OŚRODEK BADAWCZO–ROZWOJOWY

CENTRUM KOMPUTEROWYCH SYSTEMÓW AUTOMATYKI I POMIARÓW „MERA–ELWRO"

PION OPROGRAMOWANIA

PROGRAMOWANIE W JĘZYKU ASM

Wrocław, styczeń 1978 r.

Spis treści

WSTĘP

Niniejsza praca zawiera komplet informacji potrzebnych osobom piszącym programy w kodzie wewnętrznym maszyny M przy założeniu, że nie istnieje żadne oprogramowanie podstawowe tej maszyny z wyjątkiem translatora języka symbolicznego ASM (typu „assembler"); translator tego języka jest samodzielnym programem, niezależnym od jakiegokolwiek programu sterującego lub systemu operacyjnego.

Pierwsze dwa rozdziały tej pracy zawierają opis funkcjonalny maszyny M zredagowany z punktu widzenia programisty. Rozdział trzeci zawiera kompletny (ale nieformalny) opis symbolicznego języka programowania ASM; jego formalna gramatyka jest podana w Dodatku 2 na końcu pracy. Należy pamiętać, że język ten jest przeznaczony do pisania programów dla maszyny „surowej" i w związku z tym w przyszłości - po opracowaniu programu sterującego - może ulec zmianom i uzupełnieniom. Własności eksploatacyjne niezależnego translatora języka ASM będą opisane w oddzielnej publikacji.

Omawianą tutaj wersję języka ASM opracował większy zespół pracowników dwóch instytucji: Instytutu Informatyki Uniwersytetu Wrocławskiego oraz Ośrodka Badawczo-Rozwojowego Centrum Komputerowych Systemów Automatyki i Pomiarów „MERA-ELWRO"; składu zespołu nie podajemy - ze względu na jego zmienność. Niniejszy zeszyt dokumentacji jest wynikiem pracy zespołu 7-osobowego w składzie:

dr Krystyna Jerzykiewicz (U.Wr.)
mgr Piotr Kremienowski (MERA-ELWRO)
mgr Zdzisław Płoski (U.Wr.)
mgr Teresa Strzelczyk (MERA-ELWRO)
dr Jerzy Szczepkowicz (U.Wr.)
mgr Lidia Zajchowska (MERA-ELWRO)
mgr Alina Ziobro (MERA-ELWRO)

ROZDZIAŁ 1. Ogólna organizacja maszyny cyfrowej M

1.0. *Uwagi ogólne*

Mówiąc o maszynie cyfrowej M mamy na myśli jednostkę centralną z pamięcią operacyjną o pojemności 64 K bajtów oraz odpowiedni zestaw urządzeń.

W skład minimalnego zestawu urządzeń potrzebnego do uruchomienia translatora muszą wchodzić takie urządzenia, które umożliwią wprowadzenie informacji z odpowiedniego nośnika, wyprowadzenie informacji na nośnik oraz pozwolą na zorganizowanie współpracy z operatorem. Będą to więc takie urządzenie jak:

- czytnik taśmy papierowej CT 2030,
- perforator taśmy papierowej DT 105-S,
- drukarka mozaikowa z klawiaturą - DZM 180K, spełniająca rolę konsoli operatora.

Ponadto w poszerzonym zestawie przewidziane są następujące urządzenia:

- czytnik kart DARO 1220,
- drukarka mozaikowa DZM 180,
- pamięć taśm kasetowych PK1,
- pamięć taśmowa PT 105.

Zestaw wyżej wymienionych urządzeń zewnętrznych może być rozbudowany w miarę konkretnych potrzeb.

1.1. *Moduły maszyny*

W skład maszyny M wchodzą następujące moduły:

- jednostka arytmetyczna i sterowanie,
- pamięć mikroprogramów (opcjonalnie),
- pamięć operacyjna,
- zintegrowane kanały.

Jednostka arytmetyczna służy do wykonywania operacji arytmetycznych, logicznych i innych, w sposób określony przez sterowanie.

Sterowanie koordynuje pracę procesora według funkcji programowych i informacji sterujących z pamięci mikroprogramów.

Pamięć mikroprogramów (PM) przechowuje informację sterującą, która

określa sposób interpretacji treści rozkazów. Pamięć mikroprogramów może być oddzielnym modułem lub może mieścić się w pamięci operacyjnej.

Pamięć operacyjna (PO) służy do przechowywania programów oraz informacji wykorzystywanych przez system, czyli program pracujący w stanie S = 1 (p. 1.2). Jeden moduł pamięci operacyjnej może mieć pojemność maksymalnie 64K bajtów (1K = 1024 bajty), czyli 65536 bajtów.

Zintegrowane kanały służą do przekazywania informacji pomiędzy pamięcią operacyjną (w niektórych przypadkach przez jednostkę arytmetyczną), a urządzeniami zewnętrznymi, które również traktowane są jako moduły maszyny.

Moduły maszyny wybierane są przez 8-bitowy numer, stąd maksymalna ilość modułów wynosi 256 - są one ponumerowane od 0 do 255. Przykładowe numery modułów:

0 - jednostka arytmetyczna i sterowanie,
1 - pamięć mikroprogramów (PM),
2 - pierwszy moduł pamięci operacyjnej,
3 - numer rezerwowy, który może być przydzielony drugiemu modułowi pamięci operacyjnej.

Numery od 4 wzwyż przeznaczone są dla urządzeń zewnętrznych.

1.2. *Rejestry*

Maszyna M jest wyposażona (zależnie od potrzeby) w cztery grupy rejestrów 16-bitowych, po 16 rejestrów w każdej grupie. Bity w rejestrach numerowane są od strony lewej do prawej, od 0 do 15.

Funkcje poszczególnych rejestrów przedstawia poniższa tabela:

Numer grupy	Numer rejestru w grupie	Bezwzględny numer rejestru	Funkcja rejestrów
1	2	3	4
0	0-15	0-15	Rejestry programowe dostępne poprzez adresowanie częścią RA lub RM rozkazu
1	0	16	Premodyfikator
1	1	17	Maska przerwań
1	2	18	Klucz, numer modułu

1	2	3	4
1	3	19	Słowo stanu procesora (SSP)
1	4	20	Datum
1	5	21	Klawiatura, wyświetlacz
1	6	22	Licznik kroków
1	7	23	Numer przerwania TIME-OUT
1	8	24	Rejestr przeznaczony do wykorzystania przez pamięć wirtualną
1	9	25	Klucz/numer modułu tylko na najbliższy cykl PO dla arytmometru. Potem automatycznie wykorzystuje się rejestr 18 (do użytku mikroprogramu) tzw. dubler (nie można odczytywać)
1	10-15	26-31	Nie używane. Brak realizacji technicznej
2	0-15	32-47	Rejestry robocze mikroprogramów
3	0-15	48-63	Dodatkowa grupa rejestrów aktualnie nie używana

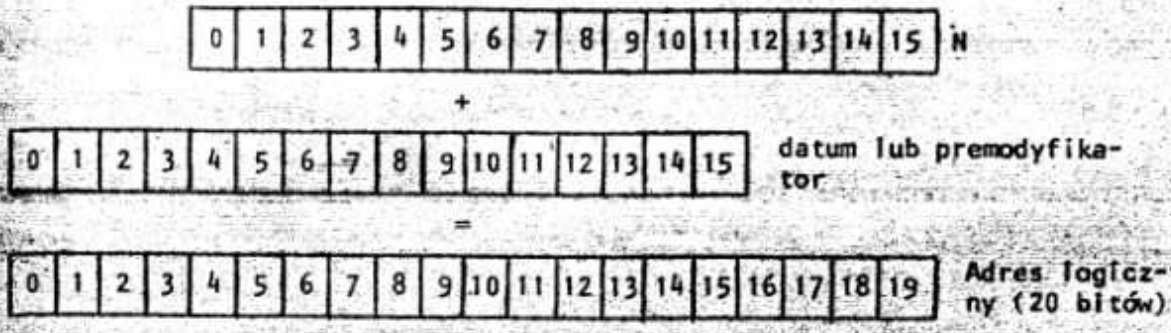
Grupę 0 rejestrów stanowią rejestry o numerach od 0 do 15. Mogą być one wykorzystane w programach jako rejestry robocze. Ich numery występują w rozkazach na częściach RA i RM (p. 1.4). Rejestr 15 pełni funkcję licznika rozkazów i jego zawartość odpowiada zawsze adresowi aktualnie wykonywanego rozkazu. Rejestr 15 ma więc specjalne przeznaczenie i nie należy go używać jako normalnego rejestru programowego. Każda zmiana jego zawartości musi być zamierzona, gdyż powoduje zmianę sekwencji wykonywanych w programie rozkazów.

Przy wykonywaniu kolejnych rozkazów zawartość R15 zmienia się o 2 lub 4, zależnie od tego czy rozkaz miał długość 2 bajty, czy 4 bajty (p. 1.4).

Grupę 1 rejestrów stanowią rejestry o numerach od 16 do 31. Rejestry te są dostępne dla programisty poprzez specjalnie do tego celu przeznaczone rozkazy (p. 2.12). Funkcja poszczególnych rejestrów tej grupy jest omówiona poniżej.

Rejestry premodyfikator i datum służą do przeadresowania adresu N w rozkazie (p. 1.4), w wyniku którego powstaje 20-bitowy adres logiczny. Polega

to na zsumowaniu arytmetycznym adresu N z zawartością odpowiedniego rejestru (datum lub premodyfikator) przesuniętą o 4 bity w lewo. Można to przedstawić w następujący sposób:



W stanie S = 1, jeśli pamięć mikroprogramów jest oddzielnym modułem, datum zawiera adres początkowy pamięci równy 0.

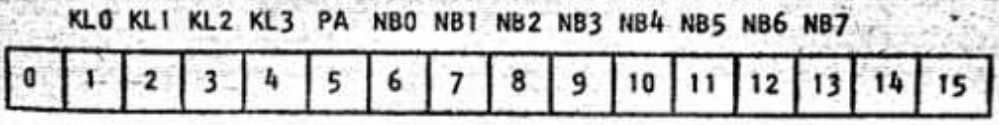
Jeśli pamięć mikroprogramów zajmuje część pamięci operacyjnej, to datum zawiera najbliższy podzielny przez 16 adres pamięci nie zajętej już przez mikroprogramy, podzielony przez 16. Tak określone datum nazywamy DPS (datum początkowe systemu).

Rejestr premodyfikator służy do przeadresowywania pól danych. Zawartość premodyfikatora nie może być mniejsza od zawartości datum (premod ≥ datum).

Maska przerwań jest 16-bitowym rejestrem, w którym poszczególne bity służą do maskowania odpowiadających im poziomów przerwań (16 poziomów). Poziomy przerwań związane są z przerwaniami od urządzeń zewnętrznych. Każdemu urządzeniu odpowiada jeden poziom przerwania, chociaż do jednego poziomu można podłączyć kilka urządzeń zewnętrznych. Poziomy przerwań ponumerowane są od 0 do 15 i każdemu poziomowi odpowiada w masce przerwań bit o tym samym numerze.

Warunkiem przyjęcia przerwania od urządzenia zewnętrznego jest ustawienie jedynki w masce przerwań na bicie, którego numer jest numerem poziomu przerwań odpowiadającego danemu urządzeniu.

W rejestrach o numerach 18 i 25 zapisywana jest informacja dotycząca klucza ochrony dla danego modułu pamięci i numer tego modułu.



KL0-KL1-KL2-KL3 - klucz ochrony pamięci

NB0-NB1-NB2-NB3-NB4-NB5-NB6-NB7 - numer modułu

Klucz ochrony służy do zabezpieczenia informacji przechowywanej w danym module pamięci operacyjnej przed nieuprawnionymi użytkownikami. Dla każdego zadania wykonywanego w procesorze jest ładowany klucz odpowiedni

dla tego zadania. Przy kontakcie procesora z pamięcią operacyjną następuje sprawdzenie, czy podany klucz jest identyczny z kluczem ochrony zaadresowanej informacji. Jeśli jest zgodność, to cykl pamięci jest wykonywany. W przypadku, gdy klucze są różne, rozpatrywane są jeszcze dwie możliwości:

- jeśli KL0 = KL1 = KL2 = KL3 = 0 (procesora) - cykl odbywa się,
- jeśli informacja chroniona jest tylko przed zapisem, to dla operacji CZYTAJ cykl odbywa się, a dla pozostałych operacji ustawia się wskaźnik błędu i następuje przerwanie typu TIME-OUT (p. 1.7).

W innych modułach maszyny klucz ochrony może pełnić inną funkcję zgodnie z opisem modułu.

Bit 5 rejestru 18, oznaczony przez PA, jest wykorzystywany jako maska dla niektórych przerwań wewnętrznych:

PA = { 0 w przypadku przerwań typu BP, TIME-OUT i nielegalny rozkaz następuje STOP mikropr. (patrz p. 1.7)
     { 1 zezwolenie na wyżej wymienione przerwania

UWAGA

Rejestr 25 wykorzystywany jest tylko dla jednego cyklu pamięci operacyjnej (pierwszego po załadowaniu rejestru 25), a później procesor korzysta z rejestru 18.

Słowo stanu procesora (SSP) zawiera 16-bitową informację, w której poszczególne bity mają następujące znaczenie:

numer bitu	nazwa symboliczna bitu	znaczenie
0	V	wskaźnik nadmiaru
1	C	wskaźnik przeniesienia
2	U	wskaźnik ujemnego wyniku
3	Z	wskaźnik wyniku zero
4	M	maska przerwania od wskaźnika V, jeśli M = 0 - przerwanie jest ignorowane
5, 6, 7	F, G, H	bity rezerwowe, mogą być wykorzystane dla celów programowych
8, 9, 10	I, J, K	bity rezerwowe, mogą być wykorzystane przez program pracujący w stanie S = 1

numer bitu	nazwa symboliczna bitu	znaczenie
11, 12	Q, R	starsza część adresu rejestrów RA i RM. Odpowiednie ustawienie wartości tych bitów umożliwia użycie jako rejestrów programowych szesnastu rejestrów dowolnej grupy 0, 1, 2 lub 3 (wybranej przez te bity)
13	P	wartość najmniej znaczącego bitu, wyniku operacji
14	W	adresacja wirtualna: W = 1 adresacja wirtualna, W = 0 adresacja bezpośrednia
15	S	system: S = 1 praca systemu, S = 0 praca programu użytkowego

Rejestr 21 (pierwsza grupa rejestrów) służy do kontaktowania się z pulpitem technicznym (z klawiaturą i wyświetlaczem).

Odczyt stanu klawiatury odbywa się poprzez odczyt zawartości rejestru 21 (bity od 0 do 15).

Zapis informacji do rejestru 21 powoduje wyświetlenie tej informacji na 16 lampkach na pulpicie technicznym.

Rejestry 22 i 23 są używane przez mikroprogramy, dlatego nie będziemy ich omawiać.

Rejestr 24 jest przeznaczony do przyszłościowego wykorzystania przez pamięć wirtualną.

1.3. *Bajty, słowa, adresy*

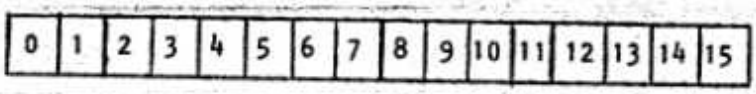
Maszyna M jest maszyną bajtową, której podstawową jednostką jest bajt czyli 8 kolejnych bitów. Dwa kolejne bajty, z których pierwszy ma adres parzysty, tworzą 16-bitowe słowo maszynowe. W słowie tym bajt o adresie parzystym nazywamy lewym bajtem słowa, a bajt o adresie o 1 większym - prawym bajtem słowa.

Przyjmuje się, że adresy słów maszynowych są adresami parzystymi, tzn. adresem słowa jest adres lewego bajtu tego słowa. Najmniej znaczący bit

adresu słowa (tj. ostatni bit nie jest uwzględniany.

Na przykład bajty o adresach 152 i 153 tworzą słowo o adresie 152, a w skład słowa o adresie 117 wchodzą bajty o adresach 116 i 117.

Bity w słowie maszynowym są ponumerowane od 0 do 15 (od strony lewej do prawej). Można to zilustrować następującym rysunkiem:



Zawartość słowa maszynowego może być interpretowana jako:

- rozkaz (p. 1.4),
- adres pamięci (p. 1.4),
- dane (np. liczby stałoprzecinkowe - p. 1.6).

1.4. *Rozkaz*

Rozkazy maszyny M można podzielić na 2 rodzaje:

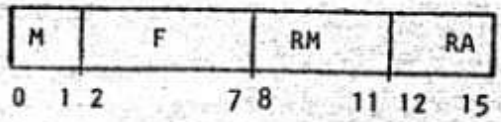
- rozkazy typu rejestr-rejestr (R-R),
- rozkazy typu rejestr-pamięć (R-M).

Rozkazy typu R-R są to rozkazy działające na zawartościach rejestrów. Numery rejestrów podawane są na częściach RA i RM rozkazu.

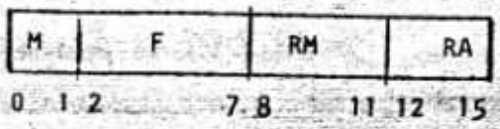
Dla rozkazów typu R-M jednym z argumentów jest adres pamięci.

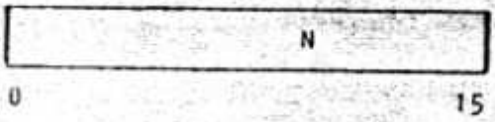
Rozkaz maszyny zajmuje 2 lub 4 bajty, tj. 16 bitów lub 32 bity. Postać rozkazu jest następująca:

a) rozkaz 2-bajtowy



b) rozkaz 4-bajtowy





gdzie M - jest rozszerzeniem kodu operacji (F), a mianowicie:

bit 0 = 0 oznacza typ a) rozkazu (tj. rozkaz 2-bajtowy),

bit 0 = 1 oznacza tyb b) rozkazu (tj. rozkaz 4-bajtowy),

bit 1 = 0 oznacza brak modyfikacji adresu (p. 1.5),

bit 1 = 1 oznacza, że istnieje modyfikacja adresu zawartością rejestru, którego numer mieści się na części RM rozkazu.

I tak w zależności od typu adresacji część M rozkazu przyjmuje następujące wartości:

M = 0, jeśli rozkaz jest 2-bajtowy typu rejestr-rejestr (R-R),

M = 1, jeśli rozkaz jest 2-bajtowy typu rejestr-pamięć (R-M); adres pamięci mieści się w rejestrze, którego adres występuje na części RM rozkazu,

M = 2, jeśli rozkaz jest 4-bajtowy typu rejestr-pamięć (R-M); adres pamięci mieści się na części N rozkazu,

M = 3, jeśli rozkaz jest 4-bajtowy typu rejestr-pamięć (R-M); adres pamięci jest tworzony poprzez dodanie do adresu występującego na części N rozkazu zawartości rejestru, którego numer występuje na części RM.

F - kod rozkazu określający jego funkcję, który w wielu wypadkach należy rozpatrywać łącznie z zawartością pola M.

RM - numer rejestru biorącego udział w działaniach rozkazów typu R-R lub numer rejestru modyfikacji (p. 1.5) w rozkazach typu R-M, lub argument rozkazu (p. np. 2.1, 2.8, 2.9).

RA - numer rejestru pełniącego rolę akumulatora programowego.

Dla przykładu rozkazy języka ASM (p. 3):

a) LD #1, #A

b) ADD #1, +#2, ALA

zajmują odpowiednio jedno i dwa słowa maszynowe i mają następujące przedstawienie ósemkowe poszczególnych części rozkazu:

	M	F	RM	RA
(a)	0	00	12	1
(b)	3	15	2	1
	adres słowa oznaczonego etykietą ALA			

Przebieg wykonywania rozkazu jest następujący:

a) pobranie rozkazu,
b) modyfikacja adresu (obliczenie adresu argumentu),
c) wykonanie treści rozkazu,
d) dodanie do R15 (licznik rozkazów) wielkości 2 lub 4 w zależności od długości rozkazu.

Rozkazy skokowe wykonywane są według następującego schematu:

a) pobranie rozkazu,
b) sprawdzenie warunku (dla skoków warunkowych) i przy spełnionym warunku ładowanie adresu argumentu do R15,
c) jeśli warunek nie jest spełniony, to wykonanie czynności d) opisanej wyżej.

1.5. *Modyfikacja*

W programach można używać dwóch następujących typów modyfikacji:

- Modyfikacja rozkazu, za pomocą której można zmienić cały następny rozkaz lub tylko jego części (p. 2.6). Ten rodzaj modyfikacji wykonuje się za pomocą rozkazów modyfikacji.
- Modyfikacja adresu, która jest pewnym rodzajem modyfikacji indeksowej. Modyfikacja ta występuje wtedy, gdy wartość bitu b1 rozkazu jest równa 1 (p. 1.4).

Jeśli w programie występują oba rodzaje modyfikacji, to najpierw wykonuje się modyfikację rozkazu (wykonanie odpowiedniego rozkazu modyfikacji - p.2.6), a dopiero potem modyfikację adresu.

Oczywiście modyfikacja rozkazu może zmienić część M następnego rozkazu. W tym przypadku o tym czy modyfikacja adresu ma wystąpić czy nie, decyduje nowa wartość M (obliczona po modyfikacji rozkazu).

PRZYKŁADY

rozkazy	skutek wykonania
LD #1,+#5	Jeśli r5 = ALA, to rozkaz z lewej jest równoważny LD #1,ALA
LD #3,+#7,WK	Jeśli r7 = 6, to rozkaz z lewej jest równoważny rozkazowi LD #3,WK+6
MODM #1 LD #2,+#3,ST	Jeśli r1 = 4, a r7 = 100, to w wyniku wszystkich modyfikacji wykonane zostanie podstawienie r2:=ST+100

1.6. *Arytmetyka stałoprzecinkowa*

Dane w pamięci maszyny można pamiętać w postaci liczb:

- stałoprzecinkowych krótkich (jednosłowowych),
- stałoprzecinkowych długich (wielosłowowych).

Liczby stałoprzecinkowe krótkie zajmują jedno słowo maszynowe (16 bitów) i należą do przedziału $\langle -2^{15}, 2^{15}-1 \rangle$. Liczby ujemne pamiętane są w arytmetyce uzupełnieniowej. Bit zerowy jest bitem znaku. Np. liczbie 15 w postaci stałoprzecinkowej krótkiej odpowiada ciąg bitów 00...01111, a liczbie -5 odpowiada ciąg bitów 11...11011. Liczby stałoprzecinkowe można także traktować jako liczby nieujemne z przedziału $\langle 0, 2^{16}-1 \rangle$, np. adres ostatniego bajtu jest równy 65535, a zapamiętany w słowie maszynowym ma taką samą reprezentację jak liczba -1.

Liczby stałoprzecinkowe długie można tworzyć i pamiętać w programach jako ciągi kolejnych słów, a działania na tych liczbach można programować jako ciągi działań na liczbach krótkich, uwzględniając w odpowiedni sposób wskaźnik przeniesienia C (p. 2.1).

1.7. *Przerwania*

Początkowy obszar pamięci operacyjnej maszyny jest wykorzystywany przez mikroprogramy jako miejsce na zapamiętanie informacji dotyczącej sposobu reakcji na przerwania i sposobu transmisji do lub z urządzeń zewnętrznych. Ponieważ w każdej chwili pracy programu może wystąpić przerwanie, zawartość początkowego obszaru pamięci musi być określona w standardowy sposób w momencie uruchomienia programu. Aby określić postać informacji tworzącej tę zawartość, zdefiniujemy kilka pojęć wiążących się z przerwaniami.

Tablicą warunków przerwania będziemy nazywali ciąg kolejnych sześciu słów maszynowych zawierających odpowiednio następujące wartości:

- datum procedury obsługi przerwania,
- słowo stanu procesora, w którym ostatni bit ma wartość 1 (S = 1),
- klucz ochrony (na bitach o numerach od 1 do 4) dla modułu pamięci zawierający procedurę obsługi przerwania; bit o numerze 5 jest tzw. wskaźnikiem PA, którego znaczenie omówimy dalej; na bitach o numerach od 6 do 13 znajduje się numer modułu pamięci (w standardowych zestawach maszyny wyposażonych tylko w jeden moduł pamięci operacyjnej o pojemności 64 KB, numer ten jest równy 2),
- maska przerwań,
- premodyfikator procedury obsługi przerwania,
- adres względny pierwszego rozkazu obsługi przerwania względem podanej wartości datum.

Z każdym urządzeniem zewnętrznym o numerze z przedziału ⟨4,255⟩ jest związany tzw. numer poziomu przerwań, zależny od danej konfiguracji maszyny. Numer ten jest liczbą całkowitą z przedziału ⟨0,15⟩.

Przerwaniem zewnętrznym będziemy nazywali przerwanie związane z dowolnym z urządzeń, to znaczy takie przerwanie, które może powstać tylko wtedy, gdy dane urządzenie wyśle sygnał przerwania, a w masce przerwań wartość bitu o numerze równym poziomowi przerwań danego urządzenia jest równa 1.

Pozostałe przerwania będziemy nazywali przerwaniami wewnętrznymi i oznaczymy je symbolami:

BP błąd parzystości w czasie odczytu informacji z modułu maszyny o numerze od 2 do 255,
TOUT time-out, niezgodność klucza lub błąd przy kontakcie z modułem maszyny o numerze od 2 do 255, do którego nastąpił zwrot,
REST restart po zaniku napięć (szczegóły tego przerwania omówiono dalej),
ZO zgłoszenie operatora,
PST przekroczenie stosu (pojęcie stosu wyjaśnimy dalej),
VS brak adresu wirtualnego,
EKSK esktrakod,
NIELR nielegalny rozkaz (np. rozkaz modyfikacji po rozkazie modyfikacji).

Przerwanie wewnętrzne ZO jest traktowane podobnie jak przerwanie zewnętrzne, to znaczy ma przyporządkowany numer poziomu przerwań równy 0.

Tak więc, jeśli w masce przerwań bit 0 jest zerem, to nie dojdzie do przerwania wskutek zgłoszenia operatora.

Część pozostałych przerwań wewnętrznych wystąpi zawsze (po zaistnieniu odpowiedniej przyczyny), jeśli tylko ustawiony ostatnio wskaźnik PA (między innymi występuje on w tablicy warunków przerwań) był jedynką. W przypadku, gdy PA = 0 i zaistniało przerwanie wewnętrzne BP, TOUT lub NIELR, następuje zatrzymanie pracy maszyny (stop mikroprogramu).

Istotnym elementem przerwań jest organizacja tzw. stosu. Stosem nazywamy ciąg słów pamięci operacyjnej poczynając od słowa o adresie względnym 1028 (względem datum początkowego systemu - DPS). Długość tego ciągu musi być równa N*2+1 (N*42 bajtów), gdzie N≥3 jest maksymalną liczbą nieobsłużonych przerwań. Słowa o adresach względnych 1024 i 1026 zawierają odpowiednio wskaźnik stosu, czyli bezwzględny adres początku stosu (wartość początkowa musi być równa bezwzględnemu adresowi następnego słowa) i granicę stosu, czyli bezwzględny adres końca stosu. Tak więc przed uruchomieniem programu:

(1024) = DPS*16+1026

(1026) = (1024)+N*42

dla ustalonej wartości N. Liczba w nawiasach oznacza zawartość słowa o podanym adresie względnym.

Słowa o adresach 0, 2,..., 1024 muszą mieć też określoną zawartość, którą mikroprogramy interpretują w następujący sposób:

Adres słowa	Zawartość
0	adres tablicy warunków przerwania BP
2	adres tablicy warunków przerwania TOUT
4	adres tablicy warunków przerwania REST
6	adres tablicy warunków przerwania ZO
8	adres tablicy warunków przerwania PST
10	adres tablicy warunków przerwania VS
12	adres tablicy warunków przerwania EKSK
14	adres tablicy warunków przerwania NIELR
16	adres pola sterującego urządzenia o numerze 4
18	adres tablicy warunków przerwania od urządzenia 4
20	adres pola sterującego urządzenia o numerze 5

Adres słowa	Zawartość
22	adres tablicy warunków przerwania od urządzenia 5
...	
1020	adres pola sterującego urządzenia o numerze 255
1022	adres tablicy warunków przerwania od urządzenia 255
1024	wskaźnik stosu
1026	granica stosu

Wszystkie adresy tablic warunków przerwań i adresy pól sterujących urządzeń są bezwzględne.

Pola sterujące dla urządzeń zostaną omówione w p. 1.8.

Jeśli powstanie przerwanie i S = 1, to mikroprogram zapisuje na stos następujące informacje:

Adres słowa	Zawartość
(1024)+2	rejestr 0
(1024)+4	rejestr 1
:	
(1024)+32	rejestr 15
(1024)+34	premodyfikator
(1024)+36	maska przerwań
(1024)+38	klucz ochrony, wskaźnik PA i numer modułu
(1024)+40	słowo stanu procesora
(1024)+42	datum

Następnie mikroprogram zwiększa o 42 wskaźnik stosu, określa nowe wartości datum, słowa stanu procesora, klucza ochrony, wskaźnika PA, numeru modułu, maski przerwań, premodyfikatora i rejestru 15 (licznika rozkazów) na podstawie tablicy warunków procedury obsługi odpowiedniego przerwania. W zależności od typu przerwania mikroprogram pamięta odpowiednią informację w rejestrze 14, a dla niektórych przerwań dodatkowo określa zawartości rejestrów R8 - R13. Rodzaj informacji pamiętanych w tych rejestrach opisano w pracy „Opis funkcjonalny mikroprogramowanej jednostki sterującej OS-1844901-9" wydanej przez OBRKSAIP MERA-ELWRO 1977. Jeśli po wykonaniu wymienionych wyżej czynności w stosie nadal jest miejsce na zapamiętanie

informacji o trzech przerwaniach ((1024)+3*42≤(1026)), to następuje przejście do obsługi przerwania (wg licznika rozkazów); w przeciwnym razie występuje następne przerwanie z powodu przekroczenia stosu (PST), w którym już nie sprawdza się, ile miejsca pozostało w stosie. Konieczność zachowania w stosie zapisu na 3 przerwania wiąże się z tym, że zapas stosu jest sprawdzany po wystąpieniu przerwania (pierwsze przerwanie), po którym w stosie musi zostać miejsce na zapisanie przerwania przekroczenia stosu (drugie przerwanie), a jako ostatnie (trzecie) może jeszcze wystąpić przerwanie spowodowane zanikiem napięć, które też musi być zapamiętane na stosie.

W przypadku, gdy przerwanie występuje przy S = 0, mikroprogramy wykonują te same czynności z tą jednak różnicą, że zawartość stosu nie zmienia się, a aktualne wartości rejestrów zostaną zapisane w początkowym obszarze pamięci przerwanego programu (poczynając od komórki o adresie równym wartości datum programu). Należy więc pamiętać, aby w programach zarezerwować pierwsze 21 słów na zapisanie informacji w momencie przerwania.

W procedurach obsługi przerwań należy wyróżnić przypadek, gdy przerwanie nastąpiło przy S = 0. Można to zrobić budując wskaźnik stosu. Jeśli stos jest pusty, to wartości rejestrów przed przerwaniem należy odtworzyć według wartości datum ostatnio działającego programu.

Realizacja przerwania REST (restart) dzieli się na dwa etapy. Przerwanie następuje w momencie zaniku napięć. Na stosie (lub w początkowym obszarze programu) zapisuje się takie same informacje, jak w przypadku każdego innego przerwania, po czym następuje zatrzymanie mikroprogramu. W chwili powrotu napięć mikroprogramy przekazują sterowanie do procedury obsługi przerwania według tablicy warunków przerwania REST (restart programu).

1.8. *Sterowanie transmisją urządzeń zewnętrznych*

Każdemu urządzeniu zewnętrznemu odpowiada jedno słowo w początkowym obszarze pamięci operacyjnej przeznaczone na zapamiętanie adresu pola

sterującego transmisją (p. 1.7). W momencie zainicjowania transmisji do lub z urządzenia, mikroprogram pobiera parametry transmisji z pola sterującego, którego adres musi być określony wcześniej (w słowie o adresie bezwzględnym równym DPS*16+n*4 dla urządzenia o numerze równym n). Pole sterujące składa się z 3 lub 4 słów o wartościach oznaczających odpowiednio:

- adres bufora transmisji,
- klucz ochrony, wskaźnik PA i numer modułu (jak dla tablicy warunków procedury obsługi przerwań) dla tego obszaru pamięci, gdzie znajduje się bufor transmisji; jeśli bit 0 tego słowa jest równy 1, to mamy do czynienia z omówionym dalej łańcuchem pól sterujących,
- licznik bajtów transmisji (wartość różna od zera),
- adres następnego w łańcuchu pola sterującego, uwzględniany tylko wtedy, gdy bit 0 w drugim słowie jest równy 1.

Jeśli bit 0 w drugim słowie pola sterującego jest zerem, to cała transmisja jest opisana przez jedno pole sterujące o długości 3 słów. W przeciwnym razie mikroprogram inicjuje transmisję zadanej liczby bajtów do lub z bufora o podanym adresie, po czym wpisuje adres następnego pola sterującego (zawartość ostatniego słowa aktualnego pola sterującego) do początkowego obszaru pamięci (słowo o adresie bezwzględnym równym DPS*16+4*n), a wskazane w ten sposób nowe pole sterujące zawiera parametry dalszego ciągu transmisji. Nie ma ograniczenia na liczbę pól sterujących w jednym łańcuchu. Ostatnim polem sterującym w łańcuchu jest to pole, w którym bit 0 drugiego słowa jest zerem.

Używając łańcuchów pól sterujących można za pomocą jednej transmisji wprowadzać (lub wyprowadzać) znaki do kilku buforów (mogą być rozłączne) z różnymi kluczami ochronnymi. Należy jednak pamiętać, że po takiej transmisji adres pola sterującego dla danego urządzenia (zawartość słowa w adresie bezwzględnym równym DPS*16+n*4) wskazuje na ostatnie pole sterujące łańcucha. Jeśli więc transmisja ma być powtórzona, to należy odpowiednio określić adres pierwszego pola sterującego.

ROZDZIAŁ 2. Kod rozkazowy maszyny

2.0. *Symbolika stosowana w opisach rozkazów*

Aby opisać działanie rozkazów maszyny M przyjmujemy następujące oznaczenia:

RA - część RA rozkazu (p. 1.4),

RM - część RM rozkazu (p. 1.4),

ARG - adres argumentu rozkazu, tj. bądź część N rozkazu (p. 1.4), bądź też adres otrzymany w wyniku wykonania ewentualnej modyfikacji (p. 1.5) części N,

D - rejestr DATUM (p. 1.2),

(D) - zawartość DATUM,

ra lub (RA) - zawartość rejestru o numerze RA,

rm lub (RM) - zawartość rejestru o numerze RM,

arg lub (ARG) - wartość bajtu lub słowa (p. 1.3) o adresie ARG,

:= - znak podstawienia w sensie ALGOLu 60; przykładowy napis ra:=rm oznacza więc podstawienie do rejestru o numerze RA zawartości rejestru o numerze RM,

L(RA) - lewy bajt (p. 1.3) zawartości rejestru RA,

R(RA) - prawy bajt (p. 1.3) zawartości rejestru RA,

R0, R1,..., R15 - rejestry o numerach odpowiednio 0, 1,..., 15; w programach napisanych w języku ASM (por. ROZDZIAŁ 3) rejestry maszyny lub ich zawartości oznacza się symbolami postaci #c, gdzie c jest cyfrą szesnastkową (0, 1,..., 9, A, B, ..., F),

$(RM)_{RA}$ - wartość bitu o numerze RA zawartości rejestru o numerze RM,

l - długość rozkazu w bajtach (2 lub 4 - p. 1.4),

M0 - wartość najbardziej znaczącego bitu części M rozkazu (p. 1.4),

+ - * / - symbole działań arytmetycznych - odpowiednio: dodawania, odejmowania (lub zmiany znaku), mnożenia i dzielenia,

$\vee \wedge \not\equiv \neg$ - symbole działań logicznych - odpowiednio: dodawania, mnożenia, odejmowania (różnicy symetrycznej) i negacji.

Opisując sposób określania zawartości rejestrów warunków U, Z oraz P używamy często skrótowego sformułowania

warunki U,Z,P wg ra

oznaczającego, że bity U, Z oraz P słowa stanu procesora (p. 1.2) przyjmują następujące wartości:

Z = 1, jeśli ra = 0,

Z = 0, jeśli ra ≠ 0,

U = 1, jeśli ra<0 (tzn. ra_0 = 1),

U = 0, jeśli ra≥0 (tzn. ra_0 = 0),

P = 0, jeśli ra_{15} = 0 (tzn. zawartość RA jest parzysta),

P = 1, jeśli ra_{15} = 1 (tzn. zawartość RA jest nieparzysta).

W tym rozdziale posługujemy się również opisanymi w p. 3.9 symbolicznymi zapisami rozkazów języka ASM. Najogólniejsza postać rozkazu zawiera kilka części, które zapisujemy w następującej kolejności:

(c1) Symboliczny lub ósemkowy kod rozkazu. Symboliczny kod rozkazu jest jedną z nazw podanych w tabeli znajdującej się w Dodatku 1; nazwa taka jest zawsze skrótem angielskiej nazwy rozkazu; na przykład kod symboliczny SLL pochodzi od Shift Left Logical.

(c2) Oznaczenie części RA rozkazu, tj. jeden z szesnastu symboli #0, #1, ..., #F postaci #c, gdzie c jest cyfrą szesnastkową oznaczającą wartość części RA.

(c3) Oznaczenie części RM w postaci na ogół takiej jak oznaczenie RA; zapis części RM można również poprzedzać znakiem + (plus), który w omawianym kontekście jest symbolem modyfikacji adresu (p. 1.5).

(c4) Adres symboliczny (w ogólnym przypadku - wyrażenie symboliczne w sensie p. 3.7); w tym rozdziale będziemy stosować najprostsze (intuicyjnie oczywiste) postacie adresów, w tym binarne (tj. rozpoczynające się od 2!), ósemkowe (tj. rozpoczynające się od 8!) oraz szesnastkowe (tj. rozpoczynające się od H!).

Zapisy części c2, c3 oraz c4 oddzielamy przecinkami. Niektóre z tych części mogą nie występować w zapisie rozkazu; w takim przypadku pomija się również przecinek oddzielający tę część od następnej.

Część M rozkazu (p. 1.4) w zapisie symbolicznym nie występuje, ponieważ określa ją pośrednio postać zapisu rozkazu. Szczegóły opisano w p. 3.9 i w Dodatku 1. W tym miejscu podajemy jedynie dopuszczalne zapisy symboliczne rozkazu o kodzie LD oznaczającego pobranie słowa do rejestru o numerze równym części RA rozkazu.

Postać rozkazu	Wartości części słowa rozkazowego				
	M	F	RM	RA	N
LD #1, #2	0	0	2	1	
LD #1, +#2	1	0	2	1	
LD #1, 1234	2	0	0	1	1234
LD #1, +#2, 1234	3	0	2	1	1234

Innym przykładem rozkazu symbolicznego jest napis

SBIT #9,1

oznaczający rozkaz, w którym M = 0, F = 8!42, RM = 9 oraz RA = 1. W tym przypadku oznaczenie rejestru RA jest inne niż podano w (c2) powyżej, ponieważ część RA określa numer bitu, a nie słowo zapamiętane w rejestrze o numerze RA. Według analogicznych zasad ustalono podane w Dodatku 1 dopuszczalne postacie części adresowych rozkazów symbolicznych.

W tabeli rozkazów zaznaczono rozkazy, przy realizacji których bierze udział rejestr premodyfikatora.

Działania arytmetyczne stałoprzecinkowe

Argumentami działań arytmetycznych stałoprzecinkowych mogą być: rejestr, komórka pamięci lub część RM rozkazu. Rozkazy realizujące działania arytmetyczne stałoprzecinkowe zebrane są w poniższej tabeli. W kolumnie „Warunki" podany jest sposób ustawiania wskaźników, bądź zaznaczone są wskaźniki biorące udział w działaniu. Np. jeśli w jakimś wierszu występuje C,V oznacza to, że wskaźnik C jest ustawiany, gdy zachodzi przeniesienie, a wskaźnik V jest ustawiany, gdy wynik operacji przekracza pojemność słowa (16 bitów). Pozostałe wskaźniki warunków U, Z, P ustawiane są według zawartości rejestru RA.

Kod mnemoniczny	Treść symboliczna		Kod ósemkowy	Warunki
	typ R-R	typ R-M		
1	2	3	4	5
LD	ra:=rm	ra:=arg	00	C:=0
ADD	ra:=ra+rm	ra:=ra+arg	15	C,V
SUB	ra:=ra-rm	ra:=ra-arg	16	C,V
NGA	ra:=-rm	ra:=-arg	17	C,V
ADDC	ra:=ra+rm+C	ra:=ra+arg+C	12	C,V
SUBC	ra:=ra-rm-C	ra:=ra-arg-C	13	C,V

1	2	3	4	5
NGAC	ra:=-rm-C	ra:=-arg-C	14	C,V
LDN	ra:=RM		40	C:=0
ADDN	ra:=ra+RM		52	C,V
SUBN	ra:=ra-RM		53	C,V
LDA		ra:=ARG	40	C:=0
ST		arg:=ra	44	C:=0

Rozkazy ADDC, SUBC I NGAC uwzględniają wskaźnik przeniesienia C, tzn. wartość wskaźnika C bierze udział w działaniu. Rozkazy te służą do organizowania działań na liczbach wielosłowowych.

PRZYKŁADY

Poniższa tabela przykładów obrazuje sposoby wykorzystania działań arytmetycznych. W nawiasach kwadratowych [] ujęte są wskaźniki tablic.

Ciąg rozkazów	Skutek wykonania

LDN #1,4
ADD #1,A+8
ADD #1,B
SUBN #1,7
ST #1,KOS
	Obliczenie wartości wyrażenia kos:=kos:=4+a[8]+b-7

LDN #1,5 NGA #1,#1 ST #1,AS	Wykonanie podstawienia as:=-5

LD #5,T
ADD #5,#2
SUBN #5,1
LD #7,+#5,T
	Pobranie elementu tablicy T, którego wskaźnik obliczono w r5, co symbolicznie można zapisać: r7:=T[t+r2-1]

LD #1,+#6
ADD #1,C+4
ST #1,+#2,C
LDA #F,LAB-4
	obliczenie C[r2]:=(r6)+C[4] I przejście do etykiety LAB

Ciąg rozkazów	Skutek wykonania
LDN #7,0	r7:=0

LDA #1,ADR
LD#2,+#1
SUBN #2,7
ST #2,+#1,500
	Wykonanie podstawienia ADR[500]:=(ADR)-7

ADDC #3,A+4
ADDC #2,A+2
ADDC #1,A
	Dodanie do liczby trzysłowowej umieszczonej w R1, R2, R3 liczby trzysłowowej umieszczonej w A, A+2, A+4

2.2. *Działania logiczne*

W wyniku wykonania działania logicznego (iloczynu, sumy, różnicy symetrycznej bądź negacji) i-ty bit słowa wynikowego (i = 0, 1,..., 15) zależy od zawartości i-tych bitów argumentów działania w sposób określony w poniższej tabeli.

a_i	b_i	Iloczyn $a_i \wedge b_i$	Suma $a_i \vee b_i$	Różnica $a_i \pm b_i$	Negacja $\neg a_i$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Rozkazy działań logicznych zawarte są w tabeli:

Kod mnemoniczny	Treść symboliczna		Kod ósemkowy
	Typ R-R	Typ R-M	
AND	ra:=ra^rm	ra:=ra^arg	01
OR	ra:=ravrm	ra:=ravarg	02
XOR	ra:=ra⊻rm	ra:=ra⊻arg	03
NGL	ra:=¬rm	ra:=¬arg	04

Rozkazy te ustawiają w sposób następujący wskaźniki warunków: C:=0, V-bz, U, Z, P wg ra.

PRZYKŁAD 1.

Ustawienie na bitach 12:15 w rejestrze 7 binarnej wielkości 1111 można wykonać za pomocą następującego ciągu rozkazów:

```
LDN #2,15        [pobranie wielkości 15 do rejestru 2
OR #7,#2         [logiczne dodanie 15 do rejestru R7
```

PRZYKŁAD 2.

Pobrania wielkości (równej zero, jeśli liczba w słowie WAG jest podzielna przez 4, lub różnej od zera - w przeciwnym razie) dokonuje następujący ciąg rozkazów:

```
LDN #5,3
AND #5,WAG
```

PRZYKŁAD 3.

Sprawdzić, czy zawartość i-tego elementu tablicy T równa 14, jeśli tak skoczyć do etykiety E5. Odpowiedni fragment programu może mieć postać:

```
LDN #8,HIE
LD #7,I          [zakładamy, że I zawiera numer elementu tablicy T
XOR #8,+#7,T
BSZ E5           [jeśli Z = 1, to skok do E5; p. 2.3.
```

2.3. *Skoki*

Skokami nazywamy rozkazy, które umożliwiają zmianę wykonywanej sekwencji rozkazów w programie. Wykonanie skoku polega na umieszczeniu argumentu rozkazu skokowego w liczniku rozkazów (w rejestrze R15).

Rozróżniamy następujące rodzaje skoków:

- skoki uzależnione od określonej zawartości rejestru, które powodują zapisanie argumentu do licznika rozkazów, jeśli jest spełniona odpowiednia relacja,
- skok bezwarunkowy, którego wynikiem działania jest zawsze załadowanie argumentu do R15,
- skok ze śladem (CALL), który przed zmianą zawartości licznika rozkazów zapamiętuje jego stan zwiększony o długość rozkazu w rejestrze RA,
- skoki uzależnione od zawartości bitów w słowie stanu procesora - oznaczane: BCy i BSy, gdzie y jest zastępowane literą, która jest nazwą symboliczną bitu w SSP; RA dla tych skoków przyjmuje wartość równą numerowi bitu w SSP, określonego tą nazwą symboliczną.

Skoki: bezwarunkowy, ze śladem i według SSP nie zmieniają warunków. Skoki warunkowe powodują ustawienie warunków: U, Z, P według RA oraz C:=0.

Kod mnemoniczny	Treść symboliczna	Kod ósemkowy
BZ	jeśli ra = 0, to r15:=ARG	41
BN	jeśli ra<0, to r15:=ARG	42
BNN	jeśli ra≥0, to r15:=ARG	43
BUZ	ra:=ra-1 i jeśli z = 0, to r15:=ARG	64
JUMP	r15:=ARG	60
CALL	ra:= r15+1 i r15:=ARG, I = 2×(M+1) (p. Z.0)	50
BCy	jeśli $(SSP)_{RA}$ = 0, to r15:=ARG	61
BSy	jeśli $(SSP)_{RA}$ = 1, to r15:=ARG	62

PRZYKŁAD 1.

Podana sekwencja rozkazów oblicza sumę $S = a_0+a_1+\ldots+a_{n-1}$ (n=100) dla $a_i>0$, pomijając elementy ujemne i równe zero. Składniki sumy są pobierane z tablicy rozpoczynającej się od adresu A. Jeśli wynik S jest równy zero należy przejść do wykonania rozkazu z komórki o adresie zapamiętanym w R7.

```
    LDA #1,100
    LDN #2,0
    LDN #5,0
E2: LD #4,+#5,A
    BN #4,E1
    BZ #4,E1
    ADD #2,#4
E1: ADDN #5,2
    BUZ #1,E2
    ST #2,S
    BZ #2,+#7
```

PRZYKŁAD 2.

Porównanie zawartości komórek ST i KL. Jeśli st>kl następuje wykonanie rozkazu oznaczonego etykietą LABEL:

```
    LD #3,KL
    COMP #3,ST
    BSC LABEL
```

PRZYKŁAD 3.

Porównanie zawartości dwóch komórek i przejście do wykonania rozkazu oznaczonego etykietą ET, jeśli są one równe, realizuje każda z dwóch sekwencji rozkazów:

```
      LD #1,A
      SUB #1,A+6
      BZ #1,ET
      :
ET:
```

lub

```
      LD #1,A
      COMP #1,A+6
      BSZ ET
      :
ET:
```

Należy jednak pamiętać, że nie zawsze można rozkaz COMP zastąpić rozkazem SUB, ponieważ w wyniku odejmowania może powstać nadmiar (p. 2.1).

PRZYKŁAD 4.

Wywołanie podprogramu o nazwie DRUK wykonuje rozkaz:

```
      CALL #1,DRUK
```

W rejestrze R1 pamiętany jest ślad, czyli adres powrotu z podprogramu DRU

```
      DRUK: LD #5,AS
            .........
            .........
            JUMP +#1     [powrót z podprogramu według zawartości R1.
```

2.4. *Porównania*

Istnieją dwa rozkazy porównania - rozkaz porównania słów (wielkości 16-bitowych) i rozkaz porównania bajtów (wielkości 8-bitowych).

Porównywane słowa lub bajty są traktowane jako wielkości odpowiednio 16-bitowe i 8-bitowe nieujemne.

Bit znaku dla wielkości słowowych jest traktowany jako najbardziej znacząca cyfra rozwinięcia dwójkowego liczby. Porównując liczby (słowa) należy więc pamiętać, że liczba ujemna będzie zawsze wskazana jako większa od dodatniej, ponieważ na bicie znaku ma wartość 1.

A oto tabela określająca sposób generowania warunków dla rozkazów porównania:

Kod mnemoniczny	Treść		Kod ósemkowy	Warunki
	R-R	R-M		
COMP	ra>rm	ra>arg	05	C:=Z:=0
	ra<rm	ra<arg		C:=1; Z:=0
	ra = rm	ra = arg		C:=0; Z:=1
COMB		R(RA)>arg	25	C:=Z:=0
		R(RA)<arg		C:=1; Z:=0
		R(RA) = arg		C:=0; Z:=1

Pozostałe bity warunków są określone w następujący sposób: V-bz; U - nieokreślone, P - wg wyniku odejmowania, czyli P = 0, jeśli obie wielkości są parzyste lub obie są nieparzyste, w przeciwnym wypadku, jeśli jedna z wielkości jest parzysta, a druga - nieparzysta, P = 1.

PRZYKŁADY

Ciąg rozkazów	Skutek wykonania

LD #1,POLE
LDA #2,100
COMP #1,#2
BSC ETYK
	Jeśli zawartość komórki POLE jest mniejsza od 100, następuje przejście do ETYK.

LD #1,ALA COMP #1,OLA BSZ AS	Jeśli (ALA) = (OLA) to następuje przejście do AS.
COMP #2,+#6	Porównanie r2 z argumentem, którego adres umieszczony jest w R6.

2.5. *Mnożenie i dzielenie stałoprzecinkowe*

Rozkazy zawarte w poniższej tabeli są rozkazami typu R-M (rejestr-pamięć).

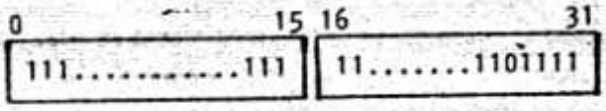
Mnożenie zdefiniowane jest dla liczb całkowitych jednosłowowych zawartych w RA i komórce pamięci. Wynik mnożenia jest liczbą podwójnej długości

zapisaną na 32 bitach dwóch kolejnych rejestrów RA i RA+1.

Dzielenie wykonywane jest dla podwójnie długiej dzielnej zawartej w dwu kolejnych rejestrach RA i RA+1 i jednosłowowego dzielnika zawartego w komórce pamięci. Wynik dzielenia zawarty jest w RA+1, reszta z dzielenia mieści się w RA. Reszta ma ten sam znak co dzielnik. Dzielenie zdefiniowane jest dla liczb całkowitych.

Kod mnemoniczny	Treść	Kod ósemkowy	Warunki
MPY	(RA,RA+1):=arg*ra	10	V-bz, C:=0, U wg ra, Z wg (RA,RA+1), P - nieokreślone
DIV	(RA,RA+1):=(RA,RA+1)/arg	11	C:=0, U,Z,P wg (RA+1), V=1, gdy dzielimy przez 0 lub wynik przekracza pojemność jednego słowa

Należy pamiętać, że liczba podwójnej długości zajmuje 32 bity. Bit zerowy najbardziej znaczącego słowa liczby jest bitem znaku. Np. liczba -17 zapisana w dwóch słowach ma postać:



PRZYKŁAD 1.

Iloczyn dwu liczb stałoprzecinkowych krótkich y i z można obliczyć za pomocą rozkazów:

```
LD #1,Y
MPY #1,Z      [wynik jest w rejestrach R1 i R2
```

PRZYKŁAD 2.

Iloraz dwu liczb krótkich nieujemnych (w myśl podstawienia c:=a/b) można obliczyć za pomocą rozkazów:

```
LDA #5,0
LD #6,A
DIV #5,B
ST #6,C       [przesłanie wyniku
```

PRZYKŁAD 3.

Jeśli dzielna jest liczbą krótką ujemną, to przed wykonaniem dzielenia do rejestru RA należy wpisać jedynki:

```
LDA #5,H!FFFF      [wpisanie jedynek
LD #6,A
DIV #5,B
ST #6,C            [przesłanie wyniku
```

2.6. *Rozkazy modyfikacji*

Istnieją trzy rozkazy modyfikacji rozkazu (p. 1.5), które zebrane są w tabeli:

Kod mnemoniczny	Treść	Kod ósemkowy
MODW	Modyfikacja następnego rozkazu zawartością rejestru RA	25
MODA	Modyfikacja części RA następnego rozkazu za pomocą 4 najmniej znaczących bitów ra	26
MODM	Modyfikacja części RM następnego rozkazu za pomocą 4 najmniej znaczących bitów ra	27

Wymienione rozkazy nie zmieniają warunków. Dwa rozkazy modyfikacji nie mogą występować kolejno po sobie.

PRZYKŁADY

Ciąg rozkazów	Skutek wykonania

LDN #1,5
MODM #1
LD #2,#3
	Wykonanie rozkazu LD #2,#8

LDN #3,2
MODA #3
LD #4,+#7,ALA
	Wykonanie rozkazu LD #6,+#7,ALA

Ciąg rozkazów	Skutek wykonania

LD #3,A
MODW #3
LD #4,+#0,ALA
	Jeśli wielkość A została zadeklarowana (p. 3.10) następująco:
	A:WORD 2!0000110100010011
	to ciąg rozkazów z lewej strony jest równoważny rozkazowi
	ADD #7,+#1,ALA

2.7. *Działania na bajtach*

Treścią rozkazów działań na bajtach jest zmiana zawartości wskazanego bajtu. Termin „wskazanego bajtu" oznacza tu lewy lub prawy bajt rejestru RA lub argumentu (p. 2.0). Rozkazy te można podzielić na dwie grupy:

- działania typu rejestr-rejestr,
- działania typu rejestr-pamięć.

Do grupy działań bajtowych typu rejestr-rejestr zaliczamy następujące rozkazy:

Kod mnemoniczny	Treść	Kod ósemkowy	Warunki
LZL	R(RA):= L(RM) L(RA):=0	20	V-bz.; C:=U:=0; Z, P wg ra
LZR	R(RA):=R(RM) L(RA):=0	21	V-bz.; C:=U:=0; Z, P wg ra
LSL	R(RA):=L(RM) L(RA) - bz.	22	bz.
LSR	R(RA):=R(RM) L(RA) - bz.	23	bz.
LLS	L(RA):=L(RM) R(RA) - bz.	24	bz.

Należy tu zwrócić uwagę, że wszystkie powyższe rozkazy zmieniają prawy bajt rejestru (z wyjątkiem rozkazu LLS), lewy bajt natomiast zerują (LZL, LZR) bądź pozostawiają bez zmian (LSL, LSR).

Do grupy działań bajtowych typu rejestr-pamięć zaliczamy następujące rozkazy:

Kod mnemoniczny	Treść	Kod ósemkowy	Warunki
LBLZ	R(RA):=arg; L(RA):=0	20	V-bz.; C:=U:=0; Z, P wg ra
LBLS	R(RA):=arg; L(RA):=bz.	22	bz.
LBRS	R(RA):=bz.; L(RA):=arg	24	bz.
STB	arg:=R(RA)	45	V-bz.; C:=0; U, Z, P wg ra
CNVB	R(RA):=(ARG+R(RA)); L(RA):=0	46	V-bz.; C:=0; U, Z, P wg ra

PRZYKŁAD 1.

Umieścić kod znaku A na prawym bajcie słowa TS, lewy bajt TS zerując. Wykona to następujący ciąg rozkazów:

```
ZNA: BYTE 'A'        [dyrektywa BYTE - p. 3.10

     LBLZ #4,ZNA     [pobranie znaku A z zerowaniem
     ST #4,TS
```

PRZYKŁAD 2.

Należy umieścić w słowie SLO dwa bajty, lewy ze słowa S1, prawy z elementu tablicy TAK, którego numer mieści się w słowie K.

```
LBRS #A,S1          [pobranie lewego bajtu ze słowa S1
LD #7,K             [umieszczenie wartości K w R7
LBLS #A,+#7,TAK     [pobranie do prawego bajtu R10 wartości (TAK+(K))
ST #A,SLO
```

PRZYKŁAD 3.

Zamiany znaku zawartego na prawym bajcie R3 na odpowiadający temu znakowi kod wewnętrzny zawarty w tablicy kodów TK, dokona rozkaz:

```
CNVB #3,TK
```

PRZYKŁAD 4.

Szukanie bajtu równego kodowi litery B wśród dziesięciu kolejnych elementów bajtowej tablicy TABA, począwszy od elementu o adresie TABA+(IL), można wykonać za pomocą następującego ciągu rozkazów:

```
ZNAK: BYTE'B'
      ......
      LDN #1,10
      LD #D,IL          [pobranie do R13 wartości IL
POCZ: LBLZ #2,+#D,TABA
      COMB #2,ZNAK      [porównanie bajtów
      BSZ ROWNE         [skok, gdy Z = 1 - wielkości są równe
      ADDN #D,1
      BUZ #1,POCZ
```

2.8. *Działania na bitach*

Do zmiany lub badania wartości pojedynczych bitów słowa służą cztery rozkazy zawarte w poniższej tabeli.

Kod mnemoniczny	Treść	Kod ósemkowy	Warunki
CBIT	$(RM)_{RA}:=0$	41	U, Z, P wg rm
SBIT	$(RM)_{RA}:=1$	42	U, Z, P wg rm
NBIT	$(RM)_{RA}:=\neg(RM)_{RA}$	43	U, Z, P wg rm
TBIT	$Z:=\neg(RM)_{RA}$	47	$U:=(RM)_{RA}$, dla RA = 0; U:=0, dla RA ≠ 0 $P:=(RM)_{RA}$, dla RA = 15; P:=0, gdy RA ≠ 15

Każdy z tych rozkazów zeruje wskaźnik C, a wskaźnik V - pozostawia bez zmian.

PRZYKŁAD 1.

W zależności od stanu bitu 4 słowa TAB (numeracja bitów od lewej i od zera) należy pobrać do R7 lewy (gdy bit 4 = 0) lub prawy (gdy bit 4 = 1) bajt z rejestru R6. Czynności te wykona następujący ciąg rozkazów:

```
      LD #5,TAB
      TBIT #5,4         [testuj czwarty bit R5
      BSZ LEWY
PRAWY: LZR #7,#6
```

```
      :
LEWY: LZL #7,#6
```

PRZYKŁAD 2.

Następujący ciąg rozkazów neguje w rejestrze R9 bit o numerze zawartym w I.

```
      LD #4,I
      MODA #4
      NBIT #9,0
```

2.9. *Przesuwania*

W maszynie M istnieją trzy następujące rodzaje przesuwań:

- arytmetyczne, które polega na tym, że przesuwając w lewo, wskaźnik V ładuje się w przypadku zmiany znaku, a przesuwając w prawo, bit znaku jest powielany;
- logiczne, polegające na tym, że przy przesuwaniu krótkim z prawej lub lewej strony (w zależności od kierunku przesuwania) wchodzą zera, a w przesuwaniu długim bierze udział wskaźnik C w sposób podany w poniższej tabeli;
- cykliczne, które polega na tym, że bity przemieszczają się w lewo lub w prawo, a bity wychodzące poza zakres przesuwania, wchodzą z drugiej strony.

Rozkazy przesuwań są podane w poniższych dwu tabelach. Pierwsza tabela podaje rozkazy przesuwań krótkich, druga rozkazy przesuwań długich.

Przesuwania krótkie realizują przesuwanie zawartości rejestru RA o RM miejsc i U, Z, P ustawiają według ra. Rozkazy te wskaźnik C zerują. Wskaźnik V zaznaczono w tabeli.

Kod mnemoniczny	Treść	Kod ósemkowy	Warunki
1	2	3	4
SCL	przesunięcie ra cyklicznie w lewo o RM miejsc	70	V-bz
SCR	przesunięcie ra cyklicznie w prawo o RM miejsc	74	V-bz

1	2	3	4
SLL	przesunięcie ra logicznie w lewo o RM miejsc	71	V-bz.
SLR	przesunięcie ra logicznie w prawo o RM miejsc	75	V-bz.
SAL	przesunięcie arytmetycznie w lewo o RM miejsc	72	V:=1, gdy następuje zmiana znaku w ra
SAR	przesunięcie arytmetyczne w prawo o RM miejsc	76	V-bz.

Przesuwania długie realizują przesuwanie zawartości (RA) i (RM) o 1 miejsce i ustawiają U według ra, Z według (ra,rm), a P - przyjmuje wartości nieokreślone. Pozostałe wskaźniki warunków podane są w tabeli.

Kod mnemoniczny	Treść	Kod ósemkowy	Warunki
SCL	przesunięcie ra, rm cyklicznie w lewo o 1 miejsce - $ra_{15}:=rm_0$, $rm_{15}:=ra_0$	30	V-bz., C:=0
SCR	przesunięcie ra, rm cyklicznie w prawo o 1 miejsce - $ra_0:=rm_{15}$, $rm_0:=ra_{15}$	34	V-bz., C:=0
SLL	przesunięcie ra, rm logicznie w lewo o 1 miejsce - $ra_{15}:=rm_0$, $rm_{15}:=C$, $C:=ra_0$	31	V-bz., $C:=ra_0$
SLR	przesunięcie ra, rm logicznie w prawo o 1 miejsce - $rm_0:=ra_{15}$, $ra_0:=C$, $C:=rm_{15}$	35	V-bz., $C:=rm_{15}$
SAL	przesunięcie ra, rm arytmetycznie w lewo o 1 miejsce - $ra_{15}:=rm_0$, $rm_{15}:=C$, $C:=ra_0$	32	V:=1, gdy zmiana znaku, $C:=ra_0$
SAR	przesunięcie ra, rm arytmetycznie w prawo o 1 miejsce - $rm_0:=ra_{15}$, $C:=rm_{15}$, ra_0 powielony	36	V-bz., $C:=rm_{15}$

PRZYKŁAD

Jeśli i-ty bit rejestru R3 jest jedynką należy przejść do etykiety LABEL5;

```
    LD #2,1
    MODM #2
    SCL #3,0        [przesunięcie krótkie o 1 bitów
    BSU LABEL5
```

UWAGA

W zapisie symbolicznym rozkazów przesuwań krótkich podaje się numer rejestru i długość przesunięcia, w przesuwaniach długich - numery dwu rejestrów biorących udział w przesuwaniu; np. SCL #3,2 oznacza przesunięcie krótkie w R3 o dwa bity, a SCL #3,#2 oznacza przesunięcie długie w rejestrach R3 i R2 (o 1 bit).

2.10. *Rozkaz zerowania słów (CLS)*

Rozkaz o kodzie mnemonicznym CLS (8155) zeruje RM+1 słów począwszy od słowa ARG. Symbolicznie treść rozkazu można zapisać następująco:

```
(ARG):=0
(ARG+2):=0
(ARG+4):=0
    .
    .
(ARG+2*RM):=0
```

Wskaźniki warunków pozostają bez zmian.

PRZYKŁADY

Ciąg rozkazów	Skutek wykonania
CLS #0,KROK	KROK:=0; zerowanie jednego słowa
MODM #3 CLS #0,TABL	zerowanie 5 lub 8 elementów tablicy TABL w zależności od zawartości R3 odpowiednio równego 4 lub 7
CLS +#0	zerowanie jednego słowa o adresie w R0
LD #0,J CLS +#0,TRA	zerowanie j-tego elementu tablicy TRA

UWAGA

Jeżeli w rozkazie zerowania stosuje się modyfikację adresu, to jako modyfikatora należy użyć rejestru, którego numer określa liczbę zerowanych słów.

2.11. *Działania na słowie stanu procesora*

Do tej grupy należą rozkazy, które służą do zmiany zawartości bitu w słowie stanu procesora. Rozkazy te są bezadresowe i ich kody mnemoniczne mają postać:

Cy - zeruj bit w SSP,
Sy - ustaw bit w SSP,
Ny - neguj bit w SSP,

gdzie y jest zastępowane literą, która jest nazwą symboliczną zmienianego bitu w słowie stanu procesora (p. 1.2).

W rozkazach tych po przetłumaczeniu część RA przyjmuje wartość zgodną z numerem bitu zmienianego danym rozkazem.

W przypadku pracy systemu (S = 1) mogą być zmieniane wszystkie bity słowa stanu procesora, tzn. bity od 0 do 15. Jeśli pracuje program (S = 0), to można zmieniać tylko bity od 0 do 7 w SSP.

Warunki ustawiane są dla tych rozkazów zgodnie z nową wartością słowa stanu procesora.

Do grupy rozkazów działających na słowie stanu procesora należy również rozkaz SETS, którego funkcją jest przepisanie lewego bajtu rejestru RA do lewego bajtu SSP. Prawy bajt SSP pozostaje niezmieniony.

Po wykonaniu rozkazu SETS warunki V, C, U, Z ustawiane są według nowej zawartości SSP.

Kod mnemoniczny	Treść	Kod ósemkowy
Cy	$(SSP)_{RA}:=0$	61
Sy	$(SSP)_{RA}:=1$	62
Ny	$(SSP)_{RA}:=\neg(SSP)_{RA}$	63
SETS	L(SSP):=L(RA); R(SSP):=bz.	56

PRZYKŁAD 1.

Rozkazy

CZ

NC

SV

wykonają kolejno: zerowanie wskaźnika Z, negowanie wskaźnika C, ustawienie wskaźnika V.

PRZYKŁAD 2.

Zakładamy, że

(SSP) = H1A001, czyli V =1, C =0, U =1, Z = 0, S = 1.

Po wykonaniu rozkazu

CV

(SSP) = H12001, czyli V:=0, a pozostałe warunki bez zmian.

PRZYKŁAD 3.

Jeśli przy tym samym założeniu jak w przykładzie 2 wykonamy rozkaz

SETS #5, gdzie (R5) = H16008

to (SSP) = H16001, czyli ustawiane zostaną warunki C = 1 i U = 1.

2.12. *Działania na rejestrach*

Rozkazy działań na rejestrach można podzielić na dwie grupy:

a) rozkazy kontaktu z pierwszą grupą rejestrów (LDR, STR),

b) rozkazy przepisywania rejestrów grupy 0 do pamięci i na odwrót (rozkazy MVRS, MVSR - p. 1.2).

Rozkazy grupy a) służą do zapisania lub odczytu dowolnego rejestru grupy 1.

Rozkazy grupy b) służą do przepisania pola rejestrów zerowej grupy do pamięci operacyjnej lub do zapisania pola pamięci w kolejnych rejestrach zerowej grupy. Poniższa tabela przedstawia treść tych rozkazów:

Kod mnemoniczny	Treść	Kod ósemkowy	Warunki
LDR	ra:=(16+RM)	50	V-bz, C:=0, U, Z, P wg ra
STR	(16+RM):=ra (tylko gdy S = 1)	51	bz., gdy RM ≠ 3; wg ra gdy RM = 3
MVRS	(ARG):=(RA) ... (ARG+2RM):=(RA+RM)	56	bz.
MVSR	(RA):=(ARG) ... (RA+RM):=(ARG+2RM)	57	V-bz, C:=0 (a) Z:=1, U:=P:=0, jeśli rozkaz kończy się i nie cały zadany blok został przepisany do rejestrów; (b) Z:=0, U:=1, P - dowolne, jeśli został przepisany cały zadany blok; (c) bz., gdy RM = 0, tj. zostanie przesłane jedno słowo

Dla rozkazu MVSR pole słów jest przepisywane kolejno. Jeśli zostanie zapisany rejestr R15, to rozkaz kończy przepisywanie pola słów pamięci do rejestrów.

PRZYKŁAD 1.

Rozkaz

LDR #2,#1

powoduje pobranie do R2 maski przerwań (rejestr 17 1-szej grupy rejestrów).

PRZYKŁAD 2.

Rozkaz

STR #2,#3 przy (R2) = H!B001

powoduje ustawienie warunków w słowie stanu procesora według (R2), czyli

(SSP):=H!B001, bo adresem SSP jest 19 (p. 1.2).

PRZYKŁAD 3.

Fragment programu

TABL:WORD 0(20)

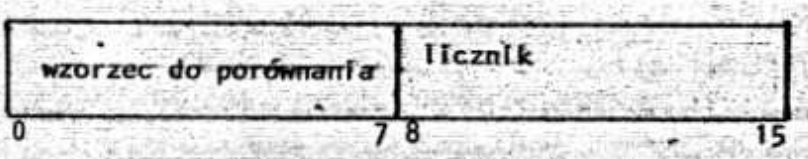
MVRS #0,#8,TABL

powoduje przepisanie zawartości rejestrów R0÷R8 do kolejnych komórek obszaru pamięci począwszy od TABL do TABL+16.

2.12. *Rozkaz szukania*

Rozkaz SEB (kod ósemkowy 47) służy do przeszukania łańcucha bajtów o określonej długości, przez porównanie kolejnych bajtów z wzorcem zapisanym w rejestrze RA. W zależności od wyniku szukania ustawiane są odpowiednie wskaźniki warunków.

W rejestrze RA należy przygotować następujące informacje:



Adres pola bajtów określony jest przez ARG. Przeszukiwanie dla licznika ≠ 0 rozpoczyna się od najwyższego adresu pola (ARG+licznik). Zawartość adresowanego bajtu porównywana jest z wzorcem i licznik zostaje zmniejszony o 1. W przypadku zgodności porównywanego bajtu z wzorcem wykonywanie rozkazu zostanie zakończone.

UWAGA

Jeśli licznik = 0, przeszukiwane pola ma długość 256 bajtów. Pierwszy porównywany jest bajt spod adresu ARG, a następnie ARG+255 i od tej chwili adresy bajtów są zmniejszane zgodnie z zawartością licznika, od którego po każdym kolejnym porównaniu odejmowane jest 1.

Warunki:

bz U:=P:=C:=1, Z:=0 - jeśli odczytany bajt jest identyczny ze wzorcem,

bz U:=P:=C:=0, Z:=1 - jeśli został wyzerowany licznik bajtów, a nie został znaleziony bajt identyczny ze wzorcem.

PRZYKŁAD

Wykonanie rozkazu:

SEB #2,TABL-1

przy założeniu, że (R2) =H!ABO7, TABL:BYTE 20,H!A1,H!AB,50,45,H!FF,H!8

polega na przeszukaniu pola o długości 7 bajtów, zaczynającego się od adresu TABL w celu znalezienia bajtu o zawartości HIAB.

Przeszukiwanie zaczyna się od adresu TABL+6. Operacja powinna się zakończyć na adresie TABL+2 i licznik w rejestrze 2 powinien mieć wartość 3. Warunki U:=P:=C:=1; Z:=0.

2.14. *Inicjowanie transmisji i rozkazy wejścia-wyjścia*

Informacje podane w tym punkcie dotyczą pracy programu w stanie S = 1. W celu zainicjowania transmisji na lub z urządzenia zewnętrznego należy wcześniej przygotować pola sterujące transmisją (p. 1.8) oraz (jeśli oczekiwanie na zakończenie transmisji jest organizowane poprzez przerwania) procedurę obsługi przerwania (p. 1.7).

Maska przerwań dla tego urządzenia powinna być odblokowana (tzn. powinna być zapisana jedynka na odpowiednim bicie w rejestrze maski - p. 1.2).

Każde urządzenie zewnętrzne posiada rejestry statusowe, które zależnie od funkcji można zapisywać lub odczytywać. Poprzez odczytanie informacji z odpowiedniego rejestru statusowego można dowiedzieć się, jaki jest stan urządzenia (statusy urządzeń podane są w opisach urządzeń). Zapisanie informacji do odpowiedniego rejestru wiąże się na ogół z wydaniem dyspozycji oraz określeniem rodzaju czynności, jakie to urządzenie ma wykonać (np. zainicjowanie transmisji).

Do odczytu lub zapisu statusu urządzenia służą rozkazy „czytaj status" - RST (kod ósemkowy 63) i „pisz status" - WST (kod ósemkowy 66).

Rejestr RA dla obu rozkazów zawierać powinien informację następującej postaci:

- bit 0 = 0,
- bity od 1 do 4 - klucz ochrony,
- bit 5 = 0,
- bity od 6 do 13 - numer modułu (zależnie od urządzenia),
- bity 14 i 15 = 0.

Status zapisywany jest z rejestru RA+1 i odczytywany do rejestru RA+1. W przypadku zapisu statusu, po zakończeniu operacji w rejestrze RA+1 znajduje się odczytany status urządzenia określający stan urządzenia. Jeśli urządzenie posiada więcej niż 2 rejestry statusowe (zapisywalne lub odczytywalne), to ich wybór

odbywa się przez efektywny adres rozkazów - ARG. Adres ten nie jest modyfikowany przez datum lub premodyfikator.

Dla rozkazów RST i WST warunki generowane są następująco: C:=0, U, Z, P wg (RA+1).

Do bezpośredniego przesłania informacji do dowolnego modułu systemu służą rozkazy „czytaj bezpośrednio" i „pisz bezpośrednio".

Informacja wysyłana jest z rejestru RA+1 i odczytywana również do rejestru RA+1. Rejestr RA powinien zawierać informacje takiej samej postaci jak dla rozkazów WST i RST.

Kod mnemoniczny	Treść	Kod ósemkowy
WDIR	pisz bezpośrednio	52
RDIR	czytaj bezpośrednio	53

Warunki: C:=0, U, Z, P według (RA+1).

PRZYKŁAD

Wprowadzenie do pola pamięci operacyjnej tekstu z taśmy papierowej. Tekst podzielony jest na wiersze zakończone znakami nowej linii. Maksymalna liczba znaków do nowej linii wynosi 100.

Zakładamy, że jest to fragment większego programu, więc mamy już uprzednio ustawioną tablicę obsługi przerwań wewnętrznych na początku programu (adresy względne 0 do 14) oraz wskaźnik stosu w 1024 i granicę stosu w 1026.

```
POCZ: LDR #2,#4        [pobranie datum programu
      SLL #2,4         [przesuń o 4 w lewo (logicznie)
      LDA #1,ADPS      [pobranie adresu pola sterującego
      ADD #1,#2        [dodanie datum
      ST #1,16         [zapamiętanie w komórce 16 ADPS+DAT
      LDA #1,ADTWP     [pobranie adresu tablicy obsługi przerwania
      ADD #1,#2        [dodanie datum
      ST #1,18         [zapamiętanie adresu tablicy obsługi przerwania
                       [w 18
      LDA #1,H!10
      LDA #2,H!A       [pobranie kodu NL
```

```
        WDIR #1,0          [ustawienie znaku końca
A1:     RST #1,0           [czytanie statusu
        SLL #2,6
        BSF *-8            [czy B6 =1?
        JUMP A1            [czekanie na operatywność
        LDA #2,H1C1        [status: żądanie transmisji, uwzględnienie zna-
                            ku końca, zezwolenie na przerwanie
        WST #1,0           [pisanie statusu
        CF                 [zerowanie bitu w SSP
        LDA #3,H18000
        STR #3,#1          [ustawienie maski przerwań
        BSU *-8            [czekanie na przerwanie
        JUMP *-4           [(w procedurze obsługi przerwania musi być wyko-
                            nany rozkaz SF)
A2:                        Dalsza część programu
        .
        .
        .
ADPS:   WORD BUF1,H18,100  [pole sterujące
ADTWP:  WORD DAT,1,H18,H10000,PREM,APROC   [tablica warunków procedury
                                            obsługi przerwań
```

Zainicjowanie transmisji jest również możliwe przy zablokowanym przerwaniu od danego urządzenia. Taka metoda jest nieopłacalna, ponieważ wymaga przesyłania po jednym znaku. Można z niej korzystać tylko w określonych przypadkach.

Rozpoczęcie przesyłania odbywa się również przez zapis odpowiedniego kodu (z ustawionym bitem żądania transmisji) do rejestru statusowego. W przypadku transmisji wejścia zapalenie gotowości urządzenia oznacza, że znak znajduje się już w buforze i może być z bufora odczytany do pamięci (rozkazem „czytaj bezpośrednio"). W przypadku transmisji wyjścia, po zapisaniu znaku do bufora (rozkazem „pisz bezpośrednio") należy czekać na zapalenie wskaźnika gotowości urządzenia, co oznacza, że znak został już wyprowadzony. Czekanie na gotowość odbywa się w obu przypadkach przez odczyt rejestru statusowego i badanie jego zawartości (bit 7 - gotowość).

PRZYKŁAD 1.

Wprowadzenie jednego znaku z taśmy papierowej do pamięci operacyjnej.

```
      LDN #2,0
      STR #2,#1          [zablokowanie maski przerwań
      LDA #1,H!10
A1:   RST #1,0           [czytanie statusu
      SLL #2,6           [przesunięcie o 6 w lewo
      BSU *8             [skok, jeśli B6 = 1
      JUMP A1            [czekanie na operatywność
A2:   LDA #2,H!80        [żądanie transmisji
      WST #1,0           [pisanie statusu
      SLL #2,5
      BSU *8             [skok, jeśli zapalony bit: transmisja
      JUMP A2
A3:   RST #1,0           [czytanie statusu
      SLL #2,7
      BSU *8             [skok, jeśli zapalony bit: gotowość
      JUMP A3            [czekanie na gotowość
      RDIR #1,0          [odczyt znaku z bufora urządzenia do rejestru 2
      LDA #4,10
      ST #2,+#4,POLE
      :                  dalsza część programu
      .
POLE: BYTE 0(10)
```

RZYKŁAD 2.

Wyprowadzenie jednego znaku z rejestru na taśmę papierową.

```
      LDN #2,0
      STR #2,#1          [zablokowanie przerwań
      LDA #1,H!14
A1:   RST #1,0           [czytanie statusu
      SLL #2, 6
      BSU *8             [skok,jeśli urządzenie operatywne
      JUMP A1            [czekanie na operatywność
A2:   LDA #2,H!80        [żądanie transmisji
      WST #1,0           [pisanie statusu
      SLL #2,5
      BSU *8             [skok jeśli zapalony bit: transmisja
```

```
        JUMP A2
        LDA #2,'B'
        WDIR #1,0          [zapisanie znaku z rejestru 2 do bufora urządze-
                            nia
A3:     RST #1,0
        SLL #2, 7
        BSU *+8            [skok jeśli zapalony bit: gotowość
        JUMP A3            [czekanie na gotowość
        :                   dalsza część programu
```

2.15. *Rozkazy związane z ochroną pamięci*

Informacja przechowywana w module maszyny może być chroniona przed nieuprawnionymi użytkownikami. W tym celu moduł może być wyposażony w pamięć kluczy ochrony. Ma to szczególne zastosowanie dla modułu pamięci operacyjnej. Moduł pamięci operacyjnej dzieli się na tak zwane strony o pojemności 2K bajtów.

Bity 0÷4 adresu określają numer strony modułu, a bity 5÷15 adres wewnątrz strony.

Dla każdej strony w pamięci kluczy przeznaczona jest jedna 7-bitowa komórka (adresowana bitami 0÷4 adresu pamięci operacyjnej), której poszczególne bity zawierają następujące informacje:

Numer bitu	Zawartość	Komentarz
0÷3	klucz ochrony	sprawdzany jest z liniami KL szyny
4	= 1	ochrona zachodzi dla wszystkich cykli pamięci
	= 0	zezwolenie na CZYTAJ
5	dowolny stan	załadowany przez PISZ KLUCZ
	= 1	przy każdym kontakcie ze stroną pamięci chronioną przez ten klucz
6	= 1	oznacza, że ze stroną chronioną przez dany klucz został wykonany cykl zmieniający informację przechowywaną przez tę stronę

Rozkazy, za pomocą których można zapisać (lub odczytać) informację do pamięci kluczy, wykonywane są poprzez PISZ KLUCZ lub CZYTAJ KLUCZ. Bity 0÷4 adresu pamięci, wygenerowanego w danym cyklu określają komórkę pamięci kluczy, do której zapisywana jest informacja z RA+1, bądź z której odczytywana jest informacja do RA+1. Rejestr RA w obu przypadkach zawiera następujące informacje:

Numer bitu	Zawartość
0	0
1÷4	klucz ochrony (sprawdzany)
5	0
6÷13	numer modułu
14÷15	0

Wskaźniki warunków pozostają bez zmian. Rozkazy te mogą być używane tylko przez system.

Kod mnemoniczny	Treść	Kod ósemkowy
RKEY	Czytaj klucz	51
WKEY	Pisz klucz	65

PRZYKŁAD 1.

Załóżmy, że (R1) = H18, a (R2) = H1F8. Rozkaz

```
WKEY #1,ADRS
```

zapisuje do komórki w pamięci kluczy odpowiadającej tej stronie PO (pamięci operacyjnej), która zawiera adres ADRS, klucz mający wartość liczbową H1F. Ochrona zachodzi dla wszystkich cykli pamięci, ponieważ bit 4=1.

PRZYKŁAD 2.

Aby odczytać tę komórkę z pamięci kluczy, należy użyć rozkazu RKEY #1,ADRS przy (R1)=H18.

2.16. *Rozkaz RET*

Rozkaz RET o kodzie ósemkowym 67 ma dwie interpretacje:

- jako rozkaz powrotu według stosu; jest wtedy rozkazem bezadresowym i może być użyty tylko w stanie S = 1,
- jako rozkaz powrotu według adresu; adres powrotu określony jest przez ARG; rozkaz może być wykonany tylko w stanie S=1.

RET, jako rozkaz powrotu według stosu, może być użyty w procedurze obsługi przerwania. Użycie go w takim przypadku powoduje odnowienie stanu procesora (21 rejestrów) według 21-słowowego pola zapisanego na stosie oraz zmniejszenie wskaźnika stosu o 42 (p. 1.7).

Wskaźniki warunków ustawiają się według odtworzonego SSP.

RET, użyty jako rozkaz powrotu według adresu, powoduje odnowienie stanu procesora według 21-słowowego bloku informacji o identycznej budowie, jak informacje zapisane na stosie. Mieści się on w pamięci począwszy od datum programu, a ARG rozkazu RET określa adres ostatniego słowa tego bloku (słowo 21). Wskaźniki warunków ustawiane są według odtworzonego tym rozkazem SSP.

2.17. *Inne rozkazy (TS, NULL, STOP, SETM)*

Rozkaz TS o kodzie ósemkowym 54 realizuje pobranie zawartości słowa o adresie ARG do rejestru RA i jednocześnie wypełnienie lewego bajtu tego słowa jedynkami. W przyjętych oznaczeniach treść tego rozkazu można zapisać tak:

ra:=arg, L(ARG):=HIFF

Wskaźniki warunków ustawiane są następująco: V - bz, C:=0, U, Z, P wg ra.

Rozkaz NULL ma kod ósemkowy 77. Powoduje tylko zwiększenie licznika rozkazów o 2, a wszystkie warunki pozostają bez zmian.

Rozkaz STOP o kodzie ósemkowym 60 jest rozkazem systemowym, tzn. wykonywany jest tylko w stanie S = 1. Służy do zatrzymania wykonywania bieżącej sekwencji rozkazów, aż do momentu wznowienia pracy przez operatora. W stanie STOP wykonywany jest mikroprogram obsługi pulpitu operatora.

Rozkaz SETM o kodzie ósemkowym 06 jest rozkazem systemowym. Symbolicznie treść jego można zapisać następująco:

r16:=rm+r20

Funkcją tego rozkazu jest dodanie zawartości rejestru datum do zawartości rejestru wskazanego przez część RM rozkazu. Wynik dodania zostaje odesłany do rejestru premodyfikatora (p. 1.2).

Wskaźniki warunków pozostają bez zmian.

ROZDZIAŁ 3. Język symboliczny ASM

3.0. Uwagi wstępne

Język ASM jest przeznaczony do symbolicznego zapisywania programów w kodzie wewnętrznym maszyny M. Omówioną w tym rozdziale pierwszą wersję języka ASM projektowano w okresie, gdy nie istniało jeszcze własne oprogramowanie maszyny M. W związku z tym w języku nie zakłada się istnienia jakiegokolwiek systemu operacyjnego lub programu sterującego i umożliwia się symboliczne zaprogramowanie każdej operacji wykonalnej w maszynie M za pomocą ustalonych mikroprogramów (por. p. 3.9) każdym technicznie dostępnym sposobem. W szczególności nie narzuca się żadnego sposobu gospodarowania pamięcią operacyjną: każdy bajt pamięci jest dostępny dla programu.

Przewiduje się, że po skonstruowaniu podstawowego programu sterującego powstanie rozszerzona wersja języka ASM przystosowana do tego programu. Należy przyjąć, że w tej wersji mogą pojawić się pewne ograniczenia dotyczące wykorzystania pamięci operacyjnej, wykonalności pewnych rozkazów itp.

W dalszym ciągu rozdziału określono nieformalnie składnię i znaczenie tekstów źródłowych w języku ASM; formalną gramatykę tego języka podano w Dodatku 1. W treści rozdziału nie korzystamy bezpośrednio z gramatyki; układ treści i terminologia odpowiada jednak układowi i terminologii gramatyki.

3.1. Alfabet języka ASM

Zbiór znaków używanych do pisania programów w języku ASM dobrano w ten sposób, że każdy znak tego zbioru ma naturalną transkrypcję na automacie piszącym CONSUL, dziurkarce kart typu ARITMA 130 oraz na drukarce mozaikowej DZM-180. Alfabet zawiera następujące znaki:

- duże litery alfabetu łacińskiego

  A B C D E F G H I J K L M

  N O P Q R S T U V W X Y Z

- cyfry

  0 1 2 3 4 5 6 7 8 9

- inne znaki widoczne w tekście drukowanym

  ! " # $ % & ' ( ) * + , - . / : ;

>< = ? @ [\ ] ↑ ←

oraz odstęp (widoczny w maszynopisie tylko wśród innych znaków) i zmianę wiersza.

W komentarzach zawartych w programie (p. 3.8) można również używać niektórych znaków automatu CONSUL, które nie mają naturalnej reprezentacji na drukarce ani na kartach. Transkrypcja takich znaków na drukarce zostanie określona w opisie translatora języka ASM.

W dalszym ciągu tego rozdziału w opisach składni elementów języka ASM będziemy używać również małych liter łacińskich lub ciągów takich liter, jednak zawsze w celu oznaczenia pewnego elementu złożonego, tj. zdefiniowanego w terminologii alfabetu języka lub elementów zdefiniowanych w innym miejscu.

3.2. *Znaki układu graficznego*

Podstawowym urządzeniem do przygotowywania tekstów źródłowych w języku ASM jest automat CONSUL produkujący maszynopis tekstu i równoważną taśmę papierową. O układzie graficznym tekstu decyduje sposób użycia odstępów i zmian wiersza. Używając kart papierowych jako nośnika tekstu, zamiast znaku zmiany wiersza stosujemy przejście do następnej karty.

Format tekstu w języku ASM jest swobodny, tzn. o znaczeniu elementu języka decyduje kontekst, w którym użyto tego elementu, a nie kolumny druku, w których element umieszczono. Na przykład słowo języka ASM

```
WORD:WORD WORD,WORD+WORD
```

zawiera pięć identycznych nazw, z których każda ma znaczenie określone przez swoje położenie w słowie (p. 3.10); słowo to można poprzedzić odstępami. Odstępów i zmian wiersza można w tekście używać swobodnie, jednak z następującymi ograniczeniami wynikającymi z definicji języka i własności translatora:

(a) Każde słowo symboliczne (p. 3.8 - 3.16) mieści się w całości w jednym wierszu.

(b) Zmiana wiersza jest jednym z dopuszczalnych symboli końca słowa programu (p. 3.8).

(c) Wiersz nie może zawierać więcej niż 80 znaków (pojemność karty dziurkowanej).

(d) Odstępy użyte poza tekstami (p. 3.10), symbolicznymi kodami znaków

(p. 3.3) i komentarzami (p. 3.8) służą do oddzielania jednostek leksykalnych języka. W tej roli (separatora) można zamiast jednego odstępu użyć wielu odstępów - bez zmiany znaczenia programu.

Pominięcie odstępu w roli separatora jest dopuszczalne jedynie w tym przypadku, gdy składnia elementów leksykalnych umożliwia ich jednoznaczne rozpoznanie. Na przykład słowo

```
LD #1, SUMA
```

można również napisać w postaci

```
LD#1,SUMA
```

ale w słowie

```
WORD SUMA+SUMA
```

odstępu po symbolu WORD nie wolno pominąć.

Jak wynika z ograniczenia (d), nie można używać odstępów we wnętrzu większości jednostek leksykalnych. Na przykład napisanie rozkazu

```
ADDN #F,4
```

w postaci

```
ADD N #F,4
```

jest błędem syntaktycznym.

3.3. *Liczby bez znaku*

W języku ASM liczby bez znaku są napisami oznaczającymi liczby całkowite (w tradycyjnym sensie) z przedziału $[0,2^{16}-1]$, tj. przedziału $[0,65535]$; w rejestrach maszyny liczby z tego przedziału reprezentuje się na ogół za pomocą słowa 16-bitowego

$$b_0 b_1 \ldots b_{15}$$

interpretowanego jako wartość równa sumie

$$b_0 * 2^{15} + b_1 * 2^{14} + \ldots + b_{15}$$

(por. p. 1.6).

Liczby bez znaku można pisać w jednej z pięciu postaci, których nazwy są następujące:

- liczba naturalna,
- liczba dwójkowa,
- liczba ósemkowa,
- liczba szesnastkowa,
- kod znaku.

Liczba naturalna jest ciągiem co najwyżej pięciu cyfr dziesiętnych i ma zwykłe znaczenie. Następne trzy postacie liczb rozpoczynają się odpowiednio od znaków:

2I
8I
HI

Po wymienionych parach znaków pisze się odpowiednio ciąg co najwyżej

- 16 cyfr dwójkowych (0 lub 1),
- 6 cyfr ósemkowych (od 0 do 7),
- 4 cyfr szesnastkowych (od 0 do F).

Napisy tej postaci interpretuje się jako pozycyjne zapisy liczb w układzie przy podstawie odpowiednio 2, 8 i 16. Cyfry szesnastkowe o wartościach od 10 do 15 oznacza się literami od A do F. Przykładowe napisy

2I1111111000
8I770
HI1F8

oznaczają więc liczbę równą 504; w postaci liczby naturalnej można ją zapisać jako ciąg trzech znaków

504

Ciąg pięciu znaków

00504

oznacza tę samą liczbę.

Ostatnim wymienionym sposobem zapisu liczby jest kod znaku; nazywamy tak napis o jednej z dwóch postaci

'c'
'cd'

gdzie c, d są dowolnymi znakami spośród wymienionych w p. 3.1 oraz w Dodatku 3, z wyjątkiem znaku ' oraz nowej linii. Napis 'c' oznacza liczbę równą kodowi znaku c w sensie tabeli podanej w Dodatku 3. Napis postaci 'cd' oznacza liczbę równą wartości

'c'*256+'d'

3.4. *Liczby*

Liczbą nazywamy liczbę bez znaku w sensie definicji podanej w p. 3.3 lub taką samą liczbę poprzedzoną znakiem - (minus). Jeżeli wl jest wartością liczby bez znaku l, to wartość liczby o postaci

-l

powstaje przez wykonanie operacji NGA (negacja arytmetyczna - p. 2.1) na wartości wl lub operacji równoważnej (np. odejmowanie od zera).

Z własności arytmetyki maszyny (p. 1.6) oraz definicji liczb bez znaku wynika więc, że w języku ASM można również zapisywać symbolicznie wartości interpretowane jako liczby całkowite z przedziału $[-2^{15},2^{15}-1]$, tj. przedziału [-32768,32767].

3.5. *Nazwy*

Nazwą w języku ASM jest co najwyżej 8-znakowy ciąg liter lub cyfr zaczynający się od litery. Nazwy służą do zapisywania kodów rozkazów i dyrektyw oraz do oznaczania komórek pamięci i stałych. Nazwy służące do oznaczania kodów rozkazów i dyrektyw są ustalone (p. 3.9 - 3.16), ale nie są zastrzeżone, ponieważ składnia języka ASM umożliwia rozpoznanie znaczenia nazwy przez kontekst, w którym ta nazwa została użyta. Nazwy służące do oznaczania komórek pamięci oraz stałych można dobierać dowolnie.

W dalszym ciągu rozdziału termin „nazwa" odnosi się jedynie do komórek pamięci i stałych; przez „kod rozkazu" i „kod dyrektywy" będziemy rozumieć wspomniane już ustalone nazwy - użyte w odpowiednim kontekście.

Wybrane przez programistę znaczenie nazw można ograniczyć do określonego poziomu oznaczeń: jednego globalnego poziomu oznaczeń i dowolnej liczby lokalnych poziomów oznaczeń - zwanych blokami (p. 3.14). Własność ta umożliwia pisanie różnych fragmentów programu przez różne osoby bez konieczności uzgadniania wszystkich oznaczeń; w szczególności jest możliwe stworzenie biblioteki podprogramów ogólnego użytku.

3.6. *Etykiety i nadawanie wartości nazwom*

Etykietą jest nazwa napisana na początku słowa programu (p. 3.8 - 3.16) i oddzielona dwukropkiem od pozostałej części słowa; część słowa następująca po dwukropku może być pusta. Na przykład w słowie

```
LAB: CALL #1,SUBRSQRT
```

nazwa LAB jest etykietą umieszczoną przed rozkazem CALL #1,SUBRSQRT, a słowo postaci

```
POINT:
```

zawiera tylko etykietę z dwukropkiem; taka etykieta odnosi się do następnego słowa programu. Na przykład we fragmencie programu

```
CASE1:
CASE5:
CASE71: CALL #1,XXX
```

albo

```
CASE1:
CASE5:
CASE71:
        CALL #1,XXX
```

możemy każdą z nazw CASE1, CASE5, CASE71 uważać za etykietę umieszczoną przed rozkazem CALL #1,XXX.

Etykieta służy do dwóch celów:

- wyróżnienia pewnego słowa programu (poprzez opatrzenie go nazwą),
- przyporządkowania nazwie użytej w roli etykiety wartości równej adresowi pewnego bajtu pamięci operacyjnej.

W typowym przypadku (jak w podanych przykładach) translator nadaje nazwie użytej w roli etykiety wartość równą adresowi pierwszego bajtu przekładu napisu następującego po dwukropku. Użycie etykiety nie ma wpływu na postać przekładu programu; jest to jedynie pewna informacja dla translatora i programisty.

Nazwie można również nadać dowolną wartość - niekoniecznie oznaczającą adres bajtowy fragmentu przekładu programu. Do tego celu służy dyrektywa nadania wartości. Jej kodem jest znak = (równość). Dyrektywa ta zawsze jest poprzedzana etykietą z dwukropkiem i ma postać

```
n:=ws
```

gdzie n jest nazwą, a ws jest tzw. wyrażeniem symbolicznym (p. 3.7), na przykład liczbą (p. 3.3 i 3.4). Dyrektywa tej postaci nie ma przekładu, a jedynym skutkiem przeczytania jej przez translator jest nadanie nazwie n wartości podanej po prawej stronie znaku równości; na przykład dyrektywa

```
ROZMTAB:=7000
```

nadaje nazwie ROZMTAB wartość równą 7000, a dyrektywa

```
B:='B'
```

nadaje nazwie B wartość równą kodowi litery B (p. Dodatek 3 - tabela kodów). Głównym zastosowaniem tej dyrektywy jest parametryzacja własności ilościowych programu (np. wielkości tablic, adresów segmentów) w celu ułatwienia ewentulanych zmian tych własności; ciąg takich dyrektyw zwykle umieszcza się na początku programu lub jakiegoś jego bloku.

Jeżeli wyrażenie ws w dyrektywie n:=ws zawiera nazwę, to każdej z nich trzeba nadać wartość we wcześniejszej części programu.

Dalsze szczegóły dotyczące przyporządkowywania wartości nazwom są związane m.in. z dyrektywami o kodach EVEN, PREM, REL, EREL oraz LOC i omawiamy je wraz z tymi dyrektywami w p. 3.12 - 3.15.

Ogólna reguła dotycząca etykiet jest następująca: na dowolnym ustalonym poziomie oznaczeń (lokalnym lub globalnym - p. 3.14) każda nazwa użyta w programie musi dokładnie jeden raz wystąpić w roli etykiety; dotyczy to każdego poziomu oznaczeń z osobna i niezależnie od pozostałych.

3.7. *Wyrażenia symboliczne*

Przyjmijmy następujące oznaczenia:

$s_0, s_1, s_2, \ldots$ - liczby bez znaku (p. 3.3) lub nazwy (p. 3.5),
$d_1, d_2, \ldots$ - znaki + (plus) lub - (minus).

Wyrażeniem symbolicznym bezwzględnym nazywamy dowolny napis o jednej z dwóch postaci:

$$s_0\ d_1\ s_1\ d_2\ s_2\ \ldots\ d_k\ s_k$$
$$-s_0\ d_1\ s_1\ d_2\ s_2\ \ldots\ d_k\ s_k$$

dla $k \geq 0$. Przykładami wyrażeń symbolicznych są więc napisy

```
180
-180
ADDR
-ADDR
-180+'A'
ABC+XY-KONIEC+2
-'A'+B
```

Wartości wyrażeń symbolicznych oblicza translator w czasie tłumaczenia programu; są one interpretowane jako adresy lub liczby - zależnie od kontekstu. Znaki + (plus) oraz - (minus) łączące składniki wyrażenia translator interpretuje odpowiednio jako dodawanie i odejmowanie w sensie rozkazów ADD oraz SUB (p. 2.7). Minus na początku oznacza zmianę znaku wartości s_o (w sensie rozkazu NGA).

Oto przykłady słów programu zawierających wyrażenia symboliczne:

```
      CALL #1,SUBRADDR+10
      LOC SUBREND+AD2-AD1+2
      WORD -7,10,'AB',AD1+AD2,-PAR+1
```

Wyrażeniem symbolicznym względnym nazywamy każdy napis postaci

```
      *wb
```

gdzie wb jest wyrażeniem symbolicznym bezwzględnym; jeżeli wartość wyrażenia wb jest zerem, to wyrażenie wb można w tym kontekście pominąć.

Wartość wyrażenia względnego *wb jest równa wartości wyrażenia postaci

```
      wb+a
```

gdzie a jest adresem przekładu rozkazu, słowa lub bajtu zawierającego dane wyrażenie symboliczne. Na przykład ciąg rozkazów

```
      BNN #1,*8
      BSC CSET
      LD #1,INDEX
```

działa tak samo jak ciąg

```
      BNN #1,LAB
      BSC CSET
LAB:  LD #1,INDEX
```

a dyrektywa

```
      WORD *
```

oznacza to samo, co

```
      PAR:WORD PAR
```

(p. 3.10) i powoduje zapamiętanie słowa o wartości równej jego adresowi.

3.8. Ogólna postać programu i symbole końca słowa

Podstawowymi słowami programu w języku ASM są rozkazy i dyrektywy. Podział ten jest czysto umowny. Mówiąc o rozkazach mamy na myśli symboliczne zapisy słów maszynowych, które na ogół mają być interpretowane jako rozkazy maszynowe, nie wykluczamy jednak potraktowania rozkazów jako danych; rozkaz ma zawsze przekład w postaci jednego lub dwóch słów maszynowych (p. 1.4). Przez dyrektywy rozumiemy bądź pewne informacje dla translatora nie mające bezpośredniego odpowiednika w przekładzie programu, bądź też symboliczne zapisy danych - nie wykluczając jednak potraktowania ich jako rozkazów maszyny.

Ogólna postać rozkazów i dyrektyw jest taka sama: są to napisy postaci

$$k\ p_1, p_2, \ldots, p_n$$

($n \geq 0$), gdzie k oznacza kod rozkazu lub dyrektywy; najczęściej jest to nazwa ustalona dla danego rozkazu lub dyrektywy. Symbole $p_1, p_2, \ldots, p_n$ są parametrami odpowiednimi dla danego rozkazu lub dyrektywy. W przypadku rozkazów cały ciąg parametrów nazywamy częścią adresową (lub krócej: adresem); ciąg ten zawiera co najwyżej trzy parametry odpowiadające częściom RA, RM i N słowa rozkazowego (p. 1.4). Część M słowa rozkazowego określa się pośrednio na podstawie postaci części adresowej (por. np. p. 2.9)

Program w języku ASM jest zawsze pewnym ciągiem dyrektyw i rozkazów - ew. poprzedzonych etykietami. Tak zbudowane elementy nazywamy słowami (prostymi) programu. Po każdym słowie musi wystąpić co najmniej jeden symbol końca słowa programu. Dopuszczalnymi i syntaktycznie równoważnymi symbolami końca słowa są:

- zmiana wiersza (p. 3.2),
- średnik,
- komentarz.

Komentarz jest ciągiem znaków rozpoczynających się od nawiasu kwadratowego otwierającego [i zakończonym znakiem zmiany wiersza.

Tak więc fragment programu

```
ADDC #1,VAL
BCC CLEAR
BN #1,NEGREG1
```

można również napisać w postaci

```
ADDC #1,VAL; BCC CLEAR; BN #1,NEGREG1
```

albo też w postaci

```
ADDC #1,VAL; BCC CLEAR [IF C=0, GO TO CLEAR
BN #1,NEGREG1 [IF R1<0, GO TO NEGREG1
```

bez zmiany znaczenia (działania) tego fragmentu.

Zbyteczne symbole końca słowa są pomijane; umożliwia to m.in. podzielenie tekstu programu na stronice i napisanie ciągu komentarzy na początku programu lub jakiegoś podprogramu.

Rozmieszczenie rozkazów (p. 3.9) i danych (p. 3.10 - 3.11) w pamięci operacyjnej należy do autora programu: translator produkuje przekład programu, w którym kolejność przekładów rozkazów i innych słów symbolicznych mających przekład odpowiada dokładnie ich kolejności w tekście źródłowym, z uwzględnieniem podanych w programie adresów początków segmentów (p. 3.15).

3.9. *Rozkazy*

Rozkazami w języku ASM nazywamy symboliczne zapisy słów rozkazowych maszyny (p. 1.4). Rozkaz ma jedną z dwóch postaci:

```
kr
kr adr
```

gdzie kr oznacza kod rozkazu (część operacyjną), a adr oznacza adres rozkazu (jego część adresową).

Kod rozkazu może być:

- jedną z nazw podanych w tabeli kodów rozkazów (por. Dodatek 1),
- dwucyfrową liczbą ósemkową (p. 3.3) reprezentującą sześć bitów części operacyjnej słowa rozkazowego maszyny.

Kod ósemkowy rozkazu dopuszczono w celu umożliwienia zapisania rozkazów, które w chwili projektowania języka ASM nie miały jeszcze określonej treści.

Część adresowa rozkazu (adr) ma postać 1-, 2- lub 3-elementowego ciągu parametrów oddzielonych przecinkami określających jedną lub więcej spośród części RA, RM oraz N słowa rozkazowego. Część M słowa określa się

pośrednio - bądź na podstawie kodu rozkazu, bądź też na podstawie postaci części adresowej; wynika to m.in. z tabeli podanej w p. 1.4 i pewnych naturalnych reguł, które sugeruje poniższa tabela.

Symboliczne zapisy części RA i RM słowa rozkazowego na ogół mają postać napisu 2-znakowego

#c

gdzie c jest cyfrą szesnastkową (p. 3.3) oznaczającą numer rejestru maszyny; napis tej postaci nazywamy oznaczeniem rejestru. Wszystkie dopuszczalne postacie części adresowej wymieniono w podanej niżej tabeli, w której

- c, d oznaczają cyfry szesnastkowe,
- ws oznacza wyrażenie symboliczne w sensie p. 3.7.

Oto tabela:

Postać części adresowej	Przykład rozkazu	Związek rozkazu ze słowem rozkazowym
#c,#d	ADD #1,#A	RA = 1, RM = 10, M = 0
#c,+#d	ADD #1,+#A	RA = 1, RM = 10, M = 1
#c,ws	ADD #1,VAR	RA = 1, N = VAR, M = 2
#c,#d,ws	ADD #1,#A,VAR	RA = 1, RM = 10, N = VAR, M = 2
#c,+#d,ws	ADD #1,+#A,VAR	RA = 1, RM = 10, N = VAR, M = 3
#c	SETS #B	RA = 11, RM = 0, M = 0
+#c	JUMP +#1	RM = 1, M = 1
ws	JUMP LAB	N = LAB, M = 2
+#c,ws	JUMP +#2,LAB	RM = 2, N = LAB, M = 3
	SV	RA = 0

Trzeci i czwarty wiersz tabeli zawierają rozkazy o identycznych skutkach wykonania. Czwarta postać adresu umożliwia określenie nieistotnej dla treści niektórych rozkazów części RM, co jednak może być użyteczne, kiedy chcemy zapisać jakieś dane w postaci ciągu rozkazów (przykładem może być tablica w translatorze jakiegoś języka programowania).

Adres pierwszego bajtu przekładu rozkazu jest zawsze parzysty; ta własność translatora jest związana ze sposobem adresowania rozkazów w maszynie (p. 1.3).

3.10. *Bajty, teksty i słowa*

Do umieszczenia w programie ciągu kolejnych bajtów służy dyrektywa postaci

BYTE d_1, d_2,..., d_n

($n \geq 1$), równoważna ciągowi dyrektyw

BYTE d_1; BYTE d_2;...; BYTE d_n

Każdy z zapisów d_1, d_2,..., d_n określa wartości pewnej liczby kolejnych bajtów i ma jedną z trzech postaci:

ws

$ws_1(ws_2)$

"z_1 z_2 ... z_k"

gdzie:

- ws, ws_1, ws_2 są wyrażeniami symbolicznymi (p. 3.7),
- z_1, z_2,..., z_k są znakami alfabetu (p. 3.1), z wyjątkiem znaku " oraz znaku nowej linii.

Dyrektywa postaci

BYTE ws

oznacza jeden bajt określony za pomocą ośmiu najmniej znaczących bitów wartości wyrażenia symbolicznego ws. Jeżeli $b = ws_1$ i $m = ws_2$ są wartościami wyrażeń symbolicznych ws_1 i ws_2, to dyrektywa postaci

BYTE $ws_1(ws_2)$

jest skróconym zapisem dyrektywy

BYTE $\underbrace{b, b, \ldots, b}_{m \text{ razy}}$

Dyrektywa

BYTE "z_1 z_2 ... z_k"

oznacza to samo co

BYTE 'z_1','z_2',...,'z_k'

(por. p. 3.3).

Napis postaci

$"z_1\ z_2\ \ldots\ z_k"$

nazywamy tekstem. Tekstu można użyć również niezależnie jako słowa symbolicznego programu. Jednak przekład dyrektywy postaci

BYTE $"z_1\ z_2\ \ldots\ z_k"$

różni się od przekładu słowa symbolicznego

$"z_1\ z_2\ \ldots\ z_k"$

tym, że przekład dyrektywy zajmuje dokładnie k bajtów pamięci, a przekład słowa o postaci tekstu zajmuje zawsze k+1 bajtów, mianowicie k bajtów reprezentujących kolejno znaki $z_1, z_2, \ldots, z_k$ oraz dodatkowy bajt o wartości 255 (H'FF) w roli symbolu końca przekładu tekstu.

Dyrektywa postaci

WORD $s_1, s_2, \ldots, s_n$

(n≥1) jest równoważna ciągowi dyrektyw:

WORD s_1;WORD s_2;...;WORD s_n

i służy do umieszczenia w programie ciągu słów. Każdy z napisów $s_1, s_2, \ldots, s_n$ może mieć jedną z dwóch postaci:

ws

$ws_1(ws_2)$

których znaczenie jest analogiczne jak w omówionej poprzednio dyrektywie BYTE; jedyna różnica polega na tym, że zapamiętuje się wartości 16-bitowe.

Do rezerwacji miejsca na tablice bajtów lub słów służą dyrektywy postaci

BS ws

WS ws

gdzie ws jest wyrażeniem symbolicznym. Są one równoważne odpowiednio dyrektywom:

BYTE Ø(ws)

WORD Ø(ws)

Jeżeli wyrażenie symboliczne występujące w dyrektywie o kodzie BS,WS lub BYTE,WORD z napisem $ws_1(ws_2)$ zawiera nazwy, to każdej z nich trzeba nadać wartość we wcześniejszej części programu.

3.11. *Pakowanie danych*

Do zapisania bajtów lub słów, w których znamy wartości grup kolejnych bitów mogą oczywiście służyć dyrektywy o kodach BYTE i WORD z odpowiednio dobranymi stałymi (p. 3.3, 3.4 i 3.10). Na ogół wygodniej jest użyć w takim przypadku dyrektywy o kodzie PACK i postaci

PACK(w) $v_1,v_2,\ldots,v_n$

($n \geq 1$). Napis w określa sposób pakowania (wzorzec) i ma postać

$ws_1/ws_2/\ldots/ws_k$

gdzie ws_i ($i = 1, 2, \ldots, k$) są wyrażeniami symbolicznymi (p. 3.7) takimi, że

$$d = ws_1+ws_2+\ldots+ws_k = 8$$

albo

$$d = ws_1+ws_2+\ldots+ws_k = 16$$

Wartości tych wyrażeń oznaczają długości kolejnych grup bitów bajtu (dla $d = 8$) lub słowa (dla $d = 16$) przeznaczonych do zapamiętania wartości określonych za pomocą odpowiednich części napisów v_i ($i = 1, 2, \ldots, k$). Każdy z napisów v_i również ma postać

$ws_1/ws_2/\ldots/ws_k$

przy czym jednak wartości $ws_1, ws_2, \ldots, ws_k$ interpretuje się w tym kontekście jako wartości kolejnych grup bitów. Przykładowa dyrektywa

PACK(2/3/3) 1/5/4,0/1/3,3/7/4

oznacza więc ciąg trzech bajtów, który można w równoważny sposób określić za pomocą dyrektywy

BYTE 8!154,8!13,8!374

Jeżeli jakieś wyrażenie symboliczne występujące w dyrektywie o kodzie PACK zawiera nazwy, to każdej z nich trzeba nadać wartość we wcześniejszej części programu. Tak więc przykładowy fragment programu

```
PACK(M/N/Q/V) 2/4/5/3,A+B/A-B/A/3
M:=5
N:=6
Q:=3
V:=16-M-N-Q
```

(p. 3.6) jest błędny; dyrektywę PACK trzeba w tym przypadku napisać po dyrektywach nadających wartość nazwom M, N, Q oraz V. Nazwom A i B użytym w zapisie wartości grup bitów również trzeba nadać wartość we wcześniejszej części programu.

3.12. *Bloki względne*

Sposób nadawania wartości nazwom użytym w roli etykiet omówiono w p. 3.6. Niekiedy jest niecelowe wiązanie wartości nazwy z adresem bajtowym początku przekładu instrukcji lub dyrektywy. Przykładem może być zastosowanie premodyfikacji (p. 1.2) w odwołaniach do zmiennych lub konstrukcja fragmentów programu, np. podprogramów, które mają działać poprawnie w dowolnym obszarze pamięci operacyjnej (przy zastosowaniu modyfikacji we wszystkich rozkazach skokowych). Do ułatwienia takich konstrukcji służą tzw. bloki względne. Każdy blok względny ma postać

```
REL
Pcs P
EREL
```

gdzie $P_{cs}P$ jest ciągiem słów symbolicznych (p. 3.8 - 3.11). Dyrektywa REL rozpoczynająca blok względny oznacza zmianę zwykłego sposobu przyporządkowywania adresów; po przeczytaniu tej dyrektywy nie zmienia się sposób rozmieszczania przekładu w pamięci (kolejne bajty o kolejno rosnących adresach, poczynając od adresu podanego w dyrektywie LOC - p. 3.15), ale adresy przyporządkowywane kolejnym bajtom przekładu liczy się od zera. W efekcie nazwom S, T oraz U użytym w przykładowym bloku względnym

```
REL
S:BS 100 [p. 3.10
T:WS 500 [p. 3.10
U:WS 100
EREL
```

translator przyporządkuje adresy równe odpowiednio 0, 100, 1100.

Dyrektywa EREL unieważnia działanie dyrektywy REL, a zatem jest symbolem końca bloku względnego.

Każdy blok względny można poprzedzić etykietą. Takiej etykiecie translator przyporządkuje adres w zwykły sposób, a zatem można go użyć w rozkazie SETM (p. 2.17).

3.13. *Dyrektywy wyrównania adresów przekładu*

Jeżeli mamy w programie dyrektywę w rodzaju

```
A:BYTE 5,2,3
```

to interpretacja rozkazu

```
LD #1,A
```

nie jest jednoznaczna dla autora programu, ponieważ zależy od adresu przyporządkowanego nazwie A (p. 1.3). Dyrektywa

```
EVEN
```

(bez parametrów) powoduje, że adres początku przekładu następującego po tej dyrektywie słowa programu będzie parzysty; w razie potrzeby translator pozostawi niewykorzystany bajt po przekładzie poprzedniego fragmentu programu. Tak więc pisząc podany tutaj fragment programu w postaci

```
EVEN
A:BYTE 5,2,3
```

uzyskujemy jednoznaczną interpretację przykładowego rozkazu LD #1,A.

Stosowanie dyrektywy EVEN przed słowami, których przekład na mocy ich definicji zajmuje parzystą liczbę bajtów, jest zbyteczne, ponieważ w takich przypadkach translator zawsze umieszcza przekład słowa poczynając od bajtu o adresie parzystym.

Dyrektywa PREM ma analogiczne znaczenie jak dyrektywa EVEN, w tym przypadku następuje wyrównanie adresu do najbliższej wielokrotności liczby 16. Dyrektywę PREM stosuje się więc w razie potrzeby przed blokiem względnym (p. 3.12 i 1.2).

3.14. *Poziomy oznaczeń*

Rolę nazw w programie omówiono już w p. 3.6. Znaczenie każdej nazwy może być globalne, tj. obowiązujące w całym programie lub lokalne, tj. obowiązujące w pewnym fragmencie programu, zwanym blokiem. Postać bloków oraz sposoby odróżniania nazw globalnych od lokalnych omówiono w dalszym ciągu tego punktu. Lokalizacja znaczenia nazw służy przede wszystkim do umożliwienia pisania różnych fragmentów programu przez różne osoby bez konieczności uzgadniania wszystkich używanych nazw. W związku z tym mówimy o poziomach oznaczeń: na każdym poziomie lokalnym można swobodnie uży-

wać nazw lokalnych - niezależnie od sposobu używania takich samych nazw na innych poziomach oznaczeń. W każdym programie może wystąpić co najwyżej jeden globalny poziom oznaczeń i dowolnie wiele niezależnych lokalnych poziomów oznaczeń. Lokalnych poziomów oznaczeń nie można zanurzać (np. w sensie ALGOLu 60), ponieważ żaden blok programu nie może zawierać innego bloku.

Istnieją dwa rodzaje bloków wprowadzających lokalny poziom oznaczeń. Blok pierwszego rodzaju ma postać

```
BL nb
  e1;e2;...;en;
  csw
EBL
```

a blok drugiego rodzaju ma postać

```
BLG nb
  e1;e2;...;en
  csw
EBLG
```

gdzie

nb jest napisem o postaci dowolnej nazwy, traktowanym jako komentarz (obowiązkowy); można go wykorzystać do identyfikacji bloku,

$e_1, e_2, \ldots, e_n$ (dla $n \geq 0$) jest opcjonalnym ciągiem opisów tzw. nazw zewnętrznych; znaczenie opisu nazw zewnętrznych omówiono dalej; średnik oddzielający poszczególne opisy można zastąpić innym symbolem końca słowa (p. 3.8),

csw jest ciągiem słów programu (p. 3.8 - 3.11) lub bloków względnych (p. 3.12).

Każdy blok można poprzedzić etykietą.

Dyrektywy BL i EBL oraz BLG i EBLG stanowią informację dla translatora odpowiednio o początku i końcu bloku pierwszego rodzaju oraz bloku drugiego rodzaju; określimy je w tej kolejności.

Jeżeli blok pierwszego rodzaju nie zawiera ciągu opisów nazw zewnętrznych $e_1, e_2, \ldots, e_n$ (czyli $n = 0$), to wszystkie nazwy użyte we wnętrzu tego bloku są w nim lokalne i muszą być w nim zdefiniowane, tj. użyte w roli etykiet (p. 3.6); nazwy lokalne w bloku nie mają żadnego znaczenia poza blokiem.

Bloki pierwszego rodzaju nie zawierające opisów nazw zewnętrznych służą przede wszystkim do konstrukcji podprogramów nie odwołujących się do żadnych innych bloków ani do programu głównego. Przykładowy blok tego rodzaju może mieć postać:

```
SUBRSQRT:BL SQRT   [obliczanie SQRT(#A)
            ...
            EBL
```

Opis nazw zewnętrznych ma postać dyrektywy o kodzie EXT, mianowicie

EXT $n_1, n_2, \ldots, n_k$

($k \geq 1$), gdzie $n_1, n_2, \ldots, n_k$ są nazwami. Umieszczenie na początku bloku opisów nazw zewnętrznych oznacza, że wymienione w tych opisach nazwy nie są lokalne, tj. są dostępne na zewnątrz bloku; nazwy te oczywiście muszą być użyte w roli etykiet, ale niekoniecznie w tym samym bloku i niekoniecznie w jakimś innym bloku: może to być globalny poziom oznaczeń.

Wynika stąd, że jeżeli mamy pewną liczbę bloków pierwszego rodzaju i w każdym z nich pewna nazwa ma być używana w takim samym znaczeniu, to w każdym z tych bloków trzeba tę nazwę opisać jako zewnętrzną. Jeżeli liczby nazw i bloków o wymienionych własnościach są duże, to używanie bloków pierwszego rodzaju staje się niewygodne ze względu na konieczność wielokrotnego powtarzania tych samych opisów nazw zewnętrznych. Niedogodności tej można uniknąć stosując bloki drugiego rodzaju (rozpoczynające się dyrektywą BLG i zakończone dyrektywą EBLG).

Z blokami drugiego rodzaju są związane opisy nazw globalnych. Każdy opis nazw globalnych ma postać dyrektywy o kodzie GLOB, mianowicie

GLOB $n_1, n_2, \ldots, n_k$

($k \geq 1$), gdzie $n_1, n_2, \ldots, n_k$ są nazwami. Opis nazw globalnych można umieścić tylko poza blokiem. Własności bloków drugiego rodzaju i opisów nazw globalnych są następujące:

- jeżeli poprzedzająca blok drugiego rodzaju część programu nie zawiera żadnego opisu nazw globalnych albo opis ten nie zawiera nazw używanych w bloku, to blok drugiego rodzaju zachowuje się tak samo jak blok pierwszego rodzaju,
- w przeciwnym razie nazwy wymienione w opisie nazw globalnych i użyte w bloku zachowują się tak jak gdyby były wymienione w opisie nazw zewnętrznych (EXT ...) zawartym w bloku drugiego rodzaju.

Tak więc fragment programu postaci:

```
BL B1
EXT EN1,EN2
.....
EBL [koniec B1
BL B2
EXT EN1,EN2
.....
EBL [koniec B2
BL B3
EXT EN3
.....
EBL [koniec B3
```

można zastąpić przez:

```
GLOB EN1,EN2
BLG B1
.....
EBLG
BLG B2
.....
EBLG
BL
EXT EN3
.....
EBL
```

Jedyna istotna różnica pomiędzy blokami dwóch wymienionych rodzajów polega na tym, że znaczenie nazw użytych w bloku pierwszego rodzaju nie zależy od poprzedzających ten blok opisów nazw globalnych.

Globalny poziom oznaczeń tworzą następujące nazwy:

- nazwy użyte w roli etykiet przed słowami programu nie należącymi do wnętrza żadnego bloku pierwszego lub drugiego rodzaju,
- nazwy wymienione w opisie nazw zewnętrznych,
- nazwy wymienione w opisie nazw globalnych.

Pozostałe nazwy należą do któregoś z lokalnych poziomów oznaczeń wprowadzonego za pomocą bloku pierwszego lub drugiego rodzaju - w zależności od tego, gdzie zostały użyte jako etykiety.

3.15. *Segmenty i symbol końca programu*

Dyrektywę postaci

```
LOC ws
```

gdzie ws jest wyrażeniem symbolicznym (p. 3.7), można umieścić przed ciągiem słów (p. 3.8 - 3.13) lub bloków (p. 3.13) programu. Dyrektywa oznacza, że przekład tekstu symbolicznego następującego po tej dyrektywie translator ma przygotować do umieszczenia (w wyniku wprowadzenia przekładu binarnego) poczynając od bajtu o adresie równym wartości wyrażenia ws. Jeżeli wyrażenie ws zawiera nazwy, to każdej z nich trzeba nadać wartość we wcześniejszej części programu.

Segmentem w języku ASM nazywamy fragment programu zawarty pomiędzy dwoma kolejnymi dyrektywami o kodzie LOC lub pomiędzy ostatnią taką dyrektywą, a symbolem końca programu, którym jest dyrektywa o kodzie EP. Dyrektywa końca programu ma jedną z dwóch postaci

```
EP
EP ws
```

gdzie ws jest wyrażeniem symbolicznym, którego wartość oznacza adres pierwszego rozkazu do wykonania; adres ten wykorzystuje się do uruchomienia programu natychmiast po wprowadzeniu.

3.16. *Program*

Program w języku ASM zawiera kolejno następujące części:

- tzw. blok żądań (omówiony dalej),
- ciąg segmentów (p. 3.15), każdy poprzedzony dyrektywą o kodzie LOC,
- dyrektywę o kodzie EP oznaczającą symbol końca programu (por. p. 3.15).

Szczegóły dotyczące postaci i znaczenia bloku żądań zależą od stosowanej wersji translatora języka ASM i są określone w opisie każdego translatora. Zgodnie z formalną gramatyką języka ASM blok żądań musi wystąpić na początku programu. Ogólną formą syntaktyczną bloku żądań jest napis postaci:

```
REQ
  tb
EREQ
```

gdzie tb jest treścią bloku żądań, której składnia jest taka sama jak

treści segmentu, tj. napisu następującego po dyrektywie o kodzie LOC (p. 3.15). W konkretnym translatorze lub systemie operacyjnym treść bloku żądań może mieć mniej ogólną postać.

element składniowy	symbol	definicja
litera	⟨l⟩	A | B | ... | Z
cyfra	⟨c⟩	0 | 1 | ... | 9
znak widoczny i odstęp	⟨zwo⟩	⟨l⟩ | ⟨c⟩ | ! | " | # | ...
cyfra dwójkowa	⟨c2⟩	0 | 1
cyfra ósemkowa	⟨c8⟩	0 | 1 | ... | 7
cyfra szesnastkowa	⟨c16⟩	⟨c⟩ | A | B | C | D | E | F
nazwa	⟨n⟩	⟨l⟩ | ⟨n⟩⟨l⟩ | ⟨n⟩⟨c⟩
liczba naturalna	⟨ln⟩	⟨c⟩ | ⟨ln⟩⟨c⟩
liczba dwójkowa	⟨l2⟩	2!⟨c2⟩ | ⟨l2⟩⟨c2⟩
liczba ósemkowa	⟨l8⟩	8!⟨c8⟩ | ⟨l8⟩⟨c8⟩
liczba szesnastkowa	⟨l16⟩	H!⟨c16⟩ | ⟨l16⟩⟨c16⟩
kod znaku	⟨kz⟩	'⟨zwo⟩' | '⟨zwo⟩⟨zwo⟩'
liczba całkowita bez znaku	⟨lcbz⟩	⟨ln⟩ | ⟨l2⟩ | ⟨l8⟩ | ⟨l16⟩ | ⟨kz⟩
liczba całkowita	⟨lc⟩	⟨lcbz⟩ | -⟨lcbz⟩
rejestr	⟨r⟩	#⟨c16⟩
etykieta	⟨e⟩	⟨n⟩:
wyrażenie symboliczne bezwzględne	⟨wsb⟩	⟨lc⟩ | ⟨n⟩ | -⟨n⟩ | ⟨wsb⟩+⟨lcbz⟩ | ⟨wsb⟩-⟨lcbz⟩ | ⟨wsb⟩+⟨n⟩ | ⟨wsb⟩-⟨n⟩
wyrażenie symboliczne	⟨ws⟩	⟨wsb⟩ | * | *⟨wsb⟩
część adresowa	⟨ca⟩	⟨r⟩ | +⟨r⟩ | ⟨ws⟩ | ⟨r⟩,⟨r⟩ | ⟨r⟩,+⟨r⟩ | ⟨r⟩,⟨ws⟩ | +⟨r⟩,⟨ws⟩ | ⟨r⟩,⟨r⟩,⟨ws⟩ | ⟨r⟩,+⟨r⟩,⟨ws⟩
kod rozkazu	⟨kr⟩	⟨n⟩ | ⟨18⟩
rozkaz	⟨rozk⟩	⟨kr⟩ | ⟨kr⟩⟨ca⟩
tekst	⟨t⟩	"⟨dowolny ciąg znaków bez znaku zmiany wiersza⟩"
nadanie wartości	⟨ndw⟩	⟨e⟩=⟨ws⟩
ciąg wartości	⟨cw⟩	⟨ws⟩ | ⟨ws⟩(⟨ws⟩)
ciąg bajtów	⟨cb⟩	BYTE ⟨cw⟩ | BYTE ⟨t⟩ | ⟨cb⟩,⟨cw⟩ | ⟨cb⟩,⟨t⟩
ciąg słów	⟨cs⟩	WORD ⟨cw⟩ | ⟨cs⟩,⟨cw⟩
rezerwacja ciągu bajtów	⟨rcb⟩	BS ⟨ws⟩
rezerwacja ciągu słów	⟨rcs⟩	WS ⟨ws⟩
wyrównanie adresu do słowa	⟨was⟩	EVEN
wyrównanie adresu do premodyfikacji	⟨wapm⟩	PREM
układ bitów	⟨ub⟩	⟨ws⟩ | ⟨ub⟩/⟨ws⟩
ciąg układów bitów	⟨cbit⟩	PACK(⟨ub⟩)⟨ub⟩ | ⟨cbit⟩,⟨ub⟩
puste	⟨⟩	
słowo programu	⟨sp⟩	⟨⟩ | ⟨rozk⟩ | ⟨t⟩ | ⟨ndw⟩ | ⟨cb⟩ | ⟨cs⟩ | ⟨rcb⟩ | ⟨rcs⟩ | ⟨was⟩ | ⟨wapm⟩ | ⟨cbit⟩
komentarz	⟨kom⟩	[⟨ciąg znaków zakończony znakiem zmiany wiersza⟩
koniec słowa programu	⟨ksp⟩	; | ⟨kom⟩ | ⟨zmiana wiersza⟩
prosty ciąg słów programu	⟨pcsp⟩	⟨e⟩ | ⟨sp⟩⟨ksp⟩ | ⟨pcsp⟩⟨e⟩ | ⟨pcsp⟩⟨sp⟩⟨ksp⟩
blok względny	⟨bw⟩	REL ⟨ksp⟩⟨pcsp⟩ EREL ⟨ksp⟩
ciąg słów programu	⟨csp⟩	⟨pcsp⟩ | ⟨bw⟩ | ⟨csp⟩⟨pcsp⟩ | ⟨csp⟩⟨bw⟩
opis nazw globalnych	⟨ong⟩	GLOB ⟨n⟩ | ⟨ong⟩,⟨n⟩
opis nazw zewnętrznych	⟨onz⟩	EXT ⟨n⟩ | ⟨onz⟩,⟨n⟩

początek bloku globalnego
blok globalny
początek bloku
blok
treść segmentu

segment
ciąg segmentów
początek programu
koniec programu
program

⟨pbg⟩ BLG ⟨n⟩⟨ksp⟩ | ⟨pbg⟩⟨onz⟩⟨ksp⟩
⟨bg⟩ ⟨pbg⟩⟨csp⟩ EBLG ⟨ksp⟩
⟨pb⟩ BL ⟨n⟩⟨ksp⟩ | ⟨pb⟩⟨onz⟩⟨ksp⟩
⟨b⟩ ⟨pb⟩⟨csp⟩ EBL ⟨ksp⟩
⟨tseg⟩ ⟨csp⟩ | ⟨ong⟩ | ⟨bg⟩ | ⟨b⟩ | ⟨tseg⟩⟨csp⟩ | ⟨tseg⟩⟨ong⟩ | ⟨tseg⟩⟨bg⟩ | ⟨tseg⟩⟨b⟩
⟨seg⟩ LOC ⟨ws⟩⟨ksp⟩⟨tseg⟩
⟨cseg⟩ ⟨seg⟩ | ⟨cseg⟩⟨seg⟩
⟨ppr⟩ REQ ⟨ksp⟩⟨tseg⟩ EREQ ⟨ksp⟩
⟨kpr⟩ EP ⟨ksp⟩ | EP ⟨ws⟩⟨ksp⟩
⟨pr⟩ ⟨ppr⟩⟨cseg⟩⟨kpr⟩

Dodatek 1. Lista rozkazów

Typy adresów:
1. ⟨R⟩,⟨R⟩
2. ⟨R⟩,+⟨R⟩
3. ⟨R⟩,⟨WS⟩
4. ⟨R⟩,⟨R⟩,⟨WS⟩
5. ⟨R⟩, +⟨R⟩,⟨WS⟩
6. ⟨R⟩
7. +⟨R⟩
8. ⟨WS⟩
9. +⟨R⟩,⟨WS⟩
10. ⟨⟩

kod symb.	wartość M	R	typy adresów	treść rozkazu	str.
ADD	0 1 2 3	15	1-5	dodawanie	20
ADDC	0 1 2 3	12	1-5	dodawanie z uwzględnieniem C	20
ADDN	0	52	3	dodawanie stałej (RM)	21
AND	0 1 2 3	01	1-5	mnożenie logiczne	22
BC⟨X⟩	1 2 3	61	7-9	skok, jeśli bit X=0	24
BN	1 2 3	42	2-5	skok, jeśli ra<0	24
BNN	1 2 3	43	2-5	skok, jeśli ra>=0	24
BS⟨X⟩	1 2 3	62	7-9	skok, jeśli bit X=1	24
BUZ	1 2 3	64	2-5	zmniejszenie o 1 i skok przy wyniku różnym od zera	24
BZ	1 2 3	41	2-5	skok, jeśli ra=0	24
CALL	1 2 3	50	2-5	skok z zapamiętaniem śladu	24
CBIT	0	41	3	zerowanie bitu RA w RM	31
CLS	1 2 3	55	3,7,9	zerowanie RM+1 słów	34
CNVB	1 2 3	46	2-5	konwersja bajtu	30
COMB	1 2 3	25	2-5	porównanie bajtów	26
COMP	0 1 2 3	05	1-5	porównanie słów	26
C⟨X⟩	0	61	10	zerowanie bitu X w słowie stanu procesora	35
DIV	1 2 3	11	2-5	dzielenie	27
JUMP	1 2 3	60	7-9	skok bezwarunkowy	24
LBLS	1 2 3	22	2-5	pobranie do prawego bajtu (lewy b.z.)	30
LBLZ	1 2 3	20	2-5	pobranie do prawego bajtu (lewy Ø)	30
LBRS	1 2 3	24	2-5	pobranie do lewego bajtu (prawy b.z.)	30
LD	0 1 2 3	00	1-5	pobranie do rejestru	20
LDA	1 2 3	40	2-5	pobranie adresu do rejestru	21
LDN	0	40	3	pobranie stałej do rejestru	21
LDR	0	50	1	pobranie rejestru pomocniczego	37
LLS	0	24	1	lewy bajt na lewy, prawy b.z.	29
LSL	0	22	1	lewy bajt na prawy, lewy b.z.	29
LSR	0	23	1	prawy bajt na prawy, lewy b.z.	29
LZL	0	20	1	lewy bajt na prawy, lewy Ø	29
LZR	0	21	1	prawy bajt na prawy, lewy Ø	29
MODA	0	26	1,6	modyfikacja numeru RA	28
MODM	0	27	1,6	modyfikacja numeru RM	28
MODW	0	25	1,6	modyfikacja rozkazu	28
MPY	1 2 3	10	2-5	mnożenie	27

MVRS		1	2	3	56	2,4-5	przepisywanie rejestrów do pamieci
MVSR		1	2	3	57	2,4-5	przepisywanie pamięci do rejestrów
NBIT	0				43	3	negacja bitu w rejestrze
NGA	0	1	2	3	17	1-5	negacja arytmetyczna
NGAC	0	1	2	3	14	1-5	negacja arytm. z uwzgl. C
NGL	0	1	2	3	04	1-5	negacja logiczna
NULL	0				77	1,10	pusta treść
N⟨X⟩	0				63	10	negacja bitu X w słowie stanu procesora
OR	0	1	2	3	02	1-5	suma logiczna
RDIR		1	2	3	53	2-5	czytaj bezpośrednio
RET	0				67	10	powrót według stosu
RET		1	2	3	67	7-9	powrót według adresu
RKEY		1	2	3	51	2-5	czytaj klucz
RST		1	2	3	63	2-5	czytaj status
SAL	0				32	1	przesuwanie arytm. w lewo o 1 bit
	0				72	3	przesuwanie arytm. w lewo o RM bitów
SAR	0				36	1	przesuwanie arytm. w prawo o 1 bit
	0				76	3	przesuwanie arytm. w prawo o RM bitów
SBIT	0				42	3	ustawianie bitu w rejestrze
SCL	0				30	1	przesuwanie cykliczne w lewo o 1 bit
	0				70	3	przesuwanie cykliczne w lewo o RM bitów
SCR	0				34	1	przes. cykl. w prawo o 1 bit
	0				74	3	przes. cykl. w prawo o RM bitów
SEB		1	2	3	47	2-5	przeszukanie łańcucha bajtów
SETM	0				06	6	ustawienie premodyfikatora
SETS	0				56	6	ustawienie lewego bajtu w słowie stanu procesora
SLL	0				31	1	przes. log. w lewo o 1 bit
	0				71	3	przes. log. w lewo o RM bitów
SLR	0				35	1	przes. log. w prawo o 1 bit
	0				75	3	przes. log. w prawo o RM bitów
ST		1	2	3	44	2-5	pamiętanie
STB		1	2	3	45	2-5	pamiętanie bajtu
STR	0				51	1	pamiętanie rejestrów w rejestrach pom.
STOP	0				60	6	zatrzymanie programu
SUB	0	1	2	3	16	1-5	odejmowanie
SUBC	0	1	2	3	13	1-5	odejmowanie z uwzględnieniem C
SUBN	0				53	3	odejmowanie stałej
S⟨X⟩	0				62	10	ustawianie bitu X w słowie stanu procesora
TBIT	0				47	3	testowanie bitu
TS		1	2	3	54	2-5	pobranie z pamiętaniem 11...1
WDIR		1	2	3	52	2-5	pisz bezpośrednio
WKEY		1	2	3	65	2-5	pisz klucz
WST		1	2	3	66	2-5	pisz status
XOR	0	1	2	3	03	1-5	różnica symetryczna

Dodatek 3. Kody znaków

znak	DZM 180	CONSUL	CELLATRON	ARITMA 130	uwagi
		012			zmiana wiersza
	040	240	240	100	odstęp
!	041	041	041	117	
"	042	042	042	320	
#	043	243	243	173	
$	044	044	044	340	znak monetarny
%	045	245	245	154	
&	046	246	246	120	
'	047	047	047	116	
(	050	050	050	175	
)	051	251	251	176	
*	052	252	252	134	
+	053	053	053	172	
,	054	254	254	153	
-	055	055	055	140	
.	056	056	056	113	
/	057	257	257	141	
0	060	060	060	360	
1	061	261	261	361	
2	062	262	262	362	
3	063	063	063	363	
4	064	264	264	364	
5	065	065	065	363	
6	066	066	066	366	
7	067	267	267	367	
8	070	270	270	370	
9	071	071	071	371	
:	072	072	072	115	
;	073	273	273	114	
<	074	074	074	136	
=	075	275	275	156	
>	076	276	276	135	
?	077	077	077	155	
@	100	300	300	174	
A	101	101	101	301	
B	102	102	102	302	
C	103	303	303	303	
D	104	104	104	304	
E	105	305	305	305	
F	106	306	306	306	
G	107	107	107	307	
H	110	110	110	310	
I	111	311	311	311	
J	112	312	312	321	
K	113	113	113	322	
L	114	314	314	323	
M	115	115	115	324	
N	116	116	116	325	
O	117	317	317	326	
P	120	120	120	327	
Q	121	321	321	330	
R	122	322	322	331	
S	123	123	123	342	

T	124	324	324	343	
U	125	125	125	344	
V	126	126	126	345	
W	127	327	327	346	
X	130	330	330	347	
Y	131	131	131	350	
Z	132	132	132	351	
[	133	333	333	132	
\	134		134	133	ukośna kreska odwrotna
]	135	335	335	177	
↑	136	336	336	137	
←	137	137	137	157	