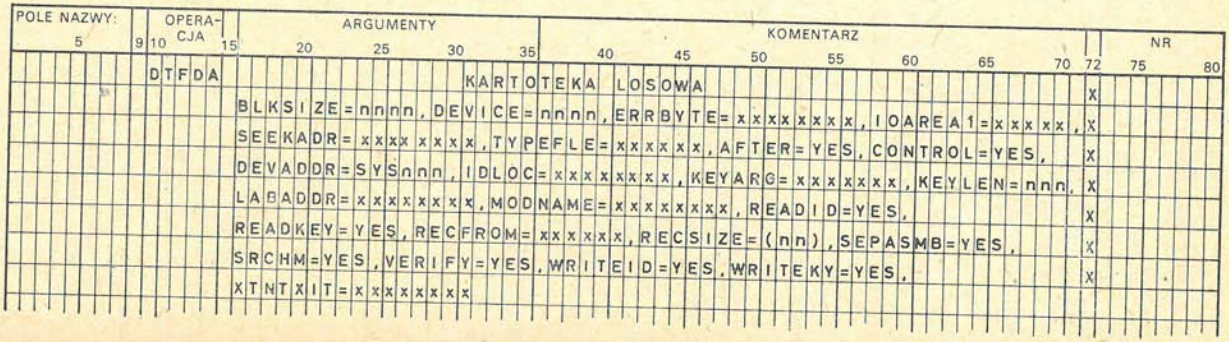
Dotyczy przetwarzania etykiet kartoteki. Parametr **STD** oznacza, że stosuje się etykiety standardowe, **NO** — kartoteki bez etykiet, **NSTD** — — kartoteki z niestandardowymi etykietami.

HINDEX = *nnnn*

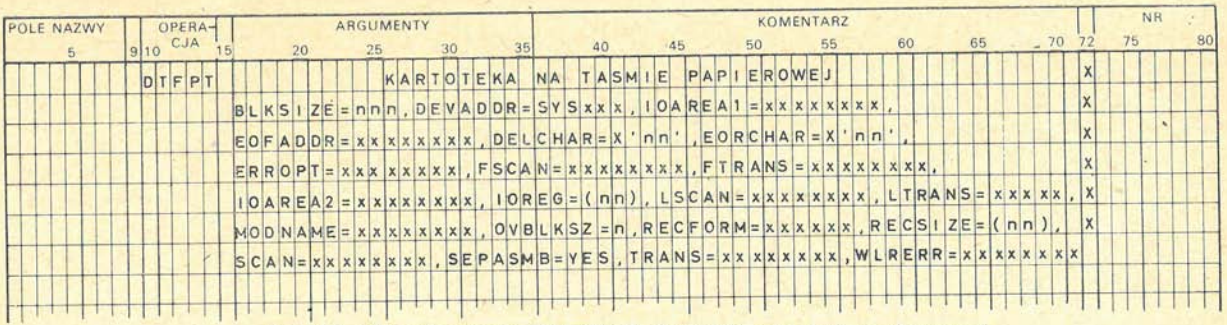
Parametr ten określa typ urządzenia zawierającego indeks wyższego poziomu, np. 5056.



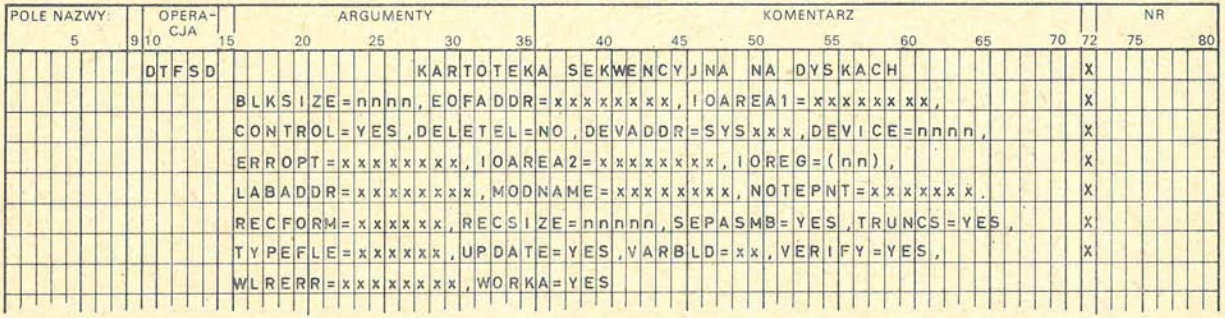
Rys. 60. Makroinstrukcja DTFSD dla kartoteki indeksowo-sekwencyjnej



Rys. 61. Makroinstrukcja DTF dla kartoteki losowej



Rys. 62. Makroinstrukcja DTFPT dla kartoteki na taśmie perforowanej



Rys. 63. Makroinstrukcja DTF dla kartoteki sekwencyjnej na dyskach

HDRINFO = YES

Podając ten parametr można spowodować wydruk zawartości etykiet standardowych, jeżeli podany jest równocześnie parametr **FILABL = STD.**

IDLOC = *xxxxxxxx*

W parametrze tym podaje się symboliczny adres pola pamięci, do którego system wnosi identyfikator zapisu.

INDAREA = *xxxxxxxx*

Parametr ten określa symboliczny adres pola pamięci, do którego będzie zapisywany indeks cylindrów. Musi być jednocześnie podany parametr **INDSIZE.**

RETRVE — aktualizacja kartoteki.

ADDRTR — uzupełnianie kartoteki lub jej aktualizacja.

INDSKIP = YES

Parametr ten powoduje pominięcie zapisów indeksu cylindrów. Powinien występować łącznie z **INDAREA** i **INDSIZE**. Zadaniem tych parametrów jest przyspieszenie przetwarzania.

INDSIZE = *nnnnn*

Parametr ten określa długość obszaru dla zapisów indeksu cylindrów, określonego w **INDAREA.**

IOAREAL = *xxxxxxxx*

Symboliczna nazwa występująca w tym parametrze określa obszar pamięci, wykorzystywany podczas tworzenia kartoteki lub dopisywania nowych zapisów.

IOAREAR = *xxxxxxxx*

Parametr ten, konieczny przy przetwarzaniu wyrywkowym, wskazuje adres symboliczny dodatkowego obszaru we/wy, niezbędnego w tym wypadku.

IOAREAS = *xxxxxxxx*

W przetwarzaniu kartotek indeksowo-sekwencyjnych metodą sekwencyjną parametr ten wskazuje obszar we/wy.

IOAREA1 = *xxxxxxxx*

Nazwa pierwszego obszaru we/wy.

IOAREA2 = *xxxxxxxx*

Jeżeli drugi obszar we/wy jest używany, wówczas w tym parametrze podaje się jego nazwę.

IOREG = (*nn*)

Parametr podaje numer rejestru, w którym umieszcza się adres przetwarzanego zapisu.

IOROUT = {**LOAD** | **ADD** | **RETRVE** | **ADDRTR**}

Parametr ten określa rodzaj wykonywanej funkcji:

LOAD — tworzenie kartoteki lub rozszerzanie jej poza ustalone uprzednio granice,

ADD — wstawianie nowych zapisów.

IOSIZE = *nnnn*

Parametr ten podaje długość obszaru we/wy, którego adres symboliczny określa parametr **IOAREAL**.

KEYARG = *xxxxxxxx*

Za pomocą parametru **KEYARD** podaje się adres symboliczny pola zawierającego klucz zapisu.

KEYLEN = *nnn*

Parametr ten określa długość klucza w kartotece.

KEYLOC = *nnnn*

Przy przetwarzaniu zapisów zblokowanych parametr ten określa pozycję klucza w zapisie.

LABADDR = *xxxxxxxx*

Parametr ten podaje adres symboliczny podprogramu użytkownika, którego zadaniem jest przetwarzanie etykiet niestandardowych.

MODNAME = *xxxxxxxx*

W tym parametrze podaje się nazwę modułu, który powinien być użyty razem z daną instrukcją **DTF**. Jeżeli parametr ten jest opuszczony, system używa standardowej nazwy modułu.

MSTIND = **YES**

Jeżeli używa się indeksu głównego, parametr ten musi wystąpić w makroinstrukcji **DTF**.

NOTEPNT = {**POINTS** | **YES**}

Jeżeli w programie użyte będą makroinstrukcje **NOTE**, **BOINTW**, **POINTR** lub **POINTS**, konieczne jest podanie parametru **YES**. Parametr **POINTS** wskazuje, że używa się tylko instrukcji o tej nazwie.

NRCDS = *nnn*

Parametr ten informuje system o ilości zapisów w bloku, o ile stosuje się blokowanie zapisów.

PRINTOV = **YES**

Parametr ten wskazuje, że w programie używa się makroinstrukcji **PRTOV**, służącej do kontroli przepełnienia formatu strony.

READ = {FORWARD |BACK}

Parametr ten wskazuje kierunek czytania taśmy magnetycznej: **FORWARD** — do przodu, **BACK** — wstecz.

READID = YES

Parametr ten powinien wystąpić w makroinstrukcji **DTF** wówczas, gdy stosuje się w programie makroinstrukcję **READ ID**, tzn. przy dostępie do zapisów za pomocą identyfikatora.

READKEY = YES

Parametr ten konieczny jest w wypadku wyszukiwania zapisów za pomocą kluczy.

RECFORM = {FIXUNB |FIXBLK |VARUNB |VARBLK |UNDEF}

W tym parametrze podaje się stosowany format zapisów. Nie zawsze można użyć wszystkich formatów, a mianowicie w makroinstrukcjach:

DTFCD — **{FIXUNB |VARUNB |UNDEF}**

DTFCN, DTFDA — **{FIXUNB |UNDEF}**

DTFIS — **{FIXUNB |FIXBLK}**

RECSIZE = (*nn*)

Parametr ten podaje długość zapisu (jeśli kartoteka składa się z zapisów stałej długości) lub numer rejestru zawierającego długość zapisu.

REWIND = {UNLOAD |NORWD}

Jeśli parametr ten jest opuszczony, wówczas taśma jest automatycznie przewijana do punktu startu, ale nie zostaje rozładowana. Podanie tego parametru powoduje: **UNLOAD** — przewijanie taśmy z rozładowaniem, **NORWD** — nie pozwala na przewijanie taśmy.

SEEKADR = *xxxxxxxx*

Parametr ten określa pole pamięci, w którym mieści się adres zapisu na dysku. Stosuje się przy wyszukiwaniu zapisów według identyfikatora.

SEPASMB = YES

Parametr ten należy podać wtedy, gdy **DTF** ma być translowana oddzielnie, niezależnie od programu.

SRCHM = YES

Parametr ten powoduje szukanie zapisu w kartotece według klucza od miejsca wskazanego przez **SEEKADR** na wszystkich następnych ścieżkach.

SSELECT = *n*

Numer wybranego magazynka w perforatorze kart. Może przyjąć wartość 1 lub 2.

TRUNCS = YES

Jeżeli kartoteka z zapisami blokowanymi stałej długości będzie zawierać skrócone bloki lub użyta będzie makroinstrukcja **TRUNC**, wtedy należy podać ten parametr.

TPMARK = NO

Parametr **TPMARK** należy zastosować wówczas, gdy użytkownik rezygnuje ze znacznika taśmy, zapamiętanego jako pierwszy zapis kartoteki wyjściowej bez etykiet.

TYPEFLE = {INPUT|OUTPUT|RANDOM|SEQNTL|RANSEQ}

Parametr ten określa typ kartoteki: kartoteka wejściowa, wyjściowa, losowa, sekwencyjna na dysku, sekwencyjna i losowa łącznie.

UCS = {ON|OFF}

Jeżeli drukarka ma specjalny bufor UCS, parametr ON wskazuje, że jeżeli pojawią się kody nie istniejących w drukarce znaków, wówczas działanie wskaże operator. Parametr OFF powoduje ignorowanie takich przekłamań i znaki takie traktuje jako spacje.

UPDATE = YES

Parametr **UPDATE** informuje system o tym, że kartoteka będzie aktualizowana.

VARBLD = (*nn*)

Parametr ten wskazuje numer rejestru, który zawiera długość pozostałego wolnego pola w obszarze wyjścia. Dotyczy przetwarzania, bez obszaru roboczego, blokowanych zapisów zmiennej długości.

VERIFY = YES

Za pomocą tego parametru można ustalić pracę systemu w ten sposób, że będzie on prowadził kontrolę parzystości zapisów w pamięci dyskowej.

WLRERR = *xxxxxxxx*

Na skutek tego parametru, jeżeli w kartotece wejściowej przeczyta się zapis z nieprawidłową długością, sterowanie zostanie przekazane do podprogramu, którego adres symboliczny określa parametr.

WORKA = YES

Za pomocą parametru **WORKA** podejmuje się decyzję o przetwarzaniu zapisów w roboczym obszarze pamięci.

WORKL = *xxxxxxxx*

Parametr ten podaje adres symboliczny obszaru we/wy, niezbędnego dla przechowywania zapisu podczas zakładania **(LOAD)** i rozszerzania **(ADD)** kartoteki.

WORKR = *xxxxxxxx*

Parametr **WORKR** określa obszar roboczy, konieczny przy przetwarzaniu wyrywkowym, gdy zapisy mają być przetwarzane w obszarze roboczym, a nie w obszarze we/wy.

WORKS = **YES**

Parametr ten mówi o tym, że w przetwarzaniu sekwencyjnym kartoteki używany będzie obszar roboczy.

WRITEID = **YES**

Parametr ten musi być użyty wtedy, gdy za pomocą identyfikatora wskazuje się miejsce zapisu na dysku używając makroinstrukcji **WRITE ID.**

WRITEKY = **YES**

Parametr ten jest konieczny dla wskazania, że miejsce na dysku określa klucz (w programie używa się makroinstrukcji **WRITE KEY).**

XTNTXIT = *xxxxxxxx*

Parametr ten podaje adres podprogramu służącego do przetwarzania informacji o segmentach kartoteki na dysku.

b. Przetwarzanie kartotek przy użyciu logicznego SSWW

Programista używając deklaracyjnych instrukcji **DTF** opisuje kartoteki, z którymi zamierza pracować. Makroinstrukcje **DTF** tworzą w pamięci tzw. tablice **DTF**, które zawierają wszelkie konieczne informacje dla przetwarzania kartotek. Każdej instrukcji **DTF** (z wyjątkiem **DTFCN**) towarzyszy instrukcja operatywna *xx***MOD** (*xx* — oznacza rodzaj kartoteki, np. **CDMOD** — dotyczy kartoteki na kartach perforowanych), która uruchamia logiczny moduł SSWW. Jest to podprogram systemowy, do którego należy: czytanie i zapisywanie danych, kontrola sytuacji wyjątkowych, łączenie zapisów w bloki, wydzielanie zapisów z bloku, umieszczenie zapisu w obszarze roboczym itd. Włączenie do modułu logicznego tych lub innych funkcji dokonuje się przez podanie odpowiednich parametrów w makroinstrukcji *xx***MOD.** Programista ma możność wyboru pewnych funkcji, jak również zrezygnowania z wielu parametrów, a nawet ograniczenia się wyłącznie do standardowych wartości parametrów. Odpowiednie moduły są włączane do programu użytkownika, przy czym programista może translować moduły **DTF** i logiczne moduły *xx***MOD** razem ze swoim programem lub też oddzielnie, przewidując ich późniejsze łączenie za pomocą programu **REDAKTOR.**

Moduły *xx***MOD** znajdują się w bibliotece modułów wynikowych i źródłowych. Dla każdego typu modułu może występować kilka wariantów o różnych nazwach. Programista podaje odpowiednią nazwę modułu logicznego w parametrze **MODNAME** w makroinstrukcji **DTF**, a jeśli tego nie uczyni, system przyjmuje nazwę standardową.

Makroinstrukcjom **DTFCD, DTFPR, DTFIS, DTFSD, DTFDA, DTFMT, DTFDI** odpowiadają moduły **CDMOD, PRMOD, ISMOD, SDMOD, DAMOD, MTMOD, DIMOD.**

Przetwarzanie rozpoczynają imperatywne instrukcje **PUT** i **GET**, w wypadku kartotek sekwencyjnych, oraz **READ** i **WRITE**, w wypadku kartotek z bezpośrednim dostępem.

Przetwarzanie sekwencyjne

Instrukcja **GET:**

```
[nazwa] GET nazwa-kartoteki [,nazwa-obszaru-roboczego]
```

Instrukcja **GET** udostępnia dla przetwarzania zapisy w obszarze we/wy lub obszarze roboczym.

Instrukcja **PUT:**

```
[nazwa] PUT nazwa-kartoteki [,nazwa-obszaru-roboczego]
```

Instrukcja **PUT** powoduje wyprowadzenie zapisów kartoteki ze wskazanych obszarów pamięci na zewnętrzne nośniki informacji.

Instrukcje **GET** i **PUT** stosuje się wyłącznie dla kartotek wejściowych lub wyjściowych z zapisami blokowanymi, stałej, zmiennej lub nieokreślonej długości. W instrukcjach tych tylko nazwa kartoteki jest parametrem obowiązkowym. Przez odpowiedni dobór obszarów we/wy możliwe jest przetwarzanie zapisów i jednoczesne przesyłanie danych. Nakładanie się tych operacji może bardzo przyspieszyć wykonanie programu. Tablica 27 przedstawia efektywność nakładania dla różnych kombinacji obszarów we/wy i obszaru roboczego.

Przetwarzanie wyrywkowe

Przetwarzanie kartotek w urządzeniach pamięciowych o bezpośrednim dostępie może odbywać się metodami wyszukiwania zapisów na podstawie:

— adresu rzeczywistego ścieżki,
— identyfikatora lub klucza zapisu.

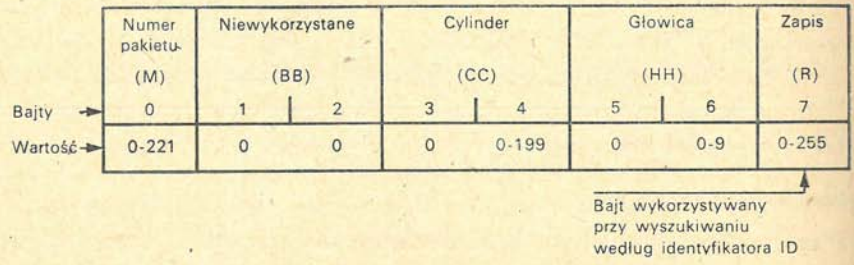
TABLICA 27

Efektywność jednoczesnego wykonywania operacji we/wy i przetwarzania

Zapisy	Liczba obszarów we/wy	Obszar roboczy	Efektywność nałożenia (overlap) operacji we/wy i przetwarzania
Nieblokowane	1	nie ma	Nałożenie tylko dla buforowanych urządzeń we/wy. Nie ma nałożenia dla pamięci taśmowych i dyskowych.
		jest	Nałożenie dla każdego zapisu
	2	nie ma	Nałożenie dla każdego zapisu
		jest	Nałożenie dla każdego zapisu. Obszar roboczy nie daje żadnych efektów.
Zblokowane	1	nie ma	Nie ma nałożenia
		jest	Nie ma nałożenia
	2	nie ma	Nałożenie dla całego bloku
		jest	Nałożenie dla całego bloku. Obszar roboczy nie daje efektów.

Przy pierwszej metodzie dostępu do zapisu kartoteki programista podaje adres ścieżki (por. rys. 64), na której znajduje się szukany zapis. Adres ten należy umieścić w polu pamięci określonym w **DTFDA** za pomocą parametru **SEEKADR**. Ósmy bajt adresu, oznaczony na rysunku 64 literą R, nie jest w tym wypadku wykorzystany.

Można bardziej dokładnie określić położenie zapisu na ścieżce podając jego identyfikator. Identyfikator zawiera adres ścieżki, na której znajduje



Rys. 64. Format adresu zapisu w pamięci dyskowej

się interesujący nas w danym momencie zapis, oraz kolejny numer tego zapisu na ścieżce (bajt R na rys. 64).

Inny sposób określania zapisu w kartotece to podanie jego klucza. W tym wypadku przed przetwarzaniem zapisu należy podać wartość klucza w polu pamięci o adresie określonym parametrem **KEYARG** w **DTFDA.**

Przetwarzanie zapisu kartoteki, znajdującego się w obszarze pamięci **IOAREA1**, powodują makroinstrukcje **WRITE** lub **READ.** Makroinstrukcja **READ** powoduje przekazanie zapisu z pamięci dyskowej do obszaru we/wy:

```
[nazwa] READ nazwa-kartoteki , { KEY }
                               { ID  }
```

W wyniku makroinstrukcji **WRITE** zapis znajdujący się w obszarze zostaje przesłany do odpowiedniego miejsca pamięci dyskowej:

```
[nazwa] WRITE nazwa-kartoteki , { KEY   }
                               { ID    }
                               { AFTER }
                               { RZERO }
```

Makroinstrukcje **READ** i **WRITE** z parametrem **KEY** umożliwiają bezpośredni dostęp do dowolnego zapisu na podstawie wartości klucza. Pole klucza powinno być określone w makroinstrukcji **DTFDA** przez parametr **KEYARG.** SSWW szuka najpierw wskazanej ścieżki, a następnie przeszukuje tę ścieżkę w celu znalezienia zapisu z podanym kluczem (jeśli w **DTFDA** zadany był parametr **SRCHM,** system przeszukuje również następne ścieżki tak długo, dopóki nie znajdzie żądanego zapisu).

Makroinstrukcje **READ** i **WRITE** z parametrem **ID** realizują wyszukiwanie według identyfikatora. W tym wypadku w adresie zapisu na dysku powinno być podane również pole R, zawierające kolejny numer zapisu na ścieżce. W instrukcji **DTFDA** powinien wystąpić parametr **READ ID** lub **WRITE ID.** Instrukcje czytania i zapisu zostają wykonywane w ten sposób, że system odszukuje wskazaną ścieżkę, a na ścieżce znajduje zapis według numeru.

Ścieżkę na dysku znajduje system na podstawie adresu w polu o nazwie wskazanej w parametrze **SEEKADR** w **DTFDA.**

Makroinstrukcja **WRITE** z argumentem **AFTER** umożliwia dopisanie zapisu do kartoteki za ostatnim zapisem kartoteki, na wskazanej ścieżce.

Jeżeli na ścieżce tej brakuje miejsca, zapis nie jest wprowadzany, a system wydaje odpowiedni komunikat.

Makroinstrukcja **WRITE** z argumentem **RZERO** przygotowuje do przetwarzania ścieżkę ukazaną w polu określonym parametrem **SEEKADR**.

Przetwarzanie kartoteki indeksowo-sekwencyjnej

Przed zapisaniem kartoteki indeksowo-sekwencyjnej na dyskach należy zapisy posortować. Zakładanie lub rozszerzanie kartoteki wymaga podania w **DTFIS** parametru **LOAD**. Na pakiecie dysków należy przygotować segmenty pamięci dla obszaru danych i indeksów, używając do tego celu operatora sterującego **EXTENT**. Następnie programista może użyć instrukcji **SETFL**:

```
[nazwa] SETEL nazwa-kartoteki
```

Makroinstrukcja **SETFL** powoduje sprawdzenie, czy przeznaczone dla kartoteki segmenty pamięci są wystarczająco duże, aby pomieścić całą kartotekę. W wypadku niezgodności system wydaje odpowiedni komunikat.

Zapisy kartoteki wprowadza się do pamięci dyskowej za pomocą makroinstrukcji **WRITE**:

```
[nazwa] WRITE nazwa-kartoteki, NEWKEY
```

Instrukcja **WRITE** powoduje przesyłanie zapisu wraz z kluczem z obszaru roboczego, określonego w **DTF** argumentem **WORKL**, do obszaru we/wy określonego przez **IOAREA1**, gdzie tworzy się w zapisie obszar licznika i przekazuje się go na dysk. Położenie danego zapisu na dysku określa adres w postaci takiej, jak to pokazuje rysunek 64. Adresy te udostępnia użytkownikowi system w polu o symbolicznej nazwie utworzonej z nazwy kartoteki z dodatkiem litery H. Na przykład jeśli kartoteka ma nazwę **KARMAT**, to adres aktualnie przetwarzanego zapisu będzie znajdować się w ośmiobajtowym polu o nazwie **KARMATH**. Programista może np. wydrukować adresy dowolnych zapisów.

Po zapisaniu kartoteki na dysku należy podać instrukcję **ENDFL**:

```
[nazwa] ENDFL nazwa-kartoteki
```

Instrukcja ta powoduje wykonanie czynności końcowych, takich jak zapisanie ostatniego bloku danych, zapisanie rekordu końca kartoteki zapisów indeksów itd.

Z sekwencyjno-indeksowej kartoteki można korzystać dwojako przetwarzając zapisy wyrywkowo lub sekwencyjnie:

a) przetwarzanie wyrywkowe (w **DTFIS** powinien być podany parametr **TYPEFLE = RANDOM**).

W przetwarzaniu wyrywkowym zapisy są wyszukiwane na podstawie wartości klucza, który powinien być umieszczony w pamięci operacyjnej w polu określonym przez argument **KEYARG** w **DTFIS**. Dla czytania zapisu wskazanego przez klucz używa się makroinstrukcji **READ**:

```
[nazwa] READ nazwa-kartoteki, KEY
```

W wyniku tej instrukcji system przegląda indeksy dla określenia ścieżki, na której znajduje się żądany zapis. Następnie szuka tej ścieżki na pakiecie dysków. Gdy zapis zostaje znaleziony, przesyła się go do obszaru we/wy określonego w **DTF** jako **IOAREAR** oraz do obszaru roboczego **WORKR**, jeśli był on podany.

Podobnie, lecz z przesyłaniem informacji w odwrotnym kierunku, działa instrukcja **WRITE**:

```
[nazwa] WRITE nazwa-kartoteki, KEY
```

Instrukcji tej używa się do aktualizacji zapisów. Każdą instrukcję **WRITE** z argumentem **KEY** musi poprzedzić instrukcja **READ**.

Za instrukcjami **WRITE** i **READ** powinna następować instrukcja **WAITF**, która zapewnia, że przetwarzanie zapisów nie rozpocznie się przed zakończeniem operacji przesyłania danych, zainicjowanych poprzedzającymi **WAITF** instrukcjami. Instrukcja **WAITF** ma postać:

```
[nazwa] WAITF nazwa-kartoteki
```

b) przetwarzanie sekwencyjne (w **DTFIS** parametr **TYPEFLE = SEQNT** lub **RANSEQ**).

Przetwarzanie sekwencyjne kartoteki indeksowo-sekwencyjnej może rozpocząć się od początku kartoteki lub od pewnego zapisu, określonego

kluczem lub identyfikatorem. Przygotowanie kartoteki do przetwarzania i ustalenie granic przetwarzania sekwencyjnego zapewnia makroinstrukcja SETL:

```
[nazwa]  SETL  nazwa-kartoteki , { adres-ID | KEY | GKEY | BOF }
```

Drugi argument wskazuje, z jakiego miejsca winno rozpocząć się przetwarzanie sekwencyjne:

adres-**ID**, podaje się nazwę pola zawierającego identyfikator (por. rys. 64), określający adres pierwszego zapisu sekwencji,

KEY — oznacza, że przetwarzanie sekwencyjne powinno rozpocząć się od klucza podanego w polu określonym przez **KEYARG**,

GKEY — określa początek grupy zapisów, które mają być przetwarzane sekwencyjnie; np. jeśli w polu klucza znajduje się wartość BZ21ØØØØ, wówczas przetwarzane będą wszystkie zapisy z kluczem BZ21xxxx, niezależnie od wartości *x*,

BOF — oznacza sekwencyjne przetwarzanie całej kartoteki.

Przetwarzanie rozpoczynają instrukcje **GET** (czytać) lub **PUT** (zapisać):

```
[nazwa]  GET  nazwa-kartoteki,[,nazwa-obszaru-roboczego]
[nazwa]  PUT  nazwa-kartoteki,[,nazwa-obszaru-roboczego]
```

Jeżeli zapisy przetwarzane są tylko w obszarze we/wy, w instrukcjach **GET** i **PUT** podaje się tylko nazwę kartoteki. Przy użyciu obszaru roboczego zapisy przenoszone są do tego obszaru, gdzie mogą być przetwarzane. W tym wypadku nazwę obszaru roboczego należy podać w instrukcji **GET** lub **PUT**.

Na zakończenie przetwarzania sekwencyjnego kartoteki należy podać instrukcję **ESETL**:

```
[nazwa]  ESETL  nazwa-kartoteki
```

Przed użyciem instrukcji **GET**, **PUT**, **READ** i **WRITE** należy przygotować kartotekę do przetwarzania. System powinien sprawdzić lub zapisać

```
         BALR   3,0
         USING  *,3
         PRINT  NOGEN
         OPEN   IN, OUT1, OUT2
READ     GET    IN
         PACK   ILOSCP(3),ILOSC (4)
         PACK   CENAP(3),CENA (5)
         ZAP    WARTP(6),CENAP (3)
         MP     WARTP(6),ILOSOP (3)
         UNPK   WART(12),WARTP (16)
         MVZ    WART+11(1), = X'FF'
         AP     SUMAP(6),WARTP(6)
         PUT    OUT1
         BC     15, READ
END      UNPK   SUMA(12),SUMAP(6)
         MVZ    SUMA+11(1), = X'FF'
         PUT    OUT2
         CLOSE  IN,OUT1,OUT2
         EOJ
IN       DTFCD  DEVADDR = SYSIPT,
                IOAREA1 = ZAPIS,
                BLKSI7E = 80,
                DEVICE = 6012,
                EDFADDR = END
OUT1     DTFPR  DEVADDR = SYSLST,
                IOAREA1 = ZAPIS,
                BLKSIZE = 92,
                DEVICE = 7030
OUT2     DTFPR  DEVADDR = SYSLST,
                IDAREA1 = SUMA,
                BLKSIZE = 12,
                DEVICE = 7030
         CDMOD
         PRMOD
ZAPIS    DS     OCL80
SYMBOL   DS     CL8
         DS     CL3
NAZWA    DS     CL20
CENA     DS     CL5
ILOSC    DS     CL4
         DS     CL40
WART     DS     CL12
WARTP    DS     CL6
SUMA     DS     CL12
SUMAP    DC     PL6'0'
CENAP    DS     CL3
ILOSCP   DS     CL3
         END
```

Rys. 65. Program rozwiązujący zadanie przykładu 58 w rozdziale VIII.

etykiety kartotek. Dokonuje się tego za pomocą makroinstrukcji **OPEN** i **CLOSE**.

Instrukcja **OPEN** (otwórz kartotekę):

```
[nazwa] OPEN nazwa-kartoteki-1 [,nazwa-kartoteki-2]...
```

Instrukcja **CLOSE** (zamknij kartotekę):

```
[nazwa] CLOSE nazwa-kartoteki-2 [,nazwa-kartoteki-2]...
```

Za pomocą jednej instrukcji **OPEN** lub **CLOSE** można stworzyć lub zamknąć do 16 kartotek.

Przykład 1

Sposób użycia makroinstrukcji logicznego SSWW pokażemy na przykładzie. Przytoczony w przykładzie program (por. rozdz. VI, przykład 58) rozwiązywał zadanie na podstawie fizycznego systemu sterowania we/wy. To samo zadanie rozwiązane przy użyciu logicznego SSWW przedstawia program na rysunku 65.

3. Przetwarzanie kartotek w języku PL/1

Trzy podstawowe metody organizacji kartotek: sekwencyjną, indeksowo-sekwencyjną i losową zapewnia standardowo język PL/1 DOS JS. Przetwarzanie kartotek w języku PL/1 wymaga znacznie mniej wysiłku od programisty, niż to było konieczne przy użyciu logicznego SSWW. Organizacje kartotek, omówione w rozdziale VIII, nie wymagają w odniesieniu do języka PL/1 żadnych dodatkowych wyjaśnień. Kilka słów należy jedynie poświęcić organizacji losowej. Możliwe są tu dwa warianty: REGIONAL(1) i REGIONAL(3).

W kartotekach typu REGIONAL(1) pamięć dyskową dzieli się na obszary stałej długości (działki). W każdej działce umieszcza się co najwyżej jeden zapis. Podziału pamięci na działki dokonuje się za pomocą standardowego programu, zwanego programem formatyzacji. Każdy obszar jest numerowany liczbami 0,1,2,3,... System przyjmuje, że numery te, zwane odtąd kluczami, są ośmiocyfrowymi liczbami, które należy deklarować w następujący sposób:

DECLARE REGNR PICTURE' (8)9**';**

W rzeczywistości numery te (klucze) nie są zapisywane w pamięci. Fizyczny SSWW oblicza każdorazowo adres działki na podstawie danego klucza. Ponieważ odbywa się to bez udziału użytkownika, z jego punktu widzenia można przyjąć, że klucze te są już adresami, ponieważ umożliwiają jednoznaczne zlokalizowanie zapisów w kartotece.

Organizacja typu **REGIONAL**(3) różni się tym, że w jednej działce pamięci umieszcza się więcej niż jeden zapis. Jedna działka zajmuje jedną ścieżkę pamięci. Działki są ponumerowane względem początku kartoteki. Każdy zapis w działce powinien zawierać jakiś wyróżnik, pozwalający na identyfikację zapisu. Wyróżnik ten zwany kluczem zewnętrznym, wnoszony jest do działek pamięci przed zapisem, razem z numerem działki. Zwykle opisuje się go w następujący sposób:

```
DECLARE 1 KL,
        2 KLUCZ_ZEW (n),
        2 REGNR PICTURE '(8)9';
```

Powinny być spełnione warunki: $n \geqslant 1$ i $m = n+8$. Można określić klucz również w inny sposób, np. jako konkatenację klucza zewnętrznego i numeru działki.

a. Deklarowanie kartotek

Każda kartoteka musi być opisana za pomocą instrukcji **DECLARE**. Zdanie **DECLARE** zawiera atrybuty zbioru oddzielone przecinkami, przy czym za słowem **DECLARE** następuje nazwa kartoteki, a następnie słowo **FILE**. Atrybuty mogą być alternatywne, czyli takie, których konkretną wartość ustala programista, lecz których obecność jest obowiązkowa. oraz uzupełniające, jakie nie zawsze muszą wystąpić.

Atrybuty alternatywne

1. Atrybuty sposobu przetwarzania kartoteki:

STREAM | RECORD

Przy każdorazowej transmisji danych z urządzenia zewnętrznego do pamięci operacyjnej i odwrotnie przesyłanie odbywa się blokami do (z) obszaru we/wy, który pełni rolę bufora. Może się to odbywać w dwojaki sposób:

RECORD — zapisy z bufora przenoszone są bez żadnych zmian do obszaru pamięci przeznaczonego dla danych lub przetwarzanie następuje bezpośrednio w buforze,

STREAM — w tym wypadku zapis nie jest traktowany jako jedna całość, lecz znaki przesyłane są strumieniem i przetwarzane w takiej kolejności, w jakiej umieszczane są w buforze; często podczas przesyłania danych następuje przekształcenie postaci danych (dokładniej operacje we/wy dokonywane w trybie **STREAM** przedstawione są w rozdziale VII).

2. Atrybuty funkcji kartoteki:

INPUT | OUTPUT | UPDATE

INPUT — oznacza, że dane mogą być przekazywane tylko z kartoteki do pamięci głównej,

OUTPUT — oznacza, że dane mogą być przekazywane tylko z pamięci głównej do kartoteki w urządzeniu zewnętrznym,

UPDATE — oznacza, że dane mogą być pobrane ze zbioru pamięci zewnętrznej do pamięci głównej, tu przetworzone i następnie przesłane z powrotem do kartoteki.

3. Atrybuty dostępu:

SEQUENTIAL | DIRECT

Atrybuty te określają sposób dostępu do danych wewnątrz kartoteki:
SEQUENTIAL — oznacza dostęp sekwencyjny,
DIRECT — oznacza dostęp bezpośredni.
Należy dodać, że kartoteki z atrybutem **STREAM** mogą mieć wyłącznie atrybut dostępu **SEQUENTIAL**, natomiast kartoteki przetwarzane zapisami mogą mieć atrybuty **SEQUENTIAL** lub **DIRECT**.

4. Atrybuty buforowania:

BUFFERED | UNBUFFERED

Bufor jest pamięcią pośrednią, mającą za zadanie zwiększenie efektywności przesyłania danych. Największa efektywność przetwarzania będzie

wówczas, gdy jednocześnie z obliczeniami wykonywane są operacje we/wy.
BUFFERED — atrybut ten powoduje utworzenie w pamięci głównej jednego obszaru jako bufora; wielkość tego obszaru równa jest długości bloku danych; najbardziej celowe jest używanie buforu przy przetwarzaniu sekwencyjnym,
UNBUFFERED — atrybut przetwarzania bez bufora.

Atrybuty uzupełniające

PRINT — oznacza, że kartoteka przeznaczona jest do wyprowadzania danych na drukarkę. Zakłada się przy tym atrybuty **STREAM** i **OUTPUT**, które w tym wypadku mogą zostać w programie opuszczone.

KEYED — stosuje się w tych zbiorach, w których zapisy są rozpoznawane za pomocą kluczy.

BACKWARDS — atrybut ten odnosi się wyłącznie do zbiorów na taśmie magnetycznej z atrybutami **RECORD, SEQUENTIAL, INPUT.**
W danym wypadku wprowadzanie danych może odbywać się od końca, tzn. w kolejności od ostatniego zapisu do pierwszego.

Atrybut ENVIRONMENT

Atrybut **ENVIRONMENT** ma specjalne znaczenie i jest wymagany w każdym zdaniu **DECLARE**. Ta odrębność wynika z faktu, iż podawane tu parametry nie są elementami języka PL/1, lecz związane są z systemem operacyjnym (w tym wypadku z DOS JS). Postać atrybutu **ENVIRONMENT** jest następująca:

ENVIRONMENT *(lista-parametrów)*

Listę parametrów tworzą następujące składniki:
1. Urządzenie we/wy:

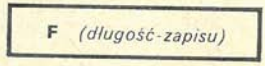
MEDIUM *(***SYS***xxx, nnnn)*

W parametrze tym **SYS***xxx* oznacza nazwę logicznego urządzenia we/wy, *nnnn* — typ urządzenia zewnętrznego. Można używać następujących urządzeń logicznych:

SYSJPT — urządzenie wejściowe (zwykle czytnik kart),
SYSLST — urządzenie wyjściowe (zwykle drukarka),
SYSPCH — urządzenie wyjściowe (zwykle perforator kart),
SYS004 — **SYS***yyy* — urządzenia logiczne programisty (*yyy* oznacza najwyższy numer urządzenia i zależy od konkretnego wariantu systemu operacyjnego).

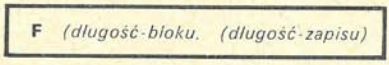
2. Postać zapisów na nośniku danych. Zapisy mogą występować w następujących czterech formatach.

a) zapisy nieblokowane stałej długości:



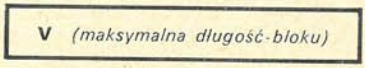
Format ten jest dostępny dla wszystkich kartotek. Natomiast następujące rodzaje kartotek mogą mieć wyłącznie postać:
— kartoteki z atrybutem **STREAM**,
— kartoteki losowej typu **REGIONAL**,
— kartoteki z atrybutem **RECORD**, pracującej z drukarką lub z kartami perforowanymi.

b) zapisy zblokowane stałej długości:



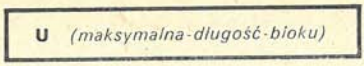
Stosunek długości bloku do długości zapisu (długości te określa w bajtach) musi być liczbą całkowitą. Blokowane zapisy stałej długości mogą występować wyłącznie w kartotekach z atrybutem **RECORD**, na taśmie magnetycznej lub dysku. Powinna to być kartoteka sekwencyjna lub indeksowo-sekwencyjna z atrybutem **BUFFERED**.

c) zapisy blokowane zmiennej długości:



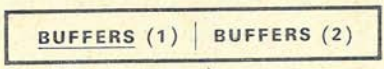
Przetwarzanie zapisów zmiennej długości wymaga ulokowania przed każdym zapisem czterobajtowego pola zawierającego długość zapisu, a na początku bloku — pola zawierającego długość bloku. Format ten może być zastosowany tylko w kartotekach sekwencyjnych na taśmach magnetycznych i dyskach.

d) zapisy nieblokowane zmiennej długości:



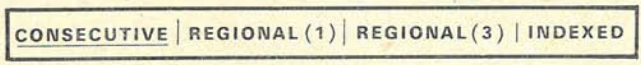
Każdy blok zawiera jeden zapis, którego długość podaje się w nawiasie.

3. Ilość buforów:



W atrybucie tym podaje się ilość używanych buforów (obszarów we/wy), niezależnie od atrybutu **BUFFERED**.

4. Metoda organizacji kartotek:



Poszczególne parametry oznaczają:

CONSECUTIVE — kartoteka sekwencyjna,

REGIONAL(1) — kartoteka losowa, w której każdy obszar pamięci może zawierać tylko jeden zapis,

REGIONAL(3) — kartoteka losowa, w której ustalone obszary zawierają dowolną ilość zapisów,

INDEXED — kartoteka indeksowo-sekwencyjna.

5. Warunki przetwarzania kartotek na taśmie magnetycznej.

Zwykle po zakończeniu przetwarzania kartoteki wskutek instrukcji **CLOSE** następuje automatyczne przewijanie taśmy do pozycji wyjściowej. Parametr **LEAVE** powoduje pominięcie przewijania.

6. Warunek sprawdzania kartoteki na dyskach:

Parametr ten powoduje kontrolę danych na dysku przy zapisywaniu. Kontrola polega na odczytaniu wprowadzonego zapisu i porównaniu danych.

7. Warunki przetwarzania kartotek o bezpośrednim dostępie.

KEYLENGTH *(n)*

n — liczba całkowita

Parametr **KEYLENGHT** określa długość klucza w kartotekach typu **REGIONAL(3)** i **INDEXED:**

$1 \leqslant n \leqslant 255$ — dla kartotek indeksowo-sekwencyjnych,

$9 \leqslant n \leqslant 255$ — dla kartotek REGIONAL(3).

EXTENTNUMBER *(n)*

Za pomocą tego parametru określa się liczbę segmentów pamięci dyskowej ($2 \leqslant n \leqslant 255$). Dla kartotek indeksowo-sekwencyjnych parametr ten musi być koniecznie ukazany. W innych wypadkach, gdy $n \leqslant 4$, można go pominąć.

8. Specjalne warunki dla kartotek indeksowo-sekwencyjnych:

INDEXMULTIPLE

Parametr ten używany jest dla bardzo dużych zbiorów, gdy korzysta się z indeksu głównego.

OFLTRACKS *(n)*

Wskazuje, ile ścieżek na każdym cylindrze ma być zajętych przez obszar nadmiaru $0 \leqslant n \leqslant 8$.

KEYLOC *(n)*

Parametr ten wskazuje położenie klucza wewnątrz zapisu.

Zamieszczamy kilka przykładowych opisów kartotek.

PRZYKŁAD 2

Kartoteka o nazwie **ALFA** zapisana jest na kartach i ma być wczytywana do pamięci głównej zapisami. Używa się przy tym jednego obszaru wejścia jako bufora.

Kartotekę tę należy opisać w następujący sposób:

```
DECLARE ALFA FILE INPUT RECORD SEQUENTIAL BUFFERED
        ENVIRONMENT (MEDIUM (SYSIPT, 6Ø12) F(8Ø)
        CONSECUTIVE);
```

Niektóre parametry przyjmowane są przez translator automatycznie, można więc to samo zdanie napisać krócej:

```
DECLARE ALFA FILE INPUT RECORD
        ENVIRONMENT (MEDIUM (SYSIPT, 6Ø12) F(8Ø));
```

PRZYKŁAD 3

Kartoteka **BETA** ma być utworzona na taśmie magnetycznej. Długość zapisu wynosi 80 bajtów, długość bloku 800 bajtów. Etykiety nie będą przetwarzane.

```
DECLARE BETA FILE OUTPUT RECORD
        ENVIRONMENT (MEDIUM (SYSØØ 5, 5Ø12) F(8ØØ,
        8Ø) NOLABEL);
```

PRZYKŁAD 4

Opisać kartotekę **GAMMA**, przeznaczoną do druku. Długość każdego zapisu wynosi 120 bajtów.

```
DECLARE GAMMA FILE PRINT ENVIRONMENT (F(12Ø)
        MEDIUM (SYSLST, 7Ø3Ø));
```

Ponieważ użyliśmy parametru **PRINT**, można było opuścić parametry **STREAM** i **OUTPUT**.

PRZYKŁAD 5

W programie wczytuje się kartotekę **DELTA** z czytnika kart do pamięci głównej. Po przetworzeniu kolejnych zapisów są one wprowadzane do kartoteki **KAPPA** typu **REGIONAL(1)** na dysku SYSØØ7.

W tym wypadku kartoteki **DELTA** i **KAPPA** opiszemy zdaniem:

```
DECLARE DELTA FILE RECORD INPUT
                ENVIRONMENT (F(8Ø) MEDIUM (SYSIPT,
                6Ø12),
        KAPPA FILE RECORD OUTPUT DIRECT KEYED
                ENVIRONMENT (F(8Ø) MEDIUM (SYSØØ7,
                5Ø56) REGIONAL(1));
```

PRZYKŁAD 6

Kartoteka indeksowo-sekwencyjna **LAMBDA** zapisana jest na dysku. W programie zapisy będą pobierane z tej kartoteki w celu aktualizacji i powrotnie zapisywane na uprzednie miejsce. Kartotekę taką należy opisać w następujący sposób:

```
DECLARE LAMBDA FILE RECORD UPDATE DIRECT KEYED
        ENVIRONMENT (F(500) MEDIUM (SYS031, 5056)
        INDEXED
        KEYLENGTH(8) EXTENTNUMBER(2) OFLTRACKS(2));
```

W tym wypadku przyjęliśmy, że zapisy kartoteki mają długość 500 bajtów każdy, w zapisie znajduje się klucz o długości ośmiu bajtów, kartoteka umieszczona jest na dysku o nazwie SYS031, w dwóch segmentach pamięci dyskowej, a w każdym cylindrze wydzielono dwie ścieżki jako obszar nadmiaru.

Otwieranie i zamykanie kartotek

Przed przetwarzaniem kartotek należy przygotować do pracy nośniki tych zbiorów danych i urządzenia zewnętrzne. Gdy odpowiednie nośniki znajdują się we właściwych urządzeniach, wówczas system wykonuje pewne czynności pomocnicze. Na przykład jeśli chodzi o taśmy magnetyczne, podprowadza początek kartoteki pod głowice, przetwarza informację zawartą w etykietach. Wszystkie te i podobne działania wykonywane są po napotkaniu w programie instrukcji **OPEN**:

```
OPEN FILE (nazwa-kartoteki) [, FILE (nazwa-kartoteki) ] ... ;
```

Czynności związane z zamykaniem kartotek, takie jak przewinięcie taśmy magnetycznej, zapis znaczników itd., wykonywane są wskutek instrukcji **CLOSE**:

```
CLOSE FILE (nazwa-kartoteki) [, FILE (nazwa-kartoteki) ] ... ;
```

b. Przetwarzanie kartotek

Przy przetwarzaniu kartotek z atrybutem **RECORD**, a tylko takimi kartotekami zajmujemy się w tym rozdziale, każda instrukcja we/wy operuje całym zapisem jednocześnie, nie dokonując w zapisie żadnych zmian. Stosuje się w tym wypadku instrukcje **WRITE** i **READ**.

Przy czytaniu zapisów kartoteki sekwencyjnej używa się instrukcji **READ** w postaci:

READ FILE *(nazwa-kartoteki)* **INTO** *(identyfikator)* ;

W celu wprowadzenia zapisów do kartoteki używa się instrukcji **WRITE:**

WRITE FILE *(nazwa-kartoteki)* **FROM** *(identyfikator)* ;

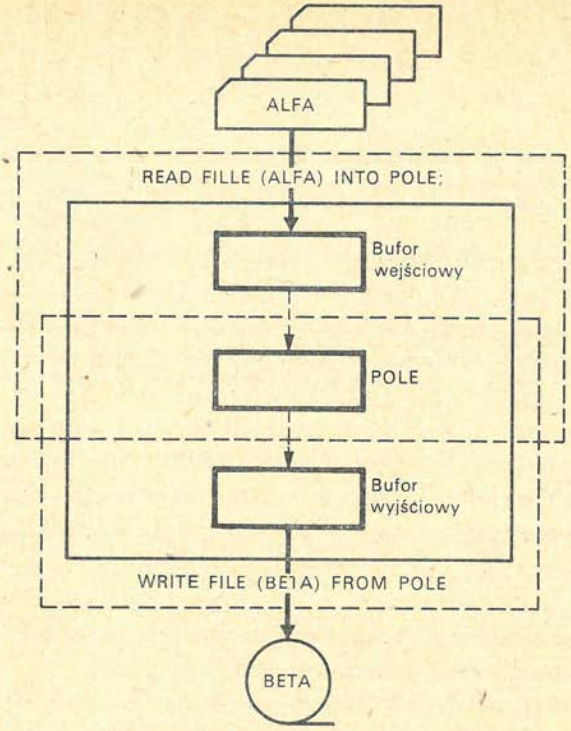
Identyfikatory w tych formatach oznaczają nazwę obszaru pamięci, którego zawartość powinna być przeczytana **(READ)** lub zapisana **(WRITE)** w kartotece. Obszar ten może być zadeklarowany jako zmienna skalarowa, tablica lub struktura.

PRZYKŁAD 7

Przepisać kartotekę ALFA, zapisaną na kartach, na taśmę magnetyczną, tworząc kartotekę BETA (por. rys. 66).

```
PROGR: PROCEDURE OPTIONS (MAIN);
       DECLARE ALFA FILE RECORD INPUT
                    ENVIRONMENT (MEDIUM (SYSIPT,6Ø12)
                    F(8Ø));
       DECLARE BETA FILE RECORD OUTPUT
                    ENVIRONMENT (MEDIUM (SYSØØ5,5Ø12)
                    F(8ØØ.8Ø) NOLABEL);
       DECLARE POLE CHARACTER(8Ø);
       OPEN FILE (ALFA), FILE (BETA);
       CZYTAJ: READ FILE (ALFA) INTO (POLE);
               WRITE FILE (BETA) FROM (POLE);
               IF POLE = 'KONIEC' THEN GOTO ZAMYK;
               GOTO CZYTAJ;
       ZAMYK: CLOSE FILE (ALFA), FILE (BETA);
               END;
```

W programie tym przyjęto, że ostatnia karta w kartotece ALFA rozpoczyna się od tekstu KONIEC.



Rys. 66. Tworzenie kartoteki na taśmie magnetycznej

Kartoteki losowe

Przy przetwarzaniu kartotek losowych typu **REGIONAL(1)** i **REGIONAL(3)** używa się instrukcji **WRITE** i **READ** w następującej postaci:

WRITE FILE *(nazwa-kartoteki)* **FROM** *(identyfikator)* **KEYFROM** *(wyrażenie)*;
READ FILE *(nazwa-kartoteki)* **INTO** *(identyfikator)* **KEY** *(wyrażenie)* ;

W instrukcjach tych *identyfikator* określa najczęściej zmienną typu tekst znakowy, lecz może to być również nazwa struktury lub tablicy. *Wyrażenie* stojące za **KEYFROM** lub **KEY** jest skalarnym wyrażeniem, które zostaje zamienione na tekst znakowy. Za pomocą instrukcji **WRITE** zapis z pola

pamięci określonego identyfikatorem zostaje przeniesiony do pamięci dyskowej zgodnie z ukazanym w wyrażeniu kluczem. Instrukcja **READ** powoduje czytanie z pamięci dyskowej zapisu określonego kluczem i wprowadzenie go do pola pamięci o nazwie podanej w instrukcji.

PRZYKŁAD 8

Utworzyć kartotekę **OMEGA** typu **REGIONAL(1)** na dysku. Dane należy brać z kartoteki kartowej o nazwie **KARTY**. Kartoteka **KARTY** jest spisem studentów, w którym każdy zapis rozpoczyna czterocyfrowy numer ewidencyjny. Należy zapewnić w programie dostęp do kartoteki **OMEGA** przez podanie numeru ewidencyjnego. Ostatnia pozycja ma numer ewidencyjny 9999.

```
H_8: PROCEDURE OPTIONS (MAIN);
     DECLARE KARTY FILE RECORD INPUT
                ENVIRONMENT (F(8Ø) MEDIUM (SYSIPT,
                6Ø12));
  DECLARE OMEGA FILE RECORD OUTPUT DIRECT KEYED
          ENVIRONMENT (F(8Ø) REGIONAL(1) MEDIUM
          (SYSØØ6,5Ø56));
  DECLARE BUF CHARACTER(8Ø),
          1 Q DEFINED BUF,
            2 NR_EWID PICTURE '(4)9',
            2 DANE CHARACTER (76),
          KLUCZ PICTURE '(8)9';
  OPEN FILE (KARTY), FILE (OMEGA);
  CZYT: READ FILE (KARTY) INTO (BUF);
        IF NR_EWID = 9999 THEN GOTO ZAM;
        KLUCZ = NR_EWID;
        WRITE FILE (OMEGA) FROM (BUF) KEYFROM
        (KLUCZ);
        GOTO CZYT;
  ZAM: CLOSE FILE (KARTY), FILE (OMEGA);
       END;
```

Kartoteki indeksowo-sekwencyjne

W kartotekach indeksowo-sekwencyjnych używa się takich samych instrukcji, jak w zbiorach losowych. Dodatkowo można użyć instrukcji **READ** w następującej postaci:

```
READ FILE (nazwa-kartoteki) INTO (identyfikator)
                            KEYTO (identyfikator) ;
```

Format taki stosuje się dla czytania kartoteki indeksowo-sekwencyjnej w sposób sekwencyjny. Identyfikator po **KEYTO** podaje nazwę pierwszego klucza sekwencji.

PRZYKŁAD 9

W programie należy przeczytać niektóre zapisy z kartoteki indeksowo-sekwencyjnej o nazwie **PSI** i wydrukować je. Wybrane zapisy wskazuje klucz, który jest pierwszą pięcioznakową częścią każdego zapisu. Zapisy mają długość 100 bajtów.

```
PRZH_9: PROCEDURE OPTIONS (MAIN);
        DECLARE PSI FILE RECORD INPUT DIRECT KEYED
                        ENVIRONMENT (F(1ØØ) INDEXED
                        KEYLENGTH(5)
                        OFLTRACKS(2) EXTENTNUMBER(2)
                        MEDIUM (SYSØØ7,5Ø56));
        DECLARE 1 OBSZ,
                        2 NUMER PICTURE '(5)9',
                        2 DANE CHARACTER (95);
        OPEN FILE (PSI);
        DO K = 5Ø BY 5 TO 9Ø;
        READ FILE (PSI) INTO (OBSZ) KEY (K);
        PUT EDIT (OBSZ) (SKIP, A);
        END;
END;
```

4. Etykiety kartotek

DOS JS zapewnia identyfikację i ochronę wszystkich kartotek na dyskach i taśmach magnetycznych. Możliwe to jest dzięki specjalnym zapisom, dołączonym do kartotek, zwanym etykietami[5]. W rozdziale tym omówimy etykiety standardowe, chociaż system umożliwia użytkownikowi tworzenie również własnych, niestandardowych etykiet.

Etykietą standardową rozpoczyna się każdy wolumen (przez wolumen będziemy rozumieli jednostkę fizyczną pamięci, np. pakiet dysków lub

[5] Por. DOS/JS *Etykiety dyskowe*; DOS/JS *Etykiety taśmowe*.

krążek taśmy magnetycznej). Jeden wolumen może zawierać kilka kartotek, z których każda ma swoje etykiety. Możliwa jest również sytuacja przeciwna — jedna kartoteka może zajmować kilka woluminów.

a. Etykiety dysków

Każdy pakiet dysków powinien mieć swoją etykietę woluminu. Jest to zawsze trzeci zapis na ścieżce zerowej zerowego cylindra. Etykieta woluminu zawiera takie informacje, jak: numer rejestracyjny pakietu, adres pola zawierającego skorowidz woluminu, dziesięciobajtową nazwę użytkownika. Etykiety woluminów zapisuje standardowy program INICJALIZACJI DYSKÓW. W czasie przetwarzania kartotek etykiety woluminów są sprawdzane, lecz nie mogą być zmieniane.

Etykiety standardowe kartotek pozwalają zidentyfikować kartotekę, określić jej położenie w woluminie i ochraniają kartotekę przed zniszczeniem. Wszystkie etykiety kartotek danego woluminu zgrupowane są w jednym obszarze pamięci dyskowej, zwanym skorowidzem woluminu (w skrócie VTOC). Skorowidz ma z kolei swoją etykietę.

Wszystkie rodzaje kartotek używają etykiet formatu 1. Informacje zawarte w tej etykiecie opisuje tablica 28. Oprócz tego stosuje się jeszcze etykiety innych formatów. Dla przykładu, etykiety formatu 2 stosuje się w kartotekach indeksowo-sekwencyjnych. Gdy kartoteka zajmuje więcej niż trzy segmenty pamięci, wówczas opis kartoteki kontynuowany jest w etykiecie formatu 3.

Informacje o etykietach są przetwarzane przez programy systemowe. Przetwarzanie polega na porównaniu informacji zawartych w etykietach z informacjami dostarczonymi przez program problemowy w operatorach sterujących **DLBL** i **EXTENT**. Operator **DLBL** ma następujący format:

```
// DLBL nazwa-kartoteki [, identyfikator ] [,data ][,kody ]
```

W tym parametrze, oprócz nazwy kartoteki, można podać:

— *identyfikator*; unikalna nazwa kartoteki (do 44 znaków ujętych w apostrofy), do której użytkownik może włączyć numer generacji i numer wersji,

— *data;* data ważności, zadana w postaci *nnnn* (liczba dni ważności) lub w postaci *rrddd* (dwie ostatnie cyfry roku i liczba dni od początku roku),
— *kody*; dwa lub trzy znaki określające typ kartoteki:

SD — organizacja sekwencyjna,
DA — organizacja z bezpośrednim dostępem,
ISC lub **ISE** — organizacja indeksowo-sekwencyjna.

TABLICA 28

Standardowa etykieta dysków (format **1**)

Bajty	Przeznaczenie
1—44	Indentyfikator kartoteki. Zawiera alfanumeryczną nazwę kartoteki.
45	Format etykiety, w tym wypadku 1.
46—51	Numer rejestracyjny kartoteki.
52—53	Numer porządkowy woluminu (jeśli kartoteka mieści się na kilku woluminach, są one kolejno numerowane 0,1,2,...).
54—56	Data utworzenia kartoteki w postaci rdd, gdzie r—rok (od 0 do 99), a dd—dzień roku (od 1 do 366).
57—59	Data ważności kartoteki, w postaci jw.
60	Licznik ilości segmentów pamięci zajmowanych przez kartotekę.
61—62	Nie wykorzystuje się.
63—75	Kod systemu programowania.
76—82	Nie wykorzystuje się.
83—84	Typ kartoteki (2000 kartoteka z bezpośrednim odstępem, 4000 organizacja sekwencyjna, 8000 organizacja indeksowo-sekwencyjna, 0000 organizacja nieokreślona).
85	Nie wykorzystuje się.
86	Kod trybu pracy podczas zakładania kartoteki (wskazuje obecność indeksu głównego, obszarów przepełnienia itp.).
87—88	Długość bloku dla zapisów stałej długości lub maksymalny rozmiar bloku dla zapisów zmiennej długości.
89—90	Długość zapisu.
91	Długość klucza.
92—93	Położenie klucza w zapisie.
94	Wskaźnik ostatniego woluminu kartoteki.
95—105	Nie wykorzystuje się.
106—115	Opis pierwszego segmentu pamięci. Zawiera takie informacje, jak typ segmentu, dolną granicę, najmniejszy adres, górną granicę itp.
116—125	Opis dodatkowego segmentu.
126—135	Opis dodatkowego segmentu.
136—140	Zera lub adres następnej etykiety, jeśli jest potrzebna dla dalszego opisu kartoteki, tzn. gdy kartoteka zajmuje więcej niż trzy segmenty.

Segmenty dysku, na których zapisuje się kartotekę, opisuje się za pomocą operatora **EXTENT**. Za każdym operatorem **DLBL** powinien następować przynajmniej jeden operator **EXTENT**. Operator ten ma następującą postać:

```
// EXTENT [SYSxxx] [,numer-rejestracyjny] [,typ]
       [,numer-segmentu] [,adres-segmentu] [,liczba-ścieżek]
```

Wszystkie parametry są nieobowiązkowe i mają następujące znaczenie:
SYS*xxx* — nazwa urządzenia logicznego, któremu przypisano dysk:
numer-rejestracyjny — od jednego do sześciu znaków oznaczających numer pakietu, na którym znajduje się opisywany segment pamięci;
typ — cyfra, oznaczająca typ segmentu:
1 — segment przeznaczony dla danych (cylindry nierozdzielone),
2 — segment dla obszaru przepełnienia (dla kartotek indeksowo-sekwencyjnych),
4 — segment dla indeksu (kartoteka indeksowo-sekwencyjna),
8 — segment dla danych (rozdzielone cylindry);
numer-segmentu — liczba dziesiętna od 0 do 255, oznaczająca numer porządkowy kartoteki, składającej się z wielu segmentów. Numeracja segmentów w kartotekach sekwencyjnych i losowych powinna zaczynać się od zera. W kartotekach indeksowo-sekwencyjnych, jeśli używa się indeksu głównego, pierwszy segment powinien mieć numer zero, w przeciwnym wypadku numeracja powinna rozpocząć się od 1;
adres segmentu — od jednej do pięciu cyfr dziesiętnych, określających względny adres pierwszej ścieżki segmentu:
adres-segmentu = 10* *nr cylindra*+*nr ścieżki cylindra*;
liczba-ścieżek — od jednej do pięciu cyfr wskazujących liczbę ścieżek segmentu.

b. Etykiety taśm magnetycznych

Podobnie jak dyski, taśmy magnetyczne mogą mieć etykiety standardowe i niestandardowe, które tutaj również pominiemy. Jednakże, w odróżnieniu od dysków, kartoteki na taśmach magnetycznych mogą być przetwarzane bez etykiet.

Każdy krążek taśmy (wolumen) zaczyna się etykietą woluminu o nazwie VOL1. Jest to zapis 80-bajtowy, którego cztery pierwsze bajty zawierają identyfikator etykiety VOL1. Standardowa etykieta woluminu zawiera numer rejestracyjny woluminu i nazwę użytkownika. Etykietę woluminu zapisuje się na taśmie za pomocą specjalnych programów systemowych.

Etykiety kartotek zapisuje się przed i po każdej kartotece. W związku z tym nazywają się etykietami początku i końca kartoteki. Zawartość tych etykiet przedstawia tablica 29.

TABLICA 29

Etykieta standarowa kartoteki na taśmie magnetycznej

Bajty	Zawartość i przeznaczenie
1—3	Identyfikator etykiety: HDR — początek kartoteki, EOF — koniec kartoteki, EOV — koniec woluminu.
4	Numer etykiety równa się 1.
5—21	Identyfikator kartoteki. Zawiera nazwę unikalną kartoteki.
22—27	Numer rejestracyjny kartoteki.
28—31	Numer porządkowy woluminu. Jeśli kartoteka zajmuje kilka krążków, kolejne woluminy są numerowane od 0001 do 9999.
32—35	Numer porządkowy kartoteki, jeżeli na woluminie mieści się kilka kartotek.
36—39	Numer generacji kartoteki.
40—41	Numer wersji generacji.
42—47	Data utworzenia kartoteki w postaci rrddd, gdzie rr—rok, ddd—dzień w roku.
48—53	Data ważności kartoteki, po minięciu której kartoteka może być zniszczona. Postać daty—jw.
54	Bajt ochrona kartotek: 0 — bez ochrony, 1 — z ochroną.
55—60	Licznik bloków. Wskazuje ilość zapisanych bloków od ostatniej etykiety początku. Wykorzystuje się tylko dla etykiet końca kartoteki.
61—73	Identyfikacja systemu programowania.
74—80	Nie wykorzystuje się.

Etykiety na taśmie są przetwarzane przez specjalne programy SSWW. Przetwarzanie sprowadza się do porównania informacji zawartych w etykietach z danymi zawartymi w operatorze sterującym TLBL. Operator ten ma następujący format:

```
// TLBL nazwa-kartoteki [,'identyfikator'] [,data] [,nr-rejestracyjny]
     [,nr-porządkowy-woluminu] [,nr-porządkowy-kartoteki]
     [,nr-generacji] [,nr-wersji-generacji]
```

W operatorze tym obowiązkowym parametrem jest tylko nazwa kartoteki, pozostałe argumenty występują w zależności od uznania programisty. Mają one następujące znaczenie:

identyfikator — do 17 znaków alfanumerycznych ujętych w apostrofy. Identyfikator można traktować jako pełną nazwę kartoteki. Jeśli jest opuszczony, wówczas system przyjmuje jako identyfikator nazwę kartoteki;

data — data ważności kartoteki; może być podana w postaci *nnnn* (liczba dni ważności) lub w postaci *rrddd* (dwie ostatnie cyfry roku i liczba dni od początku roku);

nr-rejestracyjny-kartoteki — od jednego do sześciu znaków alfanumerycznych; jeżeli parametr ten jest opuszczony, system nie prowadzi kontroli numerów rejestracyjnych kartotek;

nr-porządkowy-woluminu — od jednej do czterech cyfr, oznaczających kolejny numer woluminu kartoteki zajmującej więcej niż jeden krążek taśmy.

nr-generacji — od jednej do czterech cyfr uzupełniających identyfikator kartoteki. Jeśli parametr ten jest opuszczony, wówczas system przyjmuje jego wartość równą 0001, natomiast dla kartoteki wejściowej kontroli nie przeprowadza się;

nr-wersji-generacji — jedna lub dwie cyfry modyfikujące numer generacji.

W wypadku przetwarzania kartotek sekwencyjnych na taśmach magnetycznych lub kartotek niesekwencyjnych na dyskach konieczny jest operator **LBLTYP**, którego zadaniem jest określenie obszaru pamięci dla przetwarzania etykiet. Operator ten ma następujący format:

```
// LBLTYP { TAPE
         { NSD(nn) }
```

Parametr **TAPE** podaje się dla kartotek sekwencyjnych na taśmie magnetycznej, natomiast parametr **NSD**(*nn*) w wypadku kartotek na urządzeniach z bezpośrednim dostępem (*nn* oznacza liczbę segmentów zajmowanych przez kartotekę).

Aneks

ZAŁĄCZNIK I

Lista rozkazów maszyn cyfrowych Jednolitego Systemu

Symbol	Nazwa rozkazu	Typ	Kod	Format
AR	Dodawanie *Add*	RR	1A	R1,R2
A	Dodawanie *Add*	RX	5A	R1,D2(X2,B2)
AH	Dodawanie półsłowa *Add Halfword*	RX	4A	R1,D2(X2,B2)
ALR	Dodawanie kodów *Add Logical*	RR	1E	R1,R2
AL	Dodawanie kodów *Add Logical*	RX	5E	R1,D2(X2,B2)
NR	Koniunkcja *AND*	RR	14	R1,R2
N	Koniunkcja *AND*	RX	54	R1,D2(X2,B2)
NI	Koniunkcja *AND*	SI	94	D1,B1,I2
NC	Koniunkcja *AND*	SS	D4	D1(L,B1),D2(B2)
BALR	Skok ze śladem *Branch and Link*	RR	05	R1,R2
BAL	Skok ze śladem *Branch and Link*	RX	45	R1,D2(X2,B2)
BCR	Skok warunkowy *Branch on Condition*	RR	07	M1,R2
BC	Skok warunkowy *Branch on Condition*	RX	47	M1,D2(X2,B2)
BCTR	Skok według licznika *Branch on Count*	RR	06	R1,R2
BCT	Skok według licznika *Branch on Count*	RX	46	R1,R2(X2,B2)

ZAŁĄCZNIK I (cd.)

Symbol	Nazwa rozkazu	Typ	Kod	Format
BXH	Skok gdy indeks większy *Branch on Index High*	RS	86	R1,R3,D2(B2)
BXLE	Skok gdy indeks mniejszy jub równy *Branch on Index Low or Equal*	RS	87	R1,R3,D2(B2)
CR	Porównanie *Compare*	RR	19	R1,R2
C	Porównanie *Compare*	RX	59	R1,D2(X2,B2)
CH	Porównanie półsłów *Compare Halfword*	RX	49	R1,D2(X2,B2)
CLR	Porównanie kodów *Compare Logical*	RR	15	R1,R2
CL	Porównanie kodów Porównanie kodów	RX	55	R1,D2(X2,B2)
CLC	Porównanie znaków *Compare Logical*	SS	D5	D1(L,B1),D2(B2)
CLI	Porównanie bezpośrednie *Compare Logical*	SI	95	D1(B1),I2
CVB	Konwersja binarna *Convert to Binary*	RX	4F	R1,D2(X2,B2)
CVD	Konwersja dziesiętna *Convert to Decimal*	RX	4E	R1,D2(X2,B2)
DR	Dzielenie *Divide*	RR	1D	R1,R2
D	Dzielenie *Divide*	RX	5D	R1,D2(X2,B2)
XR	Suma modulo 2 *Exclusive OR*	RR	17	R1,R2
X	Suma modulo 2 *Exclusive OR*	RX	57	R1,D2(X2,B2)
XI	Suma modulo 2 bezpośrednia *Exclusive OR*	SI	97	D1(B1), I2
XC	Suma modulo 2 znaków *Exclusive OR*	SS	D7	D1(L,B1),D2(B2)
EX	Wykonaj *Execute*	RX	44	R1,D2(X2,B2)
HIO	Zatrzymać we/wy *Halt I/O*	SI	9E	D1(B1)
IC	Wstawić znak *Insert Character*	RX	43	R1,D2(X2,B2)
L	Ładowanie Load	RX	58	R1,D2(X2.B2)

Symbol	Nazwa rozkazu	Typ	Kod	Format
LR	Ładowanie *Load*	RR	18	R1,R2
LA	Ładowanie adresu *Load Address*	RX	41	R1,D2(X2,B2)
LTR	Ładowanie z testowaniem *Load and Test*	RR	12	R1,R2
LCR	Ładowanie uzupełnienia *Load Complement*	RR	13	R1,R2
LH	Ładowanie półsłowa *Load Halfword*	RX	48	R1,D2(X2,B2)
LM	Ładowanie grupowe *Load Multiple*	RX	98	R1,R3,D2(B2)
LNR	Ładowanie negatywne *Load Negative*	RR	11	R1,R2
LPR	Ładowanie pozytywne *Load Positive*	RR	10	R1,R2
LPSW	Ładowanie PSW *Load PSW*	SI	82	D1(B1)
MVI	Przesyłka bezpośrednia *Move*	SI	92	D1(B1),I2
MVC	Przesyłka znaków *Move*	SS	D2	D1(L,B1),D2(B2)
MVN	Przesyłanie cyfr *Move Numerics*	SS	D1	D1(L,B1),D2(B2)
MVO	Przesyłanie z przesunięciem *Move with Offset*	SS	F1	D1(L1,B1),D2(L2,B2)
MVZ	Przesyłanie stref *Move Zones*	MVZ	D3	D1(L,B1),D2(B2)
MR	Mnożenie *Multiply*	RR	1C	R1,R2
M	Mnożenie *Multiply*	RX	5C	R1,D2(X2,B2)
MH	Mnożenie półsłów *Multiply Halfword*	RX	4C	R1,D2(X2,B2)
OR	Alternatywa *OR*	RR	16	R1,R2
O	Alternatywa *OR*	RX	56	R1,D2(X2,B2)
OI	Alternatywa bezpośrednia *OR*	SI	96	D1(B1),I2

Symbol	Nazwa rozkazu	Typ	Kod	Format
OC	Alternatywa znaków *OR*	SS	D6	D1(L,B1),D2(B2)
PACK	Pakowanie *Pack*	SS	F2	D1(L1,B1),D2(L2,B2)
SPM	Testuj maskę programu *Set Program Mask*	RR	04	R1
SSM	Testuj maskę systemu *Set Systems Mask*	SI	80	D1(B1)
SLDA	Przesunięcie w lewo podwójne *Shift Left Double*	RS	81	R1,D2(B2)
SLA	Przesunięcie w lewo *Shift Left Single*	RS	8B	R1,D2(B2)
SLDL	Przesunięcie w lewo podwójne logiczne *Shift Left Double Logical*	RS	8D	R1,D2(B2)
SLL	Przesunięcie w lewo logiczne *Shift Left Single Logical*	RS	89	R1,D2(B2)
SRDA	Przesunięcie w prawo podwójne arytmetyczne *Shift Right Double*	RS	8E	R1,D2(B2)
SRA	Przesunięcie w prawo *Shift Right Single*	RS	8A	R1,D2(B2)
SRDL	Przesunięcie w prawo podwójne logiczne *Shift Right Double Logical*	RS	8C	R1,D2(B2)
SRL	Przesunięcie w prawo logiczne *Shift Right Single Logical*	RS	88	R1,D2(B2)
SIO	Start we/wy *Start I/O*	SI	9C	D1(B1)
ST	Zapamiętaj *Store*	RX	50	R1,D2(X2,B2)
STC	Zapamiętaj znaki *Store Character*	RX	42	R1,D2(X2,B2)
STH	Zapamiętaj półsłowo *Store Halfword*	RX	40	R1,D2(X2,B2)
STM	Zapamiętanie grupowe *Store Multiple*	RS	90	R1,R3,D2(B2)
SR	Odejmowanie *Subtract*	RR	1B	R1,R2
S	Odejmowanie *Subtract*	RX	5B	R1,D2(X2,B2)

Symbol	Nazwa rozkazu	Typ	Kod	Format
SH	Odejmowanie półsłowa *Subtract Halfword*	RX	4B	R1,D2(X2,B2)
SLR	Odejmowanie kodów *Subtract Logical*	RR	1F	R1,R2
SL	Odejmowanie kodów *Subtract Logical*	RX	5F	R1,D2(X2,B2)
SVC	Przywołanie SUPERVISORA Supervisor Call	RR	0A	I
TS	Testowanie i ustawienie *Test and Set*	SI	93	D1(B1)
TCH	Testowanie kanału *Test Channel*	SI	9F	D1(B1)
TIO	Testowanie we/wy *Test I/O*	SI	9D	D1(B1)
TM	Testowanie z maską *Test Under Mask*	SI	91	D1(B1),I2
TR	Przekodowanie *Translate*	SS	DC	D1(L,B1),D2(B2)
TRT	Przekodowanie i testowanie *Translate and Test*	SS	DD	D1(L,B1),D2(B2)
UNPK	Rozpakowanie *Unpack*	SS	F3	D1(L1,B1),D2(L2,B2)
	Instrukcje dziesiętne			
AP	Dodawanie dziesiętne *Add Decimal*	SS	FA	D1(L1,B1),D2(L2,B2)
CP	Porównanie dziesiętne *Compare Decimal*	SS	F9	D1(L1,B1),D2(L2,B2)
DP	Dzielenie dziesiętne *Divide Decimal*	SS	FD	D1(L1,B1),D2(L2,B2)
ED	Redagowanie *Edit*	SS	DE	D1(L,B1),D2(B2)
EDMK	Redagowanie z zaznaczaniem *Edit and Mark*	SS	DF	D1(L,B1),D2(B2)
MP	Mnożenie dziesiętne *Multiply Decimal*	SS	FC	D1(L1,B1),D2(L2,B2)

Symbol	Nazwa rozkazu	Typ	Kod	Format
SP	Odejmowanie dziesiętne *Subtrack Decimal*	SS	FB	D1(L1,B1),D2(L2, B2)
ZAP	Zerowanie z dodawaniem *Zero and Add*	SS	F8	D1(L1,B1),D2(L2, B2)
	Instrukcje bezpośredniego sterowania			
RDD	Czytanie bezpośrednie *Read Direct*	SI	85	D1(B1),I2
WRD	Zapisywanie bezpośrednie *Write Direct*	SI	84	D1(B1),I2
	Instrukcje ochrony pamięci			
ISK	Czytaj klucz ochrony *Insert Storage Key*	RR	09	R1,R2
AUR	Dodawanie bez normalizacji -krótkie *Add Unnormalized (Short)*	RR	3E	R1,R2
AU	Dodawanie bez normalizacji -krótkie *Add Unnormalized (Short)*	RX	7E	R1,D2(X2,B2)
CDR	Porównanie-długie *Compare (Long)*	RR	29	R1,R2
CD	Porównanie-długie *Compare (Long)*	RX	69	R1,D2(X2,B2)
CER	Porównanie-krótkie *Compare (Short)*	RR	39	R1,R2
CE	Porównanie-krótkie *Compare (Short)*	RX	79	R1,D2(X2,B2)
DDR	Dzielenie-długie *Divide (Long)*	RR	2D	R1,R2
DD	Dzielenie-długie *Divide (Long)*	RX	6D	R1,D2(X2,B2)
DER	Dzielenie-krótkie *Divide (Short)*	RR	3D	R1,R2
DE	Dzielenie-krótkie *Divide (Short)*	RX	7D	R1,D2(X2,B2)
HDR	Połowienie długie *Halve (Long)*	RR	24	R1,R2

Symbol	Nazwa rozkazu	Typ	Kod	Format
SSK	Ustawienie klucza ochrony *Set Storage Key*	RR	08	R1,R2
HER	Połowienie krótkie *Halve*	RR	34	R1,R2
LTDR	Ładuj i testuj-długie *Load and Test (Long)*	RR	22	R1,R2
LTER	Ładuj i testuj-krótkie *Load and Test (Short)*	RR	32	R1,R2
LCDR	Ładowanie uzupełnienia-długie *Load Complement (Long)*	RR	23	R1,R2
LCER	Ładowanie uzupełnienia-krótkie *Load Complement (Short)*	RR	33	R1,R2
LDR	Ładowanie-długie *Load (Long)*	RR	28	R1,R2
LD	Ładowanie-długie *Load (Long)*	RX	68	R1,D2(X2,B2)
LNDR	Ładowanie negatywne-długie *Load Negative (Long)*	RR	21	R1,R2
LNER	Ładowanie negatywne-krótkie *Load Negative (Short)*	RR	31	R1,R2
LPDR	Ładowanie pozytywne-długie *Load Positive (Long)*	RR	20	R1,R2

Instrukcje zmiennoprzecinkowe

ADR	Dodawanie z normalizacją -długie *Add Normalized (Long)*	RR	2A	R1,R2
AD	Dodawanie z normalizacją -długie	RX	6A	R1,D2(X2,B2)
AER	Dodawanie z normalizacją -krótkie *Add Normalized (Short)*	RR	3A	R1,R2
AE	Dodawanie z normalizacją -krótkie *Add Normalized (Short)*	RX	7A	R1,D1(X2,B2)
AWR	Dodawanie bez normalizacji -długie *Add Unnormalized (Long)*	RR	2E	R1,R2
AW	Dodawanie bez normalizacji -długie *Add Unnormalized (Long)*	RX	6E	R1,D2(X2,B2)

Symbol	Nazwa rozkazu	Typ	Kod	Format
LPER	Ładowanie pozytywne-krótkie *Load Positive (Short)*	RR	30	R1,R2
LER	Ładowanie-krótkie *Load (Short)*	RR	38	R1,R2
LE	Ładowanie-krótkie *Load (Short)*	RX	78	R1,D2(X2,B2)
MDR	Mnożenie-długie *Multiply (Long)*	RR	2C	R1,R2
MD	Mnożenie-długie *Multiply (Long)*	RX	6C	R1,D2(X2,B2)
MER	Mnożenie-krótkie *Multiply (Short)*	RR	3C	R1,R2
ME	Mnożenie-krótkie *Multiply (Short)*	RX	7C	R1,D2(X2,B2)
STD	Zapamiętaj-długie *Store (Long)*	RX	60	R1,D2(X2,B2)
STE	Zapamiętaj-krótkie *Store (Short)*	RX	70	R1,D2(X2,B2)
SDR	Odejmowanie z normalizacją -długie *Subtract Normalized (Long)*	RR	2B	R1,R2
SD	Odejmowanie z normalizacją -długie *Subtract Normalized (Long)*	RX	6B	R1,D2(X2,B2)
SER	Odejmowanie z normalizacją -krótkie *Subtract Normalized (Short)*	RR	3B	R1,R2
SE	Odejmowanie z normalizacją -krótkie *Subtract Normalized (Short)*	RX	7B	R1,D2(X2,B2)
SWR	Odejmowanie bez normalizacji -długie *Subtract Unnormalized (Long)*	RR	2F	R1,R2
SW	Odejmowanie bez normalizacji -długie *Subtract Unnormalized (Long)*	RX	6F	R1,D2(X2,B2)
SUR	Odejmowanie bez normalizacji -krótkie *Subtract Unnormalized (Short)*	RR	3F	R1,R2
SU	Odejmowanie bez normalizacji -krótkie *Subtract Unnormalized (Short)*	RX	7F	R1,D2(X2,B2)

ZAŁĄCZNIK II

Kody używane w Jednolitym Systemie

Lp.	Kod szesnastkowy	Znak w EBCDIC	Kod na kartach perforowanych	Symbole rozkazów maszynowych
0	00	NUL	12-0-1-8-9	
1	01		12-1-9	
2	02		12-2-9	
3	03		12-3-9	
4	04		12-4-9	
5	05		12-5-9	BALR
6	06		12-6-9	BCTR
7	07		12-7-9	BCR
8	08		12-8-9	SSK
9	09		12-1-8-9	ISK
10	0A		12-2-8-9	SVC
11	0B		12-3-8-9	
12	0C		12-4-8-9	
13	0D		12-5-8-9	
14	0E		12-6-8-9	
15	0F		12-7-8-9	
16	10		12-11-1-8-9	LPR
17	11		11-1-9	LNR
18	12		11-2-9	LTR
19	13		11-3-9	LCR
20	14		11-4-9	NR
21	15		11-5-9	CLR
22	16		11-6-9	OR
23	17		11-7-9	XR
24	18		11-8-9	LR
25	19		11-1-8-9	CR
26	1A		11-2-8-9	AR
27	1B		11-3-8-9	SR
28	1C		11-4-8-9	MR
29	1D		11-5-8-9	DR
30	1E		11-6-8-9	ALR
31	1F		11-7-8-9	SLR
32	20		11-0-1-8-9	LPDR
33	21		0-1-9	LNDR
34	22		0-2-9	LTDR
35	23		0-3-9	LCDR
36	24		0-4-9	HDR

ZAŁĄCZNIK II (cd.)

Lp.	Kod szesnastkowy	Znak	Kod na kartach perforowanych	Symbole rozkazów maszynowych
37	25		0-5-9	LRDR
38	26		0-6-9	
39	27		0-7-9	
40	28		0-8-9	LDR
41	29		0-1-8-9	CDR
42	2A	<	0-2-8-9	ADR
43	2B		0-3-8-9	SDR
44	2C		0-4-8-9	MDR
45	2D		0-5-8-9	DDR
46	2E		0-6-8-9	AWR
47	2F		0-7-8-9	SWR
48	30		12-11-0-1-8-9	LPER
49	31		1-9	LNER
50	32		2-9	LTER
51	33		3-9	LCER
52	34		4-9	HER
53	35		5-9	LRER
54	36		6-9	AXR
55	37		7-9	SXR
56	38		8-9	LER
57	39		1-8-9	CER
58	3A		2-8-9	AER
59	3B		3-8-9	SER
60	3C		4-8-9	MER
61	3D		5-8-9	DER
62	3E		6-8-9	AUR
63	3F		7-8-9	SUR
64	40			STH
65	41		12-0-1-9	LA
66	42		12-0-2-9	STC
67	43		12-0-3-9	IC
68	44		12-0-4-9	EX
69	45		12-0-5-9	BAL
70	46		12-0-6-9	BCT
71	47		12-0-7-9	BC
72	48		12-0-8-9	LH
73	49		12-1-8	CH
74	4A	¢	12-2-8	AH
75	4B		12-3-8	SH

Lp.	Kod szesnastkowy	Znak	Kod na kartach perforowanych	Symbole rozkazów maszynowych
76	4C	<	12-4-8	
77	4D	(	12-5-8	MH
78	4E	+	12-6-8	
79	4F	|	12-7-8	CVD
				CVB
80	50	&	12	
81	51		12-11-1-9	ST
82	52		12-11-2-9	
83	53		12-11-3-9	
84	54		12-11-4-9	
85	55		12-11-5-9	N
86	56		12-11-6-9	CL
87	57		12-11-7-9	O
88	58		12-11-8-9	X
89	59		11-1-8	L
				C
90	5A	!	11-2-8	
91	5B	$	11-3-8	A
92	5C	*	11-4-8	S
93	5D	)	11-5-8	M
94	5E	;	11-6-8	D
95	5F	¬	11-7-8	AL
96	60	—	11	SL
97	61	/	0-1	STD
98	62		11-0-2-9	
99	63		11-0-3-9	
100	64		11-0-4-9	
101	65		11-0-5-9	
102	66		11-0-6-9	
103	67		11-0-7-9	
104	68		11-0-8-9	MXD
105	69		0-1-8	LD
106	6A		12-11	CD
107	6B		0-3-8	AD
108	6C	,	0-4-8	SD
109	6D	%	0-5-8	MD
		—		DD
110	6E	>	0-6-8	
111	6F	?	0-7-8	AW
112	70		12-11-0	SW
113	71		12-11-0-1-9	STE

ZAŁĄCZNIK II (cd.)

Lp.	Kod szesnastkowy	Znak	Kod na kartach perforowanych	Symbole rozkazów maszynowych
114	72		12-11-0-2-9	
115	73		12-11-0-3-9	
116	74		12-11-0-4-9	
117	75		12-11-0-5-9	
118	76		12-11-0-6-9	
119	77		12-11-0-7-9	
120	78		12-11-0-8-9	LE
121	79		1-8	CE
122	7A	:	2-8	AE
123	7B	#	3-8	SE
124	7C	@	4-8	ME
125	7D	'	5-8	DE
126	7E	=	6-8	AU
127	7F	"	7-8	SU
128	80		12-0-1-8	SSM
129	81	a	12-0-1	
130	82	b	12-0-2	LPSW
131	83	c	12-0-3	
132	84	d	12-0-4	WRD
133	85	e	12-0-5	RDD
134	86	f	12-0-6	BXH
135	87	g	12-0-7	BXLE
136	88	h	12-0-8	SRL
137	89	i	12-0-9	SLL
138	8A		12-0-2-8	SRA
139	8B		12-0-3-8	SLA
140	8C		12-0-4-8	SRDL
141	8D		12-0-5-8	SLDL
142	8E		12-0-6-8	SRDA
143	8F		12-0-7-8	SLDA
144	90		12-11-1-8	STM
145	91	j	12-11-1	TM
146	92	k	12-11-2	MVI
147	93	l	12-11-3	TS
148	94	m	12-11-4	NI
149	95	n	12-11-5	CLI
150	96	o	12-11-6	OI
151	97	p	12-11-7	XI
152	98	q	12-11-8	LM

Lp.	Kod szesnastkowy	Znak	Kod na kartach perforowanych	Symbole rozkazów maszynowych
153	99	r	12-11-9	
154	9A		12-11-2-8	
155	9B		12-11-3-8	
156	9C		12-11-4-8	SIO
157	9D		12-11-5-8	TIO
158	9E		12-11-6-8	HIO
159	9F		12-11-7-8	TCH
160	A0		11-0-1-8	
161	A1		11-0-1	
162	A2	s	11-0-2	
163	A3	t	11-0-3	
164	A4	u	11-0-4	
165	A5	v	11-0-5	
166	A6	w	11-0-6	
167	A7	x	11-0-7	
168	A8	y	11-0-8	
169	A9	z	11-0-9	
170	AA		11-0-2-8	
171	AB		11-0-3-8	
172	AC		11-0-4-8	
173	AD		11-0-5-8	
174	AE		11-0-6-8	
175	AF		11-0-7-8	
176	B0		12-11-0-1-8	
177	B1		12-11-0-1	
178	B2		12-11-0-2	
179	B3		12-11-0-3	
180	B4		12-11-0-4	
181	B5		12-11-0-5	
182	B6		12-11-0-6	
183	B7		12-11-0-7	
184	B8		12-11-0-8	
185	B9		12-11-0-9	
186	BA		12-11-0-2-8	
187	BB		12-11-0-3-8	
188	BC		12-11-0-4-8	
189	BD		12-11-0-5-8	
190	BE		12-11-0-6-8	
191	BF		12-11-0-7-8	

ZAŁĄCZNIK II (cd.)

Lp.	Kod szesnastkowy	Znak	Kod na kartach perforowanych	Symbole rozkazów maszynowych
192	CO		12-0	
193	C1	A	12-1	
194	C2	B	12-2	
195	C3	C	12-3	
196	C4	D	12-4	
197	C5	E	12-5	
198	C6	F	12-6	
199	C7	G	12-7	
200	C8	H	12-8	
201	C9	I	12-9	
202	CA		12-0-2-8-9	
203	CB		12-0-3-8-9	
204	CC		12-0-4-8-9	
205	CD		12-0-5-8-9	
206	CE		12-0-6-8-9	
207	CF		12-0-7-8-9	
208	D0		11-0	
209	D1	J	11-1	MVN
210	D2	K	11-2	MVC
211	D3	L	11-3	MVZ
212	D4	M	11-4	NC
213	D5	N	11-5	CLC
214	D6	O	11-6	OC
215	D7	P	11-7	XC
216	D8	Q	11-8	
217	D9	R	11-9	
218	DA		12-11-2-8-9	
219	DB		12-11-3-8-9	
220	DC		12-11-4-8-9	TR
221	DD		12-11-5-8-9	TRT
222	DE		12-11-6-8-9	ED
223	DF		12-11-7-8-9	EDMK
224	E0		0-2-8	
225	E1		11-0-1-9	
226	E2	S	0-2	
227	E3	T	0-3	
228	E4	U	0-4	
229	E5	V	0-5	

ZAŁĄCZNIK II (cd.)

Lp.	Kod szesnastkowy	Znak	Kod na kartach perforowanych	Symbole rozkazów maszynowych
230	E6	W	0-6	
231	E7	X	0-7	
232	E8	Y	0-8	
233	E9	Z	0-9	
234	EA		11-0-2-8-9	
235	EB		11-0-3-8-9	
236	EC		11-0-4-8-9	
237	ED		11-0-5-8-9	
238	EE		11-0-6-8-9	
239	EF		11-0-7-8-9	
240	F0	0	0	
241	F1	1	1	
242	F2	2	2	
243	F3	3	3	
244	F4	4	4	
245	F5	5	5	
246	F6	6	6	
247	F7	7	7	
248	F8	8	8	ZAP
249	F9	9	9	CP
250	FA		12-11-0-2-8-9	AP
251	FB		12-11-0-3-8-9	SP
252	FC		12-11-0-4-8-9	MP
253	FD		12-11-0-5-8-9	DP
254	FE		12-11-0-6-8-9	
255	FF		12-11-0-7-8-9	

Literatura

Alfierowa Z.W., Lichaczewa G.N., Szurakow W.W., *Matiematiczeskoje obiespieczenije* EWM, Statistika, Moskwa 1974.

Brooks F.P.Jr., Iverson K.E., *Automatyczne przetwarzanie danych, System 360*, WNT, Warszawa 1975.

Day A.G., *Full Table Quadratic Searching for Scotter Storage*, „Communications of the ACM" 1970, nr 8.

Gienierator programm wwoda dannych dla JS IBM, Statistika, Moskwa 1976.

Fatiejew A.E., Rojtman A.I., Fatiejewa T.P., *Prikładnyje programmy w sistiemie matiematiczeskogo obiespieczenija JS* EWM, Statistika, Moskwa 1976.

Germain G., *Programing the IBM/360*, Prentice-Hall, Inc., 1967.

Jagielski R., *Operowanie danymi w DOS JS*, „Informatyka" 1974, nr 12.

Jagielski R., *Raspriedielennaja pamiat'*, „Uprawlajuszczije sistiemy i maszyny" 1976, nr 5.

Kamburelis T., *Architektura logiczna EMC JS*. Problemy informatyki, Ośrodek Badawczo-Rozwojowy Informatyki, Warszawa 1976.

Korolew L.N., *Struktury IBM i ich matiematiczeskoje obiespieczenija*, Nauka, Moskwa 1974.

Naumow W.W. i in., *Supervisor OS JS IBM, Statistika*, Moskwa 1975. *Opieracionnaja sistiema DOS JS. Obszczyje położenije*, Statistika, Moskwa 1975.

Opis języka PL/I, WNT, Warszawa 1975.

Pelc J., Sobanicc J., Świderski F., *Środki techniczne Jednolitego Systemu elektronicznych maszyn cyfrowych. Urządzenia zewnętrzne Jednolitego Systemu*, „Informatyka" 1973, nr 9.

Programmirowanije na jazykie Assemblera JS IBM, Statistika, Moskwa 1975.

Praca zbiorowa (pod red. A.M. Łarionowa), *Processor IBM JS-1020*, Statistika, Moskwa 1975.

Praca zbiorowa (pod red. A.M. Łarionowa), *Sistiema matiematiczeskogo obiespieczenija JS EWM*, Statistika, Moskwa 1974.

Praca zbiorowa (pod red. A.M. Łarionowa, *Sistiema dokumentacyi Jedinoj Sistiemy EWM*, Statistika, Moskwa 1975.

Rudniew J.P., Jemielianow A.A., *Organizacyja programmnogo obiespieczenija sistiem tieleobrabotki pri ispolzowanii opieracyonnoj sistiemy OS/360*, „Uprawlajuszczije sistiemy i maszyny" 1975, nr 5.

Sistiema IBM/360. Wwiedienije w zapominajuszczije ustrojstwa priamogo dostupa i mietody organizacyi dannych, Statistika, Moskwa 1974.

[illegible] techniczne *Jednolitego Systemu elektronicznych maszyn cyfrowych. Komputery Jednolitego Systemu*, „Informatyka" 1973, nr 8.
Sprzęt Jednolitego Systemu elektronicznych maszyn cyfrowych, Ośrodek Badawczo-Rozwojowy Informatyki, Warszawa 1976.
Stably D., *Logical Programming with System 360*, New York 1970.
Dokumentacja użytkowa systemu DOS/JS, MERA-ELWRO, Wrocław
C12001-1 DOS/JS Wprowadzenie do systemu.
C12101-1 DOS/JS Assembler — opis języka.
C12012-1 DOS/JS Assembler F.
C12401-1 DOS/JS COBOL — opis języka.
C12402-1 DOS/JS COBOL w systemie DPS/JS.
C12402-1 DOS/JS COBOL — komunikaty.
C12501-1 DOS/JS FORTRAN — opis języka.
C12502-1 DOS/JS FORTRAN — w systemie DOS/JS.
C12503-1 DOS/JS FORTRAN IV — opis języka.
C12504-1 DOS/JS FORTRAN IV w systemie DOS/JS.
C12801-1 DOS/JS RPG — opis języka.
C12802-1 DOS/JS RPG w systemie DOS/JS.
C12901-1 DOS/JS PL/I — opis języka.
C12902-1 DOS/JS PL/I — wstęp do programowania.
C12903-1 DOS/JS PL/I w systemie DOS/JS.
C13001-1 DOS/JS Organizacja wejścia-wyjścia.
C13002-1 DOS/JS Makroinstrukcje wejścia-wyjścia.
C13003-1 DOS/JS Etykiety dyskowe.
C13004-1 DOS/JS Etykiety taśmowe.
C13101-1 DOS/JS Program łączący.
C13102-1 DOS/JS Bibliotekarz.
C13103-1 DOS/JS AUTOTEST — program testujący.
C13101-1 DOS/JS CEAID — program protokołujący.
C13201-1 DOS/JS Programy transmisji.
C13202-1 DOS/JS Programy organizacyjne.
C13203-1 DOS/JS Niezależny program pomocniczy dyski — taśma — dyski.
C13204-1 DOS/JS Pomocnicze instrukcje wejścia-wyjścia.
C13301-1 DOS/JS Sortowanie zbiorów taśmowych.
C13302-1 DOS/JS Sortowanie zbiorów taśmowych i dyskowych.
C13401-1 DOS/JS Generowanie systemu.
C13600-1 DOS/JS Programy sterujące.
C13602-1 DOS/JS Makroinstrukcje łączności z supervisorem.
C14001-1 DOS/JS Podręcznik operatora.
C14002-1 DOS/JS Komunikaty operatorskie.

ERRATA

Str.	Wiersz		Jest	Powinno być
	od góry	od dołu		
9	11		kupienie	skupienie
9	16		komputerów i nazwaną	komputerów nazwaną
14	5		podzielony	podzielny
39	2		liniowe	liczące
42		15	wykonanej	wykonywanej
50	3		mu.	programu Ładowanie Początkowe IPL
50	19		prowadzenia programu	wprowadzenia programu
51	6		podłącza się logicznie	podłącza się
63	4		Temu właśnie zadaniu kraje współpracujące	Kraje współpracujące
81	4		na podstawie systemu	za pomocą systemu
116		7	nawias-kartoteki	nazwa-kartoteki
127	9		// JOB PRZYK 1	// JOB PRZYK1
283		2	ciągiem słów	ciągiem cyfr
313	12		EXTEND	EXTENT
314	podpis pod rys. 60		DTFSD	DTFSI
317	12		KEYARD	KEYARG

R. Jagielski — *Komputery Jednolitego Systemu*

Redaktor techniczny
Władysława Nasternak

Korektor
Mieczysław Szostakowski

Printed in Poland
Państwowe Wydawnictwo Ekonomiczne
Warszawa 1980
Zlec. 13/75. Wyd. I. Nakład 3000+240 egz.
Ark. wyd. 21,1. Ark. druk. 22,75
Papier druk. sat. kl. IV, 70 g. Format 61×86/16
Oddano do składania 24.I.1978. Podpisano do druku
i druk ukończono w lutym 1980 r.
Cena zł 53,—
Zakłady Graficzne w Katowicach, Zakład nr 5, Bytom, ul. Stahla 2 G-12

W serii „INFORMATYKA W PRAKTYCE" ukazały się dotychczas następujące pozycje:

MARTIN ZSCHOCKE
Elektroniczne przetwarzanie danych w gospodarce materiałowej
Warszawa 1975
s. 118, cena zł 18,—

AGATA ROJEK-GROSZEWSKA
ANDRZEJ ZALESKI
Gromadzenie! danych do elektronicznego przetwarzania
Warszawa 1976
s. 350, cena zł 40,—

KIT GRINDLEY, JOHN HUMBLE
Skuteczność wykorzystania komputera
Warszawa 1976
s. 245, cena zł 45,—

STANISŁAW ZADROŻNY
Organizacja zbiorów w małej informatyce
Warszawa 1977
s. 159, cena zł 25,—

EDWARD KOLBUSZ, EDWARD KRAM
Wdrażanie systemów informatycznych w przedsiębiorstwach przemysłowych
Warszawa 1977
s. 268, cena zł 36,—

ANDRZEJ JORDAN
Organizacja zbiorów w pamięciach dyskowych
Warszawa 1977
s. 129, cena zł 19,—

Praca zbiorowa
Przechowywanie danych
(tłum. z jęz. niem.)
Warszawa 1977
s. 154, cena zł 23,—

ZYGMUNT RYZNAR
Bank danych w przedsiębiorstwach przemysłowych
Warszawa 1978
s. 172, cena zł 26,—

ANTONI NOWAKOWSKI
WOJCIECH OLEJNICZAK
Minikomputery biurowe
Warszawa 1978
s. 168, cena zł 25,—

BRONISŁAW OBIREK
Przygotowanie przedsiębiorstwa do zastosowania informatyki w zarządzaniu
Warszawa 1978
s. 151, cena zł 23,—

IGNACY DZIEDZICZAK
Model księgowości informatycznej w przedsiębiorstwie
Warszawa 1979
s. 211, cena zł 38,—

JANUSZ ILCZUK, MARIA JERCZYŃSKA
Efektywność systemów informatycznych zarządzania
Warszawa 1979
s. 208, cena zł 38,—

ANDRZEJ Z. IDŹKIEWICZ
Ochrona informacji w procesie przetwarzania
Warszawa 1979
s. 147, cena zł 26,—

Cena zł 53.—