materiały szkoleniowe

139

PLAN
KURS PROGRAMOWANIA

(Część I)

z angielskiego przełożył
STEFAN BIEŃKOWSKI

CENTRALNY OŚRODEK DOSKONALENIA KADR KIEROWNICZYCH

CENTRALNY OŚRODEK DOSKONALENIA KADR KIEROWNICZYCH

PLAN
KURS PROGRAMOWANIA

(Część 1)

z angielskiego przełożył
STEFAN BIEŃKOWSKI

Warszawa Wrzesień 1971

Adres Redakcji Warszawa, ul. Wawelska 56 tel. 25-12-81 w. 96
Okładka S. NARGIELLO

Tytuł oryginału
PLAN CODING

CODKK Warszawa, ul. Wawelska 56
Pow. wł. zam. 169 wrzesień 1971 nakł. 375+10 egz.

S p i s t r e ś c i

Należy dokładnie przeczytać poniższe uwagi wstępne:

1. Celem tej serii, złożonej z czterech części, jest zapoznać Cię z techniką kodowania PLAN, t.j. nauczyć Cię biegłości w używaniu języka programowania - i nic poza tym.

2. Jeżeli spotykasz się z maszynami cyfrowymi po raz pierwszy, sugerujemy Ci, abyś przed przystąpieniem do tego kursu przeczytał tekst programowania I.C.T. "BASIC DIGITAL COMPUTER CONCEPTS".

3. Jeżeli potrzebujesz więcej informacji o ogólnej naturze maszyn serii 1900, jest również dostępny tekst programowania I.C.T. zwany "AN INTRODUCTION TO THE 1900 SERIES". Spis treści tej publikacji podany jest w dodatku D tej książki.

4. Dostarczamy sètki pytań i ćwiczeń, abyś mógł pracować nad nimi. Odpowiedzi, dla łatwiejszego ich odszukania, są umieszczone zwykle bezpośrednio po ćwiczeniu lub na następnej stronie. Sugerujemy Ci następujący sposób postępowania: najpierw rozwiąż problem, a następnie spójrz na rozwiązanie.

5. Nie próbuj zrobić zbyt wiele na raz. Szacujemy czas na 8 do 10 godzin na przerobienie każdej z czterech części. Osiągniesz jednak lepsze wyniki, rozkładając całą pracę na okres około 4 tygodni i poświęcając na nią nie więcej niż 2 godziny dziennie.

6. Książka ta nie jest podręcznikiem. Jest to podstawowy kurs szkoleniowy i dlatego uważaliśmy za ważniejsze ujęcie jakiegoś zagadnienia w jego rozwoju, zamiast obwarowywania tego zagadnienia przypisami o wyjątkach, ograniczeniach i t.p. Założyliśmy, że później, gdy będziesz już programował na 1900, będziesz mógł uzupełnić swoje wiadomości, zapoznając się z ostatnim wydaniem I.C.T. PLAN TRAINING MANUAL.

7. Przed rozpoczęciem tego kursu upewnij się czy masz dostateczną ilość arkuszy kodowania PLAN. Gdy popracujesz przez jakiś czas, będziesz mógł wykonywać prostsze ćwiczenia na kawałkach zwykłego papieru.

Co to jest PLAN ?

Rozkaz programowy serii 1900 przechowywany jest w pamięci operacyjnej jako zespół bitów zajmujący jedno 24-bitowe słowo.

Dowolnym przykładem może być:

1 0 1 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1 0 1 1 0 0 1

Jednostka centralna maszyny wie oczywiście, jak interpretować ten zespół. Oto jak rozdziela ona te 24 bity:

1 0 1	0 0 0 1	0 1 1	0 0	0 0 1 0 1 1 0 1 1 0 0 1

(1) (2) (3) (4)

1. Bity te podają numer akumulatora. Tutaj akumulator ma nr 5, a zwracanie się do niego oznacza się zwykle jako X5.

2. Te pozycje bitowe zawierają kod funkcji, t.j. co ma robić maszyna. W przykładzie tym kod 013 oznacza " przenieś z akumulatora do komórki pamięci operacyjnej".

3. Chwilowo pomijamy tę część rozkazu, gdyż wyjaśnienie jej zabrałoby zbyt wiele czasu i miejsca.

4. Jest to adres komórki pamięci operacyjnej.

Ogólnym wynikiem tego zespołu bitów jest przeniesienie przez maszynę zawartości X5 do komórki 729.

W 1900 te składowe części rozkazu znane są jako:

X	F	M	N
1 0 1	0 0 0 1	0 1 1	0 0	0 0 1 0 1 1 0 1 1 0 0 1

gdzie, mówiąc ogólnie,

F = funkcja, która ma być wykonana
X = numer akumulatota
M = modyfikator / więcej o nim będzie później/
N = adres operanda

Tak więc, gdyby nie było języka PLAN i rozkaz ten musiał być napisany w kodzie wewnętrznym maszyny, wówczas byłby on prawdopodobnie przedstawiony jako:

F	X	M	N
013	5	0	729

a cała strona kodowania mogłaby wyglądać następująco:

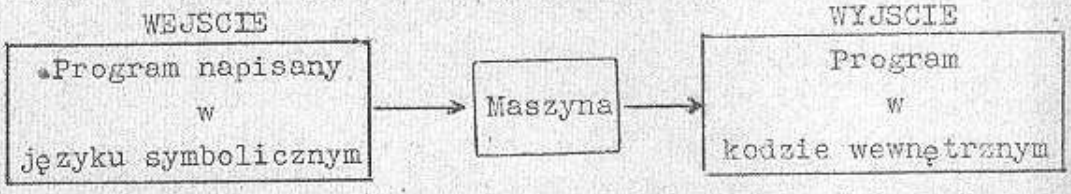
F	X	M	N
002	4	0	0719
001	4	0	0843
100	5	0	6927
003	0	1	5012
060	1	0	4030
011	7	3	3000
027	5	0	4030

Spróbuj pisać coś podobnego, a zobaczysz jakie to uciążliwe i jak łatwo o pomyłkę.

Idea języka symbolicznego /assembly language/ wypłynęła z konieczności przezwyciężenia trudności organizacyjnych związanych z kodem wewnętrznym maszyny. Celem jest pisanie programów w języku odpowiadającym programiście, a nie maszynie.

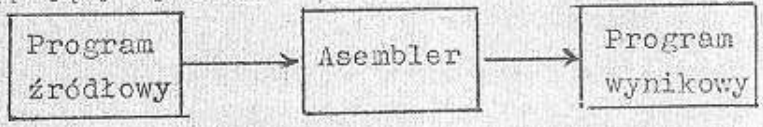
Gdy program napisany jest w języku symbolicznym, może on być następnie zmieniony na postać zrozumiałą dla maszyny. Ponieważ jest to proces bardzo złożony, najlepszym urządzeniem do przeprowadzenia zamiany jest sama maszyna.

Zamiana ta składa się zwykle z jednego lub dwóch przebiegów maszyny i daje następujący ogólny wynik:



Maszyna zawiera w tym czasie program dokonujący tej zamiany, zwany asemblerem /Assembler/.

Proces ten jest zwykle, bardziej zwięźle, przedstawiany w następujący sposób:



W książce tej interesuje nas jedynie jak pisać program źródłowy, używając PLAN /Programming Language Nineteen Hundred/.

Ale nie jest możliwe pisać program PLAN w sposób właściwy, o ile nie jest się świadomym tego, jak asembler traktuje jakieś zdanie PLAN i co pojawi się w programie wynikowym.

Na przykład, jeden rozkaz PLAN powoduje zazwyczaj generowanie jednego rozkazu w kodzie wewnętrznym maszyny. Rozkaz PLAN

ERX 5 DROOP

może być przetłumaczony jako

022 5 2174

Czasami rozkaz PLAN powoduje, że asembler generuje w programie wynikowym kilka rozkazów i programista musi być tego świadomy.

Oprócz rozkazów w kodzie wewnętrznym program wynikowy musi zawierać w sobie obszary pamięci zarezerwowane dla danych wejściowych, wierszy druku, obszary robocze, stałe arytmetyczne używane przez program i t.p.

Gdy program wynikowy jest załadowany i gotowy do realizacji, możemy go przedstawić w następujący sposób:

Obszar wejścia
Obszar wyjścia
Stałe
Rozkazy programu

Dlatego też język symboliczny oprócz tego, że jest środkiem pisania rozkazów programu zorientowanym na użytkownika, musi zawierać metody rezerwowania miejsca w obszarze pamięci operacyjnej programu wynikowego dla wszystkich tych elementów.

Następne strony opisują jak używać tego języka PLAN do pisania programu źródłowego i jaki skutek wywołuje poszczególne zdanie PLAN na tłumaczenie programu wynikowego.

Są trzy wersje PLAN-u:

PLAN 1 jest językiem podstawowym
PLAN 2 jest językiem podstawowym z dodatkowymi możliwościami
PLAN 3 jest językiem podstawowym z poszerzonymi dodatkowymi możliwościami.

Po przerobieniu kilkuset stron tej książki zobaczymy dopiero PLAN 2 i 3 i ich działanie. Do tego czasu będziemy zajmowali się wyłącznie PLANem 1 - językiem podstawowym.

Bez względu na wielkość 1900, na którą będziesz programował, musisz poznać przede wszystkim PLAN 1.

Dyrektywy i rozkazy programowe.

Rozkazy PLAN dzielą się na dwie zasadnicze grupy.

Rozkazy pierwszej grupy zwane są dyrektywami. Dyrektywy dotyczą raczej organizacji niż operacji. Zajmują się one zapamiętywaniem stałych, rezerwowaniem miejsca pamięci dla wprowadzanych danych i t.p. Powrócimy do nich później.

Właściwe rozkazy programu tworzą drugą grupę. One to mówią maszynie, aby wykonała znane operacje dodawania, odejmowania i t.p. oraz inne operacje, być może mniej znane. Jest to pierwszy etap naszej nauki.

Najpierw małe wprowadzenie.

AKUMULATORY

Najprostszą funkcją wymaganą przez nas od komórki pamięci jest przechowywanie liczb. Niezależnie od przechowywania liczb możemy chcieć dodawać je, dzielić i używać ich w innych operacjach. Jeżeli każda komórka byłaby wyposażona w elektronikę, koszt maszyny wzrósłby ogromnie.

W praktyce bilans jest zrównoważony. Tylko niektóre komórki są w pełni wyposażone do wykonania każdej operacji. Komórki te zwane są akumulatorami.

Co stanie się, gdy liczba ma być, powiedzmy, dzielona, a nie ma jej w komórce wykonującej dzielenie? Jak zobaczymy, można dokonać zupełnie łatwo kopiowania tej liczby do komórki wykonującej dzielenie t.zn. wprowadzić ją do akumulatora.

Po tych uwagach ogólnych, zapoznajmy się z akumulatorami w serii 1900.

Seria maszyn 1900 jest zorganizowana w ten sposób, że niezależnie od tego gdzie w pamięci operacyjnej umieszczony jest program,

pierwsze 8 komórek każdego programu działają jako akumulatory.

W przeciwieństwie do innych komórek 8 akumulatorom programu nie nadaje się nigdy nazw. Zwracanie się do nich odbywa się poprzez numer w zakresie od 0 do 7.

Jak wspomniano wcześniej, akumulatory rozróżniamy przez poprzedzanie ich numerów literą X. Tak więc, jeżeli mówimy o X0, to zwracamy się do akumulatora 0. Jeżeli mówimy o X1, zwracamy się do akumulatora 1 i t.d.

Fakty te zasługują na zapamiętanie ich, ponieważ znaczna większość operacji w ten czy inny sposób zwraca się do akumulatorów.

DODAWANIE

Zacznijmy od dodawania, ponieważ jest to operacja arytmetyczna, z którą jesteś najbardziej oswojony. Załóżmy, że chcemy dodać 5 do 3. Dodawanie może zachodzić tylko wtedy, gdy jedna z liczb jest w akumulatorze. Powiedzmy, że w akumulatorze 2 mamy 5, a wartość 3 jest zapamiętana w komórce o nazwie DIGIT. W jaki sposób komórka ta otrzymała tę nazwę, dowiesz się później.

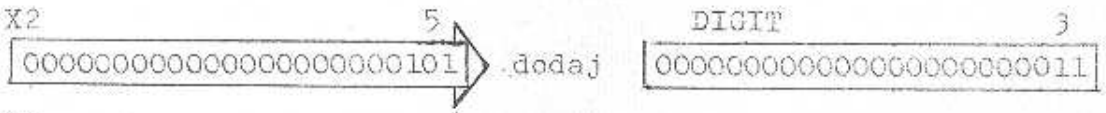
Możemy albo dodać zawartość DIGIT do akumulatora, albo też zawartość akumulatora do zawartości DIGIT.

Chociaż wynikiem każdego z tych dodawań jest 8, nie jest on jednak tworzony w tym samym miejscu. Poniższy schemat pokazuje dokładnie jak to się dzieje.

przed

po

przed



po

W każdym przypadku wynik jest tworzony w komórce, do której następuje dodawanie. Zauważ także, że zawartość pozostałej komórki jest niezmieniona.

Jeżeli uważasz, że zrozumiałeś kilka ostatnich stron, spróbuj rozwiązać następujące zadanie:

Akumulator 4 zawiera liczbę 50, a komórka CENT liczbę 100. Jakie liczby będą w nich zawarte po dodaniu zawartości CENT do zawartości akumulatora?

Napisz twoją odpowiedź i porównaj z właściwą.

Odpowiedź

Akumulator 4 zawiera 150.
Komórka CENT zawiera 100.

Rozkaz ADX /Dodaj do akumulatora/

Jeżeli przypomnimy sobie teraz pierwszy przykład dodawania, możemy rozpocząć jego kodowanie.

X2 zawiera 5, a komórka pamięci zwana DIGIT zawiera 3. Chcemy dodać zawartość DIGIT do akumulatora.

X2 5

000000000000000000000101

DIGIT 3

000000000000000000000011

Operacją, którą chcemy wykonać, jest dodanie do akumulatora. Symbolem mnemonicznym dla niej w PLANie jest ADX, t.zn. dodaj do X, gdzie X jest akumulatorem. Symbol ten piszemy na arkuszu programowym a kolumnie przewidzianej dla operacji.

ETYK. 1 6	OPER.	AK. 7						
	ADX							

Zauważ, że A jest pisane w kol.7, D w kol.8 i X w kol.9. Rozkaz, napisany inaczej niż pokazano powyżej, będzie nieprawidłowy.

Rozkaz ADS /dodaj do komórki pamięci/

Przejdźmy do drugiego przykładu dodawania. Ustawienie pierwotne jest takie samo jak poprzednio.

X2 5

000000000000000000000101

DIGIT 3

000000000000000000000011

Tym razem, zamiast dodać zawartość komórki do zawartości akumulatora, chcemy dodać zawartość akumulatora do zawartości komórki pamięci. Te dwa rodzaje dodawania są dla maszyny zupełnie różne, stąd konsekwentnie i symbol mnemoniczny PLANu jest inny.

Jest to ADS - dodaj do komórki pamięci.

Napiszmy go więc na arkuszu programowym, pamiętając, że A jest pisane w kol.7, D w kol.8 i t.d.

Następnie należy ustalić akumulator, którego zawartość jest dodawana do komórki pamięci.

W tym przypadku jest to akumulator 2. Napiszmy więc 2 na arkuszu programowym w kol.13.

Aby zakończyć ten rozkaz, musimy tylko ustalić komórkę, do której chcemy wykonać dodawanie. Jest to komórka zwana DIGIT, którą napiszemy na arkuszu programowym, zaczynając od kol.16.

Kompletny rozkaz napisany na arkuszu programowym ma postać:

ETYK.	OPER.	AK.	
	ADS	2	DIGIT

Wynikiem wykonania tych dwóch rozkazów pokazanych na arkuszu programowym było dodanie do siebie 5 i 3, pomimo faktu, że liczby 5 i 3 nie były nigdzie wspomniane w samych rozkazach.

Rozkazy maszyny cyfrowej działają nie na liczbach, a na adresach liczb przechowywanych w pamięci maszyny.

Oto pytanie przeznaczone do sprawdzenia twoich wiadomości na ten temat:

Jak pokazano niżej, akumulator 2 zawiera 4, a komórka pamięci zwana THREE zawiera 7.

X2 4

00000000000000000000100

THREE 7

00000000000000000000111

Maszyna wykonuje rozkaz:

	ADS	2	THREE		

Jakie liczby będą zawarte w akumulatorze i komórce pamięci po wykonaniu dodawania?

Odpowiedź

Akumulator 2 zawiera 4.
Komórka THREE zawiera 111.

Zanim nabierzesz wprawy w posługiwaniu się rozkazami ADX i ADS - kilka słów na temat karty wzajemnych zależności. /Reference Card/.

Karta wzajemnych zależności

Dotychczas zajmowaliśmy się dwiema pozycjami kodu rozkazów PLAN - rozkazami ADX i ADS. Będziemy zajmować się jeszcze innymi. Nie oczekujemy od ciebie, że nauczysz się ich na pamięć od razu. Do tego czasu rób użytek z dodatku C, znajdującego się w końcu tej książki.

Dla każdej operacji maszyny dodatek podaje kod PLANu i wynik operacji.

Dla ADX wygląda to np. następująco:

ADX x' = x+n+c V

Używana notacja wyjaśniona jest w dodatku C w części zatytułowanej NOTATION .

x' jest zawartością akumulatora /x/ po wykonaniu rozkazu. Znak równości ma zwykłe znaczenie.

x jest zawartością akumulatora przed wykonaniem rozkazu.

n jest zawartością komórki pamięci /N/ przed wykonaniem rozkazu

V symbol ten może być chwilowo ignorowany.

Niewiele trzeba czasu dla opanowania tej notacji, a skoro to będzie zrobione, cała rzecz wyjaśnia się sama.

Przykłady praktyczne

1. Dodaj zawartość CAKES do akumulatora 6.
2. Dodaj zawartość akumulatora 0 do komórki BUNS.
3. Dodaj zawartości PIGS, SHEEP i HENS do akumulatora 4.
4. Dodaj zawartości akumulatorów 5 i 6 do komórki SUM.

Odpowiedź:

1. ADX 6 CAKES
2. ADS 0 BUNS

3.

	ADX	4	PIGS		
	ADX	4	SHEEP		
	ADX	4	HENS		

4.

	ADS	5	SUM		
	ADS	6	SUM		

Ładowanie i pamiętanie

LDX /załaduj do akumulatora/

Jak powiedziano już wcześniej, przy dodawaniu jedna z liczb musi znajdować się w akumulatorze.

Jak więc mamy wykonać dodawanie zawartości dwóch komórek pamięci COKE i COAL?

Dodawanie może być przeprowadzone tylko wtedy, gdy jedna z wartości zostanie wpierw za pomocą rozkazu LDX wprowadzona do akumulatora.

Jeżeli zdecydujemy się umieścić w akumulatorze COKE, wówczas LDX kopiuje zawartość COKE do określonego akumulatora. Zawartość COKE nie ulega żadnym zmianom, ale poprzednia zawartość akumulatora zostaje zniszczona.

Załóżmy, że COKE zawiera liczbę 16.

przed

X3

10101111110101101111000l

COKE 16

00000000000000000010000

	LDX	3	COKE		

po

X3 16

00000000000000000010000

COKE 16

00000000000000000010000

Zauważ, że na arkuszu programowym "O" w słowie "COKE" jest przekreślone przez środek kreską poziomą. Należy to robić zawsze, aby uniknąć pomieszania litery "O" z cyfrą "O", które mają w maszynie inną postać.

Dodawanie COAL do COKE jest teraz łatwo wykonane za pomocą rozkazu ADX.

	LDX	3	COKE		
	ADX	3	COAL		

STO /zapamiętaj zawartość akumulatora/

Gdyby komórka COAL z poprzedniego przykładu zawierała 4, akumulator 3 zawierałby wynik 20. Jest mało prawdopodobne, aby wynik ten po wykonaniu obliczenia pozostał w akumulatorze. Będzie on prawdopodobnie zapamiętany gdzie indziej w pamięci maszyny z obawy przed zniszczeniem przez następne obliczenia. Zapamiętajmy np. wynik w FUEL. W tym celu używamy rozkazu STO.

przed

X3 20

00000000000000000010100

FUEL

00001111111000000101100

	STO	3	FUEL		

po

X3 20

00000000000000000010100

FUEL

00000000000000000010100

W trakcie wykonywania tej operacji poprzednia zawartość FUEL zostaje zniszczona.

Przykłady:

1. Załaduj zawartość TONS do X0.
2. Zapamiętaj zawartość X5 w MONEY.
3. Dodaj zawartość BOY do BUGLE i zapamiętaj wynik w NOISE. /5/*

Odpowiedzi:

1.

	LDX	0	TONS		

2.

	STO	5	MONEY		

3.

	LDX	5	BOY		
	ADX	5	BUGLE		
	STO	5	NOISE		

* W tym przykładzie i w następnych liczba w nawiasach odnosi się do numeru akumulatora użytego w odpowiedzi. Odpowiedź nie będzie oczywiście zła, jeżeli użyjesz innego akumulatora w zakresie 0 - 7.

Odejmowanie

Odejmowanie odpowiada dokładnie dodawaniu. Jedna z liczb musi być w akumulatorze. Istnieje możliwość odejmowania od zawartości akumulatora i od zawartości komórki pamięci. Oto symbole mnemoniczne PLAN dla tych dwóch operacji odejmowania: SBX i SBS.

SBX /Odejmij od akumulatora/

Rozkaz ten pozwala na odjęcie zawartości komórki pamięci od zawartości akumulatora. Odjęcie SPARE od X7 wykonane jest przez kodowanie:

	SBX	7	SPARE		

SBS /Odejmij od pamięci/

Rozkaz ten pozwala na odjęcie zawartości akumulatora od zawartości komórki pamięci.

Kodowanie odejmowania zawartości akumulatora 0 od STOCK jest następujące:

	SBS	0	STOCK		

Chociaż nie jest możliwe odejmowanie FISH od SEA w jednym kroku, można je jednak wykonać w zupełnie podobny sposób jak w przypadku dodawania COKE do COAL.

Oto kodowanie tego przykładu, w którym zabezpieczono wynik końcowy obliczenia poprzez zapamiętanie go w komórce HOLD:

	LDX	1	SEA		
	SBX	1	FISH		
	STO	1	HOLD		

Przykłady:

1. Odejmij zawartość PRICE od akumulatora 6.
2. Odejmij zawartość akumulatora 4 od COST.
3. Odejmij COAL od PIT. Zapamiętaj wynik w BATH.
4. Dodaj zawartości DRUM, HARP i FLUTE i zapamiętaj wynik w TRIO. Zapamiętaj DRUM + HARP w DUET. Odejmij DUET od TRIO i zapamiętaj wynik w SOLO.

Odpowiedzi:

1.

	SBX	6	PRICE		

2.

	SBS	4	COST		

3.

	LDX	0	PIT		
	SBX	0	COAL		
	STO	0	BATH		

4.

	LDX	1	DRUM		
	ADX	1	HARP		
	STO	1	DUET		
	ADX	1	FLUTE		
	STO	1	TRIO		
	SBX	1	DUET		
	STO	1	SOLO		

Negacja

Zaczynając omawianie odejmowania, powiedzieliśmy, że odejmowanie "odpowiada dokładnie dodawaniu". Istotnie, w tej maszynie i w wielu innych, odejmowanie jest dodawaniem, ale z jedną modyfikacją.

Weźmy przypadek 5 minus 3. Odejmowanie to przeprowadza się w dwóch etapach. Najpierw zestaw bitów przedstawiających 3 jest odejmowany od zera.

```
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    0
- 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1    3
-----------------------------------------------------------------
  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1   -3
```

Wynikowy zestaw bitów reprezentuje -3. O liczbie 3 mówimy, że jest zanegowana. Zanegowane 3 jest następnie dodane do 5, dając wynik właściwy.

```
  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1   -3
+ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1    5
-----------------------------------------------------------------
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0   +2
```

Negacja jest wykonywana automatycznie w trakcie rozkazu SBX lub SBS, ale jeżeli to jest konieczne, pojedyńcza operacja negacji

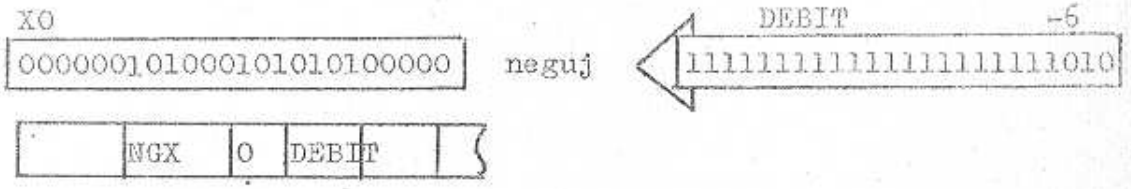
może być przeprowadzona na liczbach poza rozkazem odejmowania.

NGX /Neguj w akumulatorze/

Rozkaz ten powoduje kopiowanie liczby z komórki pamięci i negowanie jej w akumulatorze, t.j. umieszczenie tej liczby w akumulatorze ze zmienionym znakiem.

Funkcja ta używana jest najczęściej do zmiany liczb ujemnych na formę dodatnią przed wyprowadzeniem ich z maszyny. Po co jest to robione, zostanie omówione później. Funkcja ta zmienia również liczby dodatnie na ujemne.

przed



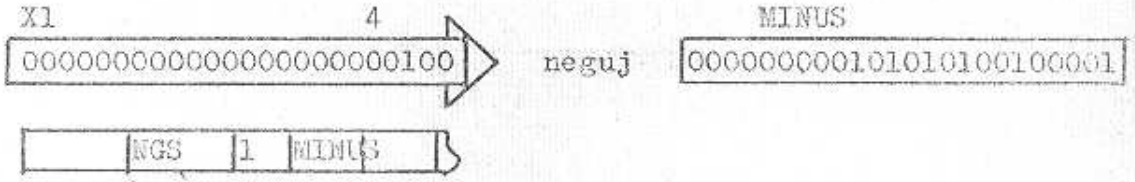
po

NGS /Neguj w pamięci/

Tutaj liczba pobierana jest z akumulatora i negowana w komórce pamięci.

W tym przykładzie wybraliśmy do zanegowania liczbę dodatnią.

przed



po

Przykłady:

1. Jeżeli BANK zawiera -17, umieść +17 w akumulatorze 3.
2. Liczba 75 jest w akumulatorze 4. Umieść -75 w komórce DED.
3. Odejmij COSTS od SALES i zapamiętaj odpowiedź w GAINS. Wykonaj ten przykład, używając rozkazu NGX. /5/

Odpowiedzi:

1.

	NGX	3	BANK	

2.

	NGX	4	DEP	

3.

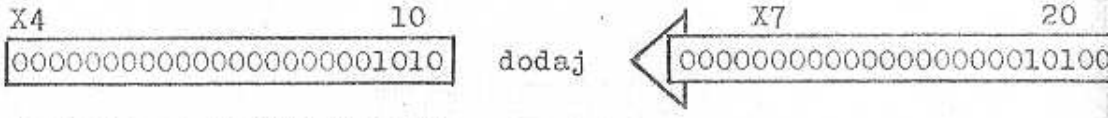
	NGX	5	COSTS	
	ADX	5	SALES	
	STO	5	GAINS	

Akumulatory jako komórki pamięci

Omówione dotychczas rozkazy wymagały, aby jedna z wchodzących w grę liczb znajdowała się w akumulatorze. Ponieważ akumulator jest także specyficzną komórką pamięci, nie ma żadnego powodu, aby obie liczby nie mogły być w akumulatorach.

Np. spójrz na poniższe:

przed

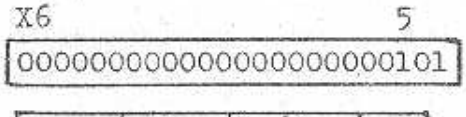


	ADX	4	7	

po

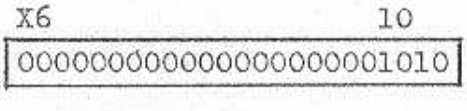
Możliwy jest nawet przypadek następujący:

przed



	ADX	6	6	

po



Takie użycie ADX podwaja zawartość akumulatora.

W ten sam sposób zawartość akumulatora może być zerowana przez SBX i negowana przez NGX.

Przykłady:

1. Dodaj zawartość X0 do X1.

2. Odejmij zawartość X2 od X7.
3. Skopiuj zawartość X6 do X4.
4. Zamień -70 w X3 przez +70.
5. Podwój zawartość X5.

Odpowiedzi:

1. | | ADX | 1 | 0 | |

albo

| | ADS | 0 | 1 | |

2. | | SBX | 7 | 2 | |

albo

| | SBS | 2 | 7 | |

3. | | LDX | 4 | 6 | |

albo

| | STO | 6 | 4 | |

4. | | NGX | 3 | 3 | |

albo

| | NGS | 3 | 3 | |

5. | | ADX | 5 | 5 | |

albo

| | ADS | 5 | 5 | |

Funkcje z operandami w postaci literali

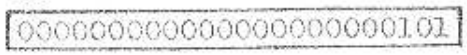
Przed rozpoczęciem pisania programu programista orientuje się, że gdzieś w środku programu potrzebuje umieścić 257 w akumulatorze. Potrzeba ta może być zaspokojona dwoma sposobami. Po pierwsze można założyć z góry na początku programu, że 257 będzie zapamiętane jako stała. Rozkaz LDX wprowadzi ją do akumulatora, gdy będzie potrzebna. Działanie tej metody zobaczymy później.

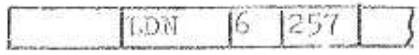
Alternatywnie, 257 może być wprowadzone do akumulatora rozkazem LDN.

Ponieważ funkcja ta zwraca się bezpośrednio do 257, a nie do komórki pamięci, operand jej ma postać literala.

LDN /Ładuj literal do akumulatora/

przed X6





po X6 257

Jest jedno ograniczenie co do użycia tej i wszystkich pozostałych funkcji tego samego typu, które należy pamiętać. Liczba w polu operandu nie może przekraczać 4095. Ograniczenie to pochodzi stąd że słowo rozkazu ma na pomieszczenie operandu tylko 12 pozycji bitowych. Największą liczbą binarną, która w formie dziesiętnej wynosi 4095, może być:

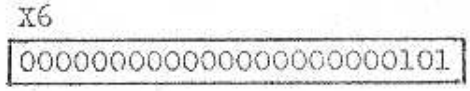
1 1 1 1 1 1 1 1 1 1 1 1

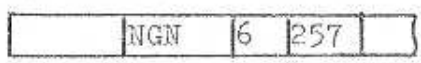
NGN /Neguj literał do akumulatora/

Dalsze ograniczenie odnośnie funkcji z operandami w postaci literała wymaga, aby literał był liczbą dodatnią. Możemy jednak załadować liczby ujemne, jeżeli użyjemy NGN.

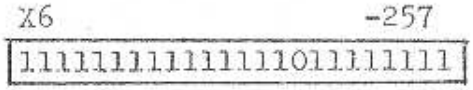
Załadujemy -257 zamiast +257.

przed X6





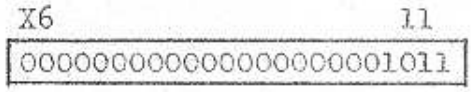
po X6 -257

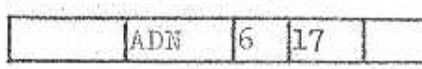


Dalej pokazane są funkcje dodawania i odejmowania, używające literali: ADN i SBN.

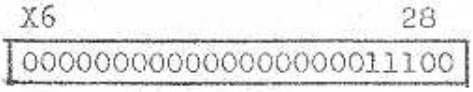
ADN /Dodaj literał do akumulatora/

przed X6 11



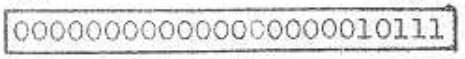


po X6 28



SBN /Odejmij literał od akumulatora/

przed X1 23



	SBN	1	15	

po X1 8

Przykłady:

1. Załaduj 2519 do X5.
2. Załaduj -877 do X6.
3. Dodaj 648 do X7.
4. Odejmij 3000 od X0.
5. Odejmij 50 od komórki HASH.
6. Dodaj 4096 do X2.

Odpowiedzi:

1.

	LDN	5	2519	

2.

	NGN	6	877	

3.

	ADN	7	648	

4.

	SBN	0	3000	

5.

	LDN	1	50	
	SBS	1	HASH	

6.

	ADN	2	4095	
	ADN	2	1	

Prawdziwa historia

Każdego miesiąca w firmie Bauble i Ska ilości paciorków szklanych, wykonanych przez poszczególne maszyny, przekazywane były do ośrodka maszyn cyfrowych w celu podsumowania. Wynik w formie raportu przekazywany był naczelnemu dyrektorowi. W lutym, marcu i kwietniu ilość ogólna produkcji dochodziła do i przekraczała 8.000.000 szt. Przyszedł maj. Produkcja rosła, a kierownik ośrodka maszyn cyfrowych osobiście sprawdzał prawidłowość wprowadzania danych do maszyny 1902. Jednak, mimo wielu prób, maszyna wytwarzała wyniki

w niczym nie zbliżone do prawdziwych. Sprawdzano do późnej nocy, a wyniki wciąż były nonsensowne. Maszyna wypisywała nonsensy. Raport dla dyrektora był zagrożony.

Co było nie w porządku?

Nadmiar

Rzecz bardzo prosta.

Dalsze badanie ujawniło, że suma za maj była zbyt duża, aby zmieścić się w jednym słowie maszyny, ale programiści nie zwrócili na to uwagi. Stąd konsternacja.

Gdy występuje nadmiar, jak to miało miejsce w tym przypadku, zostaje to wykazane w specjalnym 1-bitowym rejestrze, mianowicie rejestrze V, wewnątrz maszyny. Nadmiar umieszcza automatycznie "1" w rejestrze V. Mówimy wówczas, że rejestr V jest "ustawiony". Normalnie zawiera on "0" i mówimy wtedy, że jest "nieustawiony".

Wszystkie funkcje, które mogą w ten sposób spowodować ustawienie rejestru V oznaczone są przez V w załączniku C.

Jednak jedynym sposobem określenia, czy rejestr V jest ustawiony, czy nie, jest włączenie do programu rozkazu badania tego rejestru. W niedalekiej przyszłości spotkasz sytuacje, gdzie korzystnym będzie świadome spowodowanie nadmiaru, ale musisz być na tyle przezorny, aby nie dopuścić do wystąpienia takiej sytuacji przypadkowo

Programista musi mieć dane całkowicie określone zanim przystąpi do opracowania programu przetwarzania ich. Oczywiście, nie chodzi tu o sporządzenie uprzednich obliczeń, gdyż w tym przypadku posiadanie maszyny cyfrowej nie miałoby sensu, ale o określenie wielkości maksymalnych dla obliczeń.

Co robić, jeżeli pewne wartości są za wielkie, aby pomieścić je w jednym słowie?

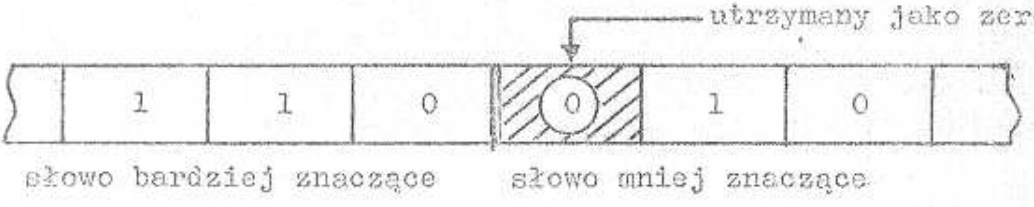
Wlej dwie szklanki wody do jednej szklanki a otrzymasz nadmiar. Wlej dwie szklanki wody do dwóch szklanek i nie będziesz miał nadmiaru. Jeżeli liczba nie mieści się w jednym słowie, zapamiętaj ją w dwóch słowach. Następnym zagadnieniem, które będziemy omawia jest Pamięć Podwójnej Długości.

Pamięć podwójnej długości

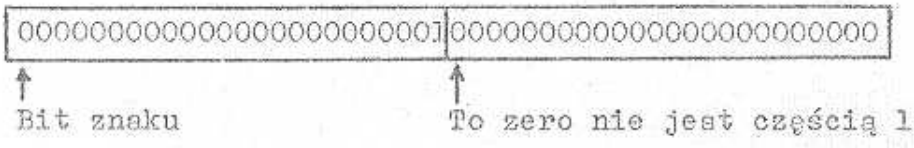
Na jedną rzecz trzeba zwrócić uwagę przy przechowywaniu liczb

w dwóch słowach zamiast w jednym.

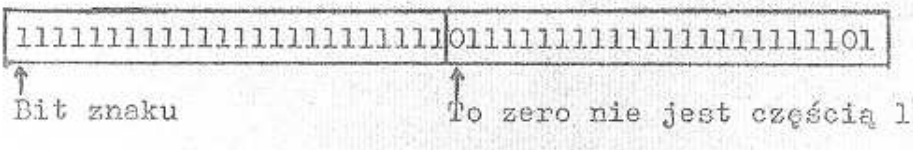
Jak wiesz, najbardziej lewy bit każdego słowa maszynowego jest bitem znaku. Przechowując jedną liczbę w dwóch słowach, mamy do naszej dyspozycji dwa bity znaku, gdy potrzebny jest tylko jeden. Przed tą redundacją maszyna zabezpiecza się w ten sposób, że utrzymuje bit znaku w słowie najmniej znaczącym stale jako zero. Pozycja ta jest więc dla celów pamiętania całkowicie ignorowana.



Pamiętaną tu wartością w słowie podwójnej długości jest 8.388.608. Liczba ta jest o jeden większa, niż największa liczba dodatnia, która może być zapamiętana w słowie pojedyńczym.



Ignorowany bit jest pokazany bardziej wyraźnie w schemacie liczby -3.

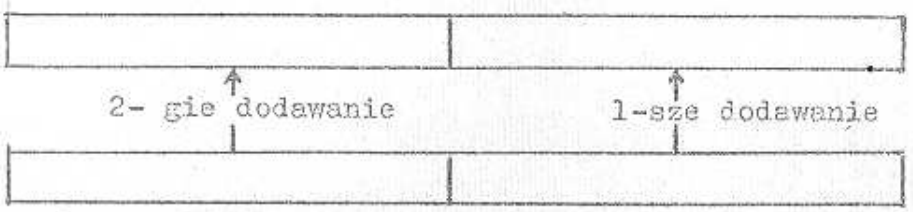


Pojemność słowa podwójnej długości przekracza o współczynnik prawie jednego miliona pojemność słowa pojedyńczej długości. W słowie podwójnej długości może być zapamiętana każda liczba w zakresie od

-70.368.744.177.664

do 70.368.744.177.663

Tyle o przechowywaniu liczb o podwójnej długości. Dodawanie do siebie dwóch liczb o podwójnej długości wykonywane jest w dwóch etapach, t.j. przez dwa rozkazy dodawania.



Traktowanie dodawania tego typu jako dwa oddzielne dodawania ma swoje ujemne strony, chociaż czasami prowadzi do celu.

1.
```
  2246
+ 1934
------
```

2.
```
  22     46
+ 19   + 34
------ ----
  41     80
```

3.
```
  2246
+1934
------
  4180
```

ale innym razem

1.
```
  7894
+ 2120
------
```

2.
```
  78     94
+ 21   + 20
------ ----
  99    114
```

3.
```
  7894
+ 2120
------
 10014
```

problem przeniesienia wymaga pewnego powiązania między jednym dodawaniem i następnym.

Wynik rozkazu ADX jest pgraniczony do akumulatora adresowanego przez ten rozkaz. Oznacza to, że dodawanie podwójnej długości będzie przeprowadzone przez dwa rozkazy ADX tylko wtedy poprawnie, gdy między jednym i drugim dodawaniem nie będzie żadnego przeniesienia.

A co robić, jeżeli jest przeniesienie, którego możemy się spodziewać 50 razy na 100?

ADX nie rozwiązuje tego problemu, ale jest podobny rozkaz, który to robi.

Arytmetyka podwójnych długości

ADXC

Rozkaz ADXC używany jest tylko dla dodawania podwójnej długości. Pod każdym względem , z wyjątkiem jednego, jest on taki sam, jak ADX. Inny jest tylko sposób traktowania przeniesienia.

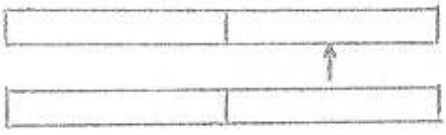
Jeżeli między dwoma dodawaniami potrzebne jest przeniesienie, ADXC umieszcza to przeniesienie w specjalnym rejestrze - rejestrze C. Ponieważ liczby przedstawiane są w postaci binarnej, wielkość przeniesienia może wynosić tylko jedną cyfrę binarną, rejestr C potrzebuje więc pojemność 1 bitu. Podobnie jek rejestr V jest "ustawiony", gdy zawiera "1", w przeciwnym przypadku jest "nieustawiony".

Drugi rozkaz dodawania zeruje automatycznie rejestr C i włącza przeniesienie do następnego dodawania.

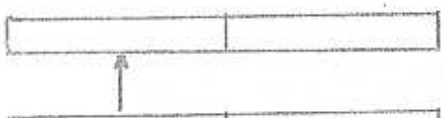
Dla drugiego dodawania nie musimy używać rozkazu ADXC. Chociaż ADX nie ustawia rejestru przeniesienia, zeruje go automatycznie. Pokazane jest to przez małe c w karcie zależności wzajemnych.

ADX x' = x + n + c

Możemy teraz zilustrować schematycznie działanie dodawania podwójnej długości z uwzględnieniem przeniesienia.



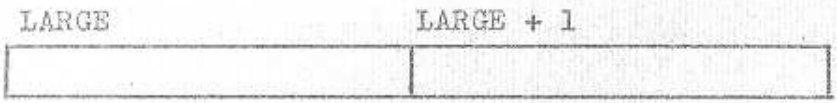
C ustawiony po pierwszym dodawaniu /ADXC/



zawartość C włączona w drugim dodawaniu /ADX/

C nieustawiony po drugim dodawaniu

Nazwa przydzielona każdej komórce pamięci musi być unikalna. Co stanie się w przypadku, gdy pojedyncza wielkość zwana, powiedzmy, LARGE przechowywana jest w dwóch komórkach? Nazwa LARGE dana jest dla słowa bardziej znaczącego, pozostałemu członowi dana jest nazwa LARGE+1.



Przejdźmy do arkusza programowego.

Dodanie wielkości podwójnej długości przechowywanej w komórkach NUM, NUM+1 do akumulatorów 5 i 6 wykonuje się następująco:

	ADXC	6	NUM+1	
	ADX	5	NUM	

W każdym dodawaniu podwójnej długości wisi nad Twoją głową niczym miecz Damoklesa niebezpieczeństwo zaniedbania przeniesienia. Dlatego też koniecznym jest, aby najmniej znaczące słowo było dodawane jako pierwsze i aby to dodawanie było wykonywane za pomocą funkcji, która może ustawiać rejestr C.

Inne funkcje ustawiające rejestr C

ADXC nie jest jedynym rozkazem mającym "dublowanie", które może ustawić rejestr C. Możesz zobaczyć w dodatku C, że wszystkie omówione dotychczas rozkazy, zostały w ten sposób zdublowane. Ich kody PLANu są łatwe do zapamiętania, ponieważ są to normalne kody rozkazów, po których następuje litera C. Tak więc, ADXC, jak to widzieliśmy, jest podwójnym rozkazem od ADX, SBXC jest podwójnym rozkazem od SBX i t.d.

Przerobimy teraz krótki przykład zanim będziesz mógł samodzielnie manipulować wielkościami podwójnej długości.

Przykład

Odejmij wielkość podwójnej długości przechowywaną w MAXB, MAXB+1 od wielkości podwójnej długości przechowywanej w MAXA, MAXA+1. Zapamiętaj odpowiedź w komórkach MAX, MAX+1.

	LDXC	6	MAXA	+1	
	LDX	5	MAXA		
	SBXC	6	MAXB	+1	
	SBX	5	MAXB		
	STOC	6	MAX	+1	
	STO	5	MAX		

Ponieważ wielkość przechowywana w MAXA i MAXA+1 jest w postaci podwójnej długości, nie ma możliwości przeniesienia przy ładowaniu jej do akumulatorów. /Pamiętaj, że bit znaku słowa najmniej znaczącego jest ustawiony jako zero/. Dwa rozkazy LDX mogłyby wykonać to zadanie równie dobrze. Te same uwagi odnoszą się także do STOC.

Przykłady

1. Dodaj wielkość w komórkach TAX, TAX+1 do akumulatorów 4, 5.
2. Odejmij wielkość w komórkach QUANT, QUANT+1 od akumulatorów 3, 4.
3. Zaneguj zawartość komórek DEBIT, DEBIT+1 do akumulatorów 0, 1.
4. Dodaj zawartość akumulatorów 6, 7 do komórek PENCE, PENCE+1.
5. Odejmij zawartość akumulatorów 1, 2 od komórek DEBIT, DEBIT+1.
6. Zaneguj zawartość akumulatorów 5, 6 do komórek MOD, MOD+1.

7. Wielkość "a" jest przechowywana w komórkach NUMA, NUMA+1; "b" w NUMB, NUMB+1 i "c" w NUMC, NUMC+1. Zapamiętaj a + b - c w komórkach ANS, ANS+1.

Odpowiedzi:

1.

	ADXC	5	TAX+1	
	ADX	4	TAX	

2.

	SBXC	4	QUANT+1	
	SBX	3	QUANT	

3.

	NGXC	1	DEBIT+1	
	NGX	0	DEBIT	

4.

	ADSC	7	PENCE+1	
	ADS	6	PENCE	

5.

	SBSC	2	DEBIT+1	
	SBS	1	DEBIT	

6.

	NGSC	6	MOD+1	
	NGS	5	MOD	

7.

	LDXC	2	NUMA+1	
	LDX	1	NUMA	
	ADXC	2	NUMB+1	
	ADX	1	NUMB	
	SBXC	2	NUMC+1	
	SBX	1	NUMC	
	STOC	2	ANS+1	
	STO	1	ANS	

Zamiana liczb pojedyńczej długości na postać podwójnej długości

Jeżeli przy dodawaniu szeregu liczb pojedyńczej długości spodziewamy się otrzymać wynik nie mieszczący się w jednym słowie maszyny, wówczas pierwsza liczba szeregu przed rozpoczęciem dodawania powinna być zamieniona na postać podwójną.

Zamiana liczby dodatniej pojedyńczej długości na postać podwójną

Załóżmy, że chcemy załadować liczbę dodatnią przechowywaną w komórce FACT do akumulatorów 6 i 7, aby była w postaci podwójnej długości.

Ponieważ liczba przechowywana w FACT jest dodatnia, bit znaku jest równy zero i konsekwentnie nie ma możliwości przeniesienia przy umieszczaniu jej w akumulatorze. Używamy rozkazu LDX i, oczywiście ładujemy liczbę do słowa najmniej znaczącego.

	LDX	7	FACT		

Wszystko, co pozostaje teraz do zrobienia, to wyzerować akumulator 6.

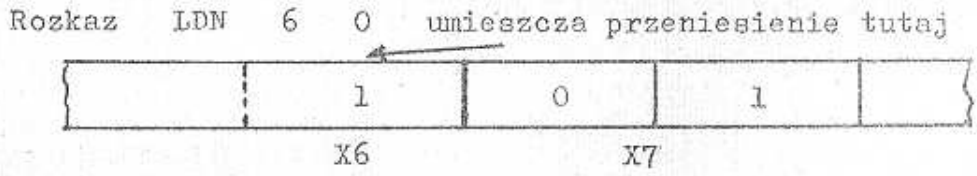
	LDX	7	FACT		
	LDN	6	0		

Zamiana liczby ujemnej pojedyńczej długości na postać podwójną

W tym przykładzie komórka FACT zawiera liczbę ujemną. Liczba ta musi mieć 1 w pozycji bitowej znaku. Aby więc załadować FACT do akumulatora, musimy użyć rozkazu LDXC, który wydzieli 1 z bitu znaku i umieści 1 w rejestrze C.

	LDXC	7	FACT		

Tym razem nie możemy użyć rozkazu LDN, bo chociaż LDN uwzględnia przeniesienie, nie ustawia on jednak 1 w pozycji bitowej znaku akumulatora 6.



Rozkazem, który musi być użyty, jest NGN.

	NGN	6	0		

Ponieważ rozkaz ten pobiera przeniesienie, efektem jego jest zanegowanie 1 do akumulatora 6. Negacja ta wypełnia akumulator 6 ciągiem "1" /jedynek/ i w ten sposób otrzymujemy prawidłowy bit znaku.

Potrzebnymi więc rozkazami są:

	LDXC	7	FACT		
	NGN	6	0		

Początkowy stan FACT

FACT

111111111111111111000001

Po wykonaniu pierwszego rozkazu

X6	X7
	011111111111111111000001

1 C "ustawiony"

Po wykonaniu drugiego rozkazu

X6	X7
111111111111111111111111	011111111111111111000001

0 C "rozładowany"

Przykłady:

1. Liczba dodatnia jest przechowywana w komórce QTY. Umieść ją w akumulatorach 2, 3 w postaci podwójnej długości.

2. To samo co w 1, gdy liczba przechowywana w QTY jest ujemna.

Odpowiedzi:

1.

	LDXC	3	QTY	
	LDN	2	0	

2.

	LDXC	3	QTY	
	NGN	2	0	

W ostatnich dwóch przykładach liczba pojedyńczej długości w pamięci była zamieniana na liczbę podwójnej długości w dwóch akumulatorach. Proces ten może być wykonany w kierunku odwrotnym.

Załóżmy, że akumulator 3 zawiera liczbę dodatnią, która ma być zapamiętana w komórkach RSLT, RSLT+1 w postaci podwójnej długości.

Jak w przykładzie pierwszym, nie ma tutaj żadnej możliwości przeniesienia, ponieważ liczba jest dodatnia. Możemy więc użyć normalnego rozkazu STO.

	STO	3	RSLT+1		
	LDN	2	0		
	STO	2	RSLT		

Zerowanie komórki RSLT wymaga dwóch rozkazów, ponieważ LDN tylko wprowadza liczby do akumulatorów.

Jeżeli liczba w akumulatorze 3 jest ujemna, musimy zastosować wersję STO, ustawiającą rejestr C, t.j. STOC.

	STOC	3	RSLT+1		
	LDN	2	0		
	NGS	2	RSLT		

Ostatnie dwa rozkazy zapewniają, że liczba otrzymuje właściwy znak. Ponieważ rozkaz LDN zeruje rejestr C, efektem jego jest umieszczenie 1 w najmniej znaczącej pozycji X2. Rozkaz NGS neguje 1 do komórki RSLT.

Przykłady

1. Liczba dodatnia znajduje się w akumulatorze 4. Zapamiętaj ją jako liczbę podwójnej długości w DBL, DBL+1.

2. To samo co w 1, gdy liczba w akumulatorze jest ujemna.

Odpowiedzi

1.

	STO	4	DBL+1		
	LDN	3	0		
	STO	3	DBL		

2.

	STOC	4	DBL+1		
	LDN	3	0		
	NGS	3	DBL		

Dodawanie i odejmowanie z liczbami pojedyńczej i podwójnej długośc

Jeżeli zmienimy najpierw liczbę pojedyńczej długości na formę podwójną, możemy wykonać dowolne dodawanie lub odejmowanie z liczbą o podwójnej lub pojedyńczej długości.

Jest jednak możliwe dodamie lub odjęcie liczby pojedyńczej długości do lub od liczby podwójnej długości bez wykonania tej zmiany.

Chcemy dodać np. wielkość pojedyńczej długości przechowywaną w NUM do wielkości podwójnej długości w akumulatorach 6, 7.

Może tu być przeniesienie, użyjemy więc rozkazu ADXC

	ADXC	7	NUM		

Jeżeli jest przeniesienie, drugi rozkaz zapewni, że zostanie ono dodane prawidłowo,

	ADN	6	0		

ponieważ ADN zeruje automatycznie rejestr C w taki sam sposób jak ADX. Nie stanie się nic złego, jeżeli nie ma przeniesienia, ponieważ dodanie 0 do wyniku nie wpływa na jego wartość.

Odjęcie NUM zamiast dodania nie jest wcale trudniejsze.

	SBXC	7	NUM		
	SBN	6	0		

Możemy także dodawać lub odejmować liczby nie przechowywane w pamięci, używając wersji ustawiającej rejestr C NGN, ADN i SBN.

Weźmy przypadek dodania 100 do liczby podwójnej długości będącej w akumualtorach 1, 2.

Piszemy.

	ADNC	2	100		
	ADN	1	0		

Drugi rozkaz ma tu, oczywiście, zająć się przeniesieniem, gdyby takie w wyniku dodawania powstało.

Odejmijmy teraz 2097 od tej samej liczby.

	SBNC	2	2097		
	SBN	1	0		

Przykłady:

1. Dodaj zawartość NUM do liczby pojedyńczej długości w X0, X1.

2. Odejmij zawartość X6 od liczby podwójnej długości w komórkach STORE, STORE+1.

3. Dodaj 3000 do liczby podwójnej długości w X5, X6.

4. Odejmij 1248 od liczby podwójnej długości przechowywanej w X3, X4.

Odpowiedzi:

1.

	ADXC	1	NUM		
	ADN	0	0		

2.

	SBSC	6	STORE+1		
	LDN	5	0		
	SBS	5	STORE		

3.

	ADNC	6	3000		
	ADN	5	0		

4.

	SBNC	4	1248		
	SBN	3	0		

Ładowanie liczb podwójnej długości nie będących w pamięci

Do wprowadzania liczb podwójnej długości mogą być używane funkcje z operandami będącymi literalami. Zarówno dla liczb pojedyńczej jak i podwójnej długości ma zastosowanie zwykłe ograniczenie, t.zn. że liczby ładowane w ten sposób mogą być tylko w zakresie -4095 do +4095.

Załadujmy, dla przykładu, akumulatory 1,2 liczbą 4095 w formie podwójnej długości. Ponieważ nie ma tu możliwości wystąpienia przeniesienia, możemy użyć LDN w jego normalnej formie.

	LDN	2	4095		
	LDN	1	0		

Rozkaz ostatni zeruje akumulator 1.

Ten sam proces powtarzany jest teraz z liczbą ujemną -540. Musimy użyć tutaj funkcji ustawiającej rejestr C.

	NGNC	2	540		
	NGN	1	0		

Rozkaz drugi zajmuje się ustawieniem znaku.

Nasze przykłady z liczbami podwójnej długości pozwoliły nam na użycie wszystkich funkcji z karty wzajemnych powiązań ustawiających rejestr C z wyjątkiem jednej - rozkazu LDNC.

Ponieważ LDNC musi mieć operand dodatni i to w zakresie 0 do 4095, jest zupełnie niemożliwe, aby jego użycie mogło spowodować przeniesienie. Ma on, w rzeczywistości, taki sam efekt jak LDN. Jeżeli całkowicie o nim zapomnisz, nie stanie się nic złego.

Przejdźmy teraz do przykładów.

Przykłady:

1. Zapamiętaj 1000 jako liczbę podwójnej długości w komórkach MILL, MILL+1.

2. Zapamiętaj -55 jako liczbę podwójnej długości w komórkach FROST, FROST+1.

3. Dodaj liczbę potrójnej długości przechowywaną w TRIN, TRIN+1, TRIN+2 do liczby potrójnej długości przechowywanej w akumulatorach 3, 4, 5. O liczbach potrójnej długości nie mówiliśmy przedtem. Obiecujemy nie mówić o nich później.

Odpowiedzi:

1.

	LDN	7	1000		
	LDN	6	0		
	STO	7	MILL+1		
	STO	6	MILL		

2.

	NGNC	7	55		
	NGN	6	0		

	STO	7	FROST+1	
	STO	6	FROST	

3.

	ADXC	5	TRIN+2	
	ADXC	4	TRIN+1	
	ADX	3	TRIN	

Drugim rozkazem w tym przykładzie musi być ADXC, ponieważ może on być potrzebny do wyzerowania, a następnie ustawienia rejestru przeniesienia.

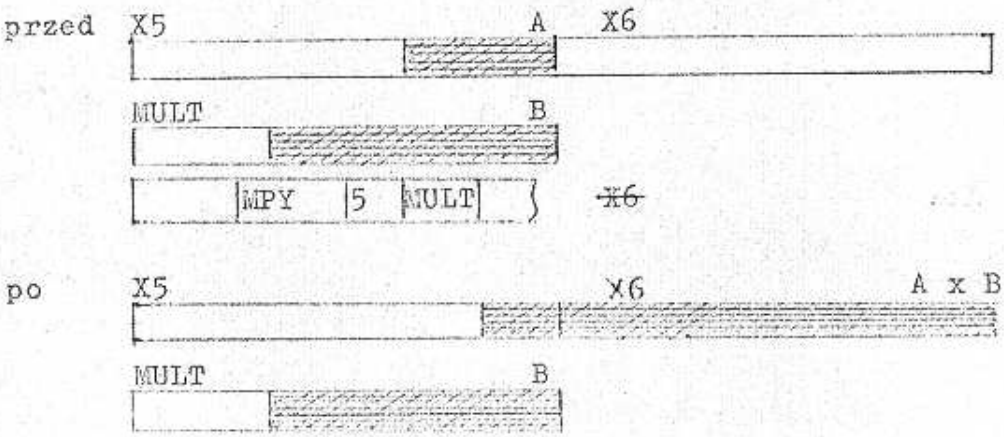
Mnożenie

Wyczerpawszy wszystkie rozkazy dodawania i odejmowania z wyjątkiem operacji zmiennoprzecinkowych, którymi w tej części nie będziemy się zajmować, możemy teraz zwrócić naszą uwagę na mnożenie i dzielenie.

MPY /Mnóż/

MPY mnoży dwie wielkości pojedyńczej długości /jedna w akumulatorze/ i daje wynik podwójnej długości w dwóch akumulatorach.

W przykładzie tym mnożymy zawartość akumulatora 5, którą nazwiemy A przez zawartość MULT, którą nazwiemy B.



O operacji tej należy zapamiętać: po pierwsze, że zawartość akumulatorów 5 i 6 jest niszczona, aby zrobić miejsce dla wyniku mnożenia i po drugie, że zawartość komórki pamięci pozostaje niezmieniona.

Ułamki binarne

Łatwo zauważyć, że MPY może być użyty do mnożenia dwóch liczb całkowitych. Ale co robić, gdy chcemy pomnożyć, powiedzmy, 5.3 przez 7.8?

Dla uniknięcia nieporozumień, powiedzmy od razu, że w maszynie nie ma ani kropki dziesiętnej, ani binarnej. Każda pozycja w słowie maszyny zawiera albo 1 albo 0 i między nimi nie ma nic.

Jak więc wykonać mnożenie 5.3 przez 7.8?

Jeżeli możesz wykonać to obliczenie na papierze, możesz wykonać je również w maszynie.

Zobaczmy, jak jest ono wykonywane na papierze. Ignorujemy kropki dziesiętne i mnożymy 53 przez 78.

```
  53
  78
 ---
 424
371
----
4134
```

Rozumujemy następnie, że liczba miejsc po kropce w wyniku jest równa sumie liczby miejsc po kropce w dwóch czynnikach. Pozwala nam to wstawić kropkę między 1 i 3 i otrzymać wynik właściwy.

41.34

Sytuacja podobna występuje w maszynie: jako wynik rozkazu MPY występuje 4134. Kropkę we właściwe miejsce wstawiamy wtedy, gdy przychodzi czas wydruku wyniku. Wydruk jednak nie jest w tej chwili przedmiotem naszego zainteresowania, nie musimy więc wchodzić już teraz głęboko w mechanikę tego problemu.

Istnieje również inny sposób traktowania ułamków.

Normalne znaczenia pozycji bitowych w słowie maszynowym są następujące:

	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	i t.d.
0	0	0	0	0	0	0

Gdy będziemy badać sposoby i środki przechowywania danych w maszynie przy realizacji programu, zobaczymy, że możliwa jest inna interpretacja.

	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	i t.d.
0	0	0	0	0	0	0

Tak więc, zamiast zapamiętać 0.5 jako 5 i ignorować kropkę, moglibyśmy zapisać to jako:

0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Zakładamy, że kropka istnieje między pierwszym 0 i 1. Kropka jest oczywiście kropką binarną, a nie dziesiętną.

0.125 lub $\frac{1}{8}$ w tej samej postaci przedstawia się jako:

0.	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

a 0.75 lub $\frac{3}{4}$ jako:

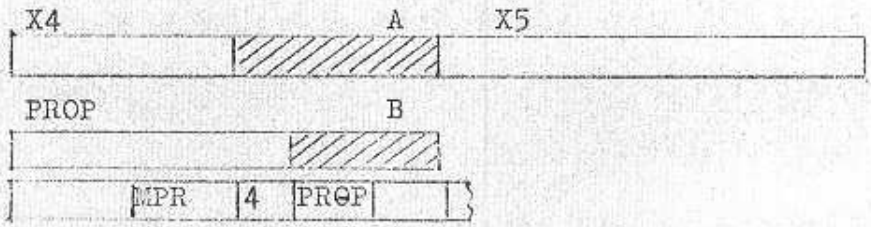
0.	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

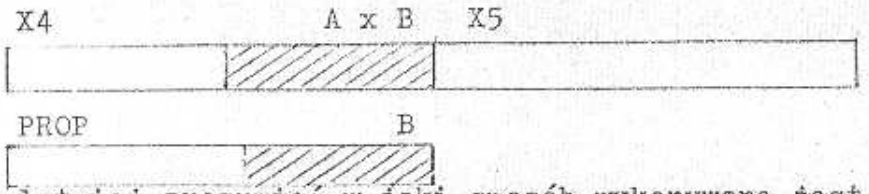
Chwila zastanowienia prowadzi do przekonania, że każda liczba, która może być przedstawiona jako ułamek dziesiętny, może być przedstawiona jako ułamek binarny w ten sam sposób.

Te krótkie uwagi mają pewne znaczenie dla następnego rozkazu mnożenia.

MPR /Mnóż i zaokrąglaj/

MPR mnoży dwie wielkości pojedyńczej długości, z których jedna musi być w akumulatorze. Wynik podwójnej długości tworzony jest tylko jako krok pośredni. Część wyniku w słowie bardziej znaczącym jest zaokrąglona w górę i zachowana, część w drugim słowie jest niszczona.



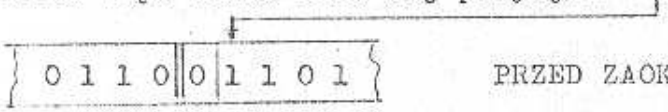


Aby łatwiej zrozumieć w jaki sposób wykonywane jest zaokrąglanie w górę, weźmy najpierw przykład dziesiętny. Załóżmy, że 3,56 ma być zaokrąglone do najbliższej pozycji dziesiętnej. Dodajemy pięć setnych /0,05/ i odrzucamy ostatnią cyfrę. Otrzymamy 3,61, następnie 3,6.

Jeżeli liczba ta ma być zaokrąglona jeszcze do najbliższej pozycji, dodajemy pięć dziesiętnych /0,5/.

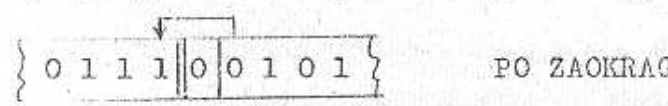
Najpierw otrzymamy 4,06, następnie 4.

Wróćmy do naszego przykładu mnożenia. Wymagana jest tylko część wyniku w słowie bardziej znaczącym. Aby zaokrąglić liczbę w górę, trzeba więc dodać 1 do tej pozycji.



PRZED ZAOKRAGLENIEM

W tym przypadku bit przeskakuje dalej, dając



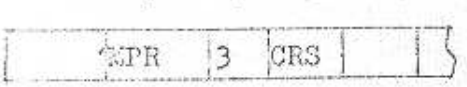
PO ZAOKRAGLENIU

Najmniej znacząca część wyniku jest następnie ignorowana.

Zrozumiałe już jest "jak" działa MPR, ale jeszcze nie "dlaczego". Zastosowanie tego rozkazu pokażemy w następnym przykładzie.

Maszyna przetwarza bieżąco program przydzielający specjalny fundusz na różne szczytne cele. Gdy podejmujemy opowieść, cały fundusz jest w akumulatorze 3. Jest on przedstawiony jako pewna liczba pensów, co jest normalną metodą przedstawiania wartości szterlingowych dla celów arytmetycznych. W komórce CRS przechowane jest $\frac{31}{365}$ w postaci ułamka. Jest to część należna Towarzystwu Reformy Kalendarza. Celem przed nami jest obliczenie $\frac{31}{365}$ funduszu w funtach, szylingach i pensach.

Wszystko co jest potrzebne, to jeden rozkaz:

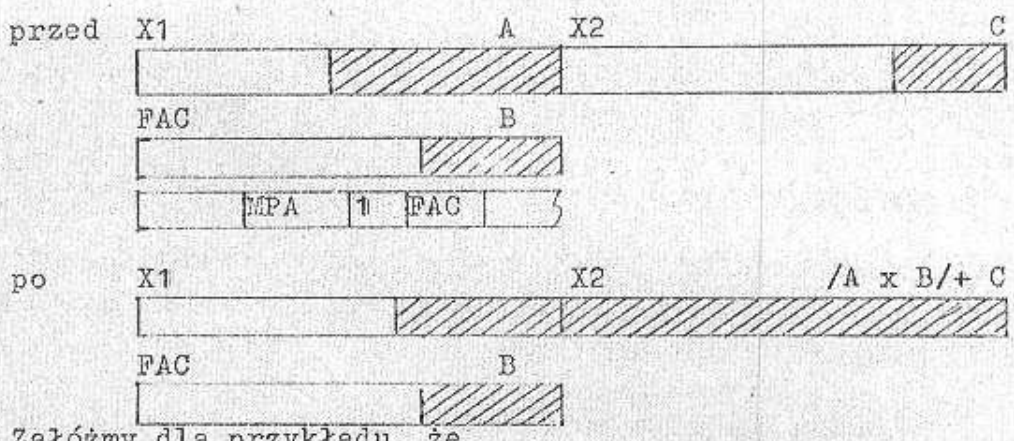


Akumulator 3 zawiera teraz wymaganą wartość zaokrągloną do najbliższej pełnej liczby pensów. Wartość ta jest następnie zamieniana dla wyprowadzenia z maszyny na funty, szylingi i pensy.

MPR działa zawsze w ten sam sposób, gdy liczba całkowita mnożona jest przez liczbę ułamkową - podawany jest wynik prawidłowy do najbliższej liczby całkowitej.

MPA /Mnóż i dodawaj/

Podobnie jak MPY, rozkaz ten daje iloraz podwójnej długości; zawartość mniej znaczącego akumulatora dodawana jest automatycznie do odpowiedzi.



Załóżmy dla przykładu, że

akumulator 1 zawiera	7
komórka FAC zawiera	8
akumulator 2 zawiera	4

7 pomnożone przez 8 daje 56, plus 4 daje 60 jako wynik końcowy, przechowywany w formie podwójnej długości w akumulatorach 1 i 2.

Przykładem użycia MPA może być program zamiany /konwersji/.

Długość w jardach, stopach i calach przechowywana jest w 3 komórkach: YARDS, FEET i NCHES. Należy zamienić całą ilość na cale i zapamiętać w NCHES. Końcowy wynik nie będzie większy od 8.000.000 cali. Komórki CF12 i CF36 przechowują współczynniki zamiany 12 i 36.

W miarę posuwania się, pisz odpowiedź na arkuszu programowym. Jako pierwszy krok trzeba załadować stopy i cale do sąsiednich akumulatorów.

```
        LDX   5   FEET
        LDX   6   NCHES
```

Następnie zamieniamy stopy na cale i, jeżeli użyjemy MPA, możemy mieć wynik dodany do cali już istniejących.

```
        MPA   5   CF12
```

Chociaż ilość cali może być technicznie podwójnej długości, jej wielkość i znak pozwalają traktować ją jako wielkość pojedyńczej długości. Jeżeli przesuniemy teraz jardy do akumulatora 5, możemy tę samą sztuczkę powtórzyć znów.

```
        LDX   5   YARDS
        MPA   5   CF36
```

Akumulator 6 zawiera żądaną liczbę cali, którą szybko zapamiętujemy.

```
        STO   6   NCHES
```

Przykłady:

1. Komórki SHEEP i FOUR zawiersją dwie liczby całkowite. Pomnóż je przez siebie i zapamiętaj wynik w komórkach LEGS, LEGS+1. /0,1/

2. Liczba ułamkowa przechowywana jest w komórkach BINF. Pomnóż ją przez liczbę całkowitą przechowywaną w komórce INT i umieść wynik /prawidłowy do najbliższej liczby całkowitej/ w komórce ROUND. /2/

3. Komórki POUND, SHILL i PENCE zawierają liczbę funtów, szylingów i pensów. Komórki CF12 i CF240 zawierają współczynniki zamiany 12 i 240. Zamień tę ilość na pensy i zapamiętaj wynik w komórce STERL. /Otrzymana ilość nie przekroczy pojemności jednego słowa/. /5,6/

Odpowiedzi:

1.
```
        LDX   0   SHEEP
        MPY   0   FOUR
        STO   0   LEGS
        STO   1   LEGS+1
```

2.

	LDX	2	INT
	MPR	2	BLNF
	STO	2	ROUND

3.

	LDX	6	PENCE
	LDX	5	SHILL
	MPA	5	CF12
	LDX	5	POUND
	MPA	5	CF240
	STO	6	STERL

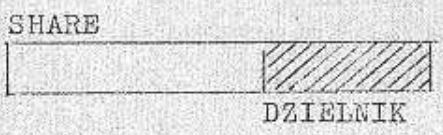
Dzielenie

Omówienie trzech przykładów dzielenia wyczerpuje resztę zwykłych operacji arytmetycznych.

DVD

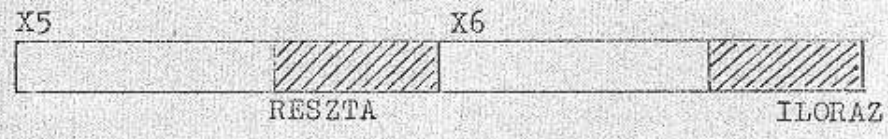
Rozkaz ten powoduje, że dzielna podwójnej długości jest dzielona przez dzielnik pojedyńczej długości, dając w wyniku iloraz i RESZ

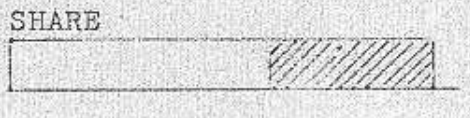
przed



	DVD	5	SHARE

po





Przede wszystkim należy zwrócić uwagę na to, że rozkaz zwraca się /adresuje się/ do akumulatora bardziej znaczącego podwójnej długości dzielnej, a następnie, że iloraz pojedyńczej długości pojawi się w akumulatorze mniej znaczącym. Jak mogłeś oczekiwać, dzielnik pozostaje bez zmiany.

Przykład

Wielkość szterlingowa przechowywana jest w formie podwójnej długości w komórkach SAVE, SAVE+1 jako liczba pensów. Zamień tę liczbę na funty, szylingi i pensy i zapamiętaj w komórkach POUND, SHILL i PENCE. Współczynniki zamiany 12 i 20 przechowywane są w komórkach CF12 i CF20. Iloraz pierwszego dzielenia może być zapamiętany w jednym słowie.

Rozwiąż przykład na arkuszu programowym.

Jako pierwszy krok - załaduj pensy do akumulatorów

	LDX	7	SAVE+1	
	LDX	6	SAVE	

Teraz zastosuj DVD, zwracając się do akumulatora bardziej znaczącego i dzieląc przez dwanaście, aby zamienić pensy na szylingi.

	DVD	6	CF12	

Reszta jest następnie zapamiętana w PENCE

	STO	6	PENCE	

Akumulator 6 zawiera w dalszym ciągu resztę. Reszta musi być zerowana, w przeciwnym razie będzie interpretowana jako część dzielnej.

	LDN	6	0		

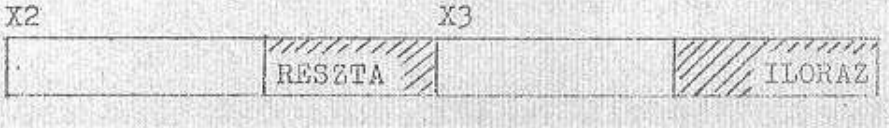
Jeżeli teraz podzielimy przez 20, szylingi pojawią się w akumulatorze 6, a funty w akumulatorze 7. Zapamiętanie funtów i szylingów kończy przykład.

	DVD	6	CF20	
	STO	6	SHILL	
	STO	7	POUND	

DVR

Między rozkazami DVR i DVD jest tylko niewielka różnica.

Reszta i iloraz zajmują miejsce jak poprzednio.



ale, używającDVR , otrzymujemy iloraz zaokrąglony, jak w rozkazie MPR.

Jeżeli dzielimy 19 przez 6,

DVD daje 3, reszta 1
DVR daje 3, reszta 1

Obie odpowiedzi są takie same, ponieważ $3\frac{1}{6}$ zaokrąglone do najbliższej liczby całkowitej jest 3.

Jeżeli teraz podzielimy 23 przez 6

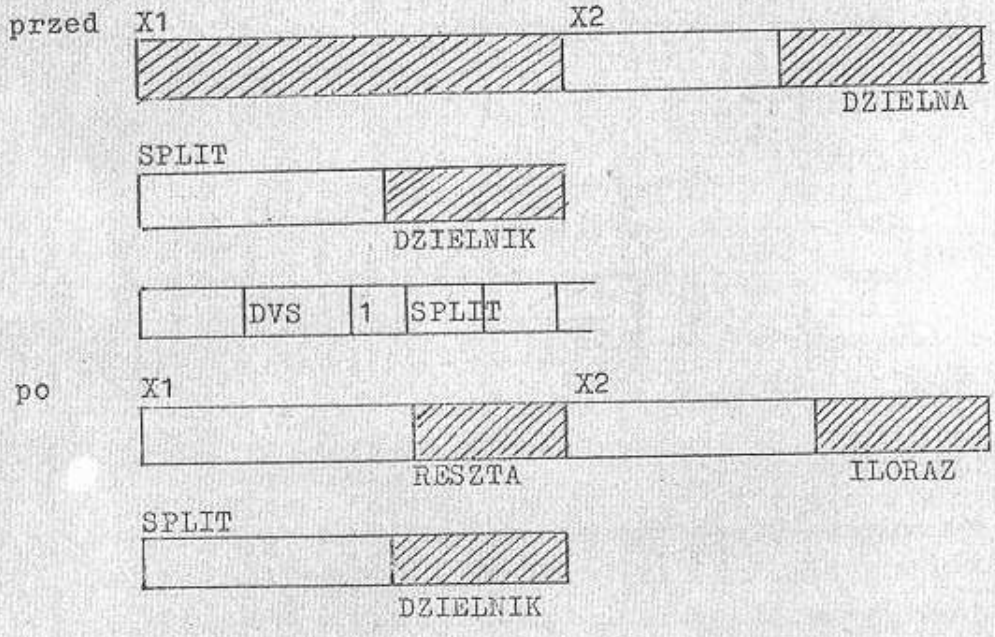
DVD daje 3, reszta 5
DVR daje 4, reszta 5

DVR daje inną odpowiedź, niż DVD, ponieważ $3\frac{5}{6}$ zaokrąglone do najbliższej liczby całkowitej jest 4.

DVS

Rozkaz ten daje ten sam efekt co DVD, działa jednak na dzielnej o pojedyńczej długości. Rozkaz musi zwracać się do akumulatora, który jest o jeden mniejszy od akumulatora zawierającego dzielną.

Zawartość tego akumulatora w procesie dzielenia jest niszczona, ażeby dać miejsce na resztę.



Przykłady

1. Komórki ASSET, ASSET+1 przechowują ilość pensów. Zamień tę ilość na funty, czylingi i pensy i zapamiętaj w komórkach POUND, SHILL i PENCE. W komórkach CF12 i CF20 znajdują się 12 i 20. Iloraz pierwszego dzielenia nie będzie przekraczał pojemności jednego słowa. /3.4/

2. PPodziel liczbę całkowitą podwójnej długości w SALES, SALES+1 przez liczbę całkowitą pojedyńczej długości w ITEMS. Zapamiętaj zaokrąglony wynik w MEAN. /0,1/

3. Liczby całkowite pojedyńczej długości są przechowywane w komórkach NUMA, NUMB. Podziel NUMA przez NUMB. Umieść iloraz w QUOT, a resztę w REM.

Odpowiedzi:

1.

	LDX	4	ASSET+1	
	LDX	3	ASSET	
	DVD	3	CF12	
	STØ	3	PENCE	
	LDN	3	0	
	DVD	3	CF2C	
	STØ	3	SHILL	
	STØ	4	PØUND	

2.

	LDX	1	SALES+1	
	LDX	0	SALES	
	DVR	0	ITEMS	
	STØ	1	MEAN	

3.

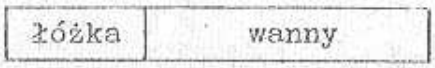
	LDX	5	NUMA	
	DVS	4	NUMB	
	STØ	4	REM	
	STØ	5	QUØT	

Pakowanie

Normalnie, słowo maszyny używane jest do zapamiętania tylko jednej liczby, chociaż, jak to właśnie widzieliśmy, liczba, jeżeli taka jest sytuacja, może być zapamiętana w dwóch słowach.

Innym odchyleniem od normy spotykanym w przetwarzaniu kartotek jest przypadek często występujący, że dwie lub więcej liczb "pakowanych" jest do jednego słowa. Pakowanie to oszczędza miejsce na taśmie magnetycznej i, co jest jeszcze ważniejsze, oszczędza czas wejścia i wyjścia.

Jeżeli mamy kartotekę, w której zapisywany jest bieżąco stan np. łóżek i wanien, wówczas zamiast tracić słowo na każdy z nich, jeżeli maksymalna wielkość każdego z nich na to pozwala, "pakujemy" oba do tego samego słowa.



Założyliśmy tutaj, że ilość łóżek potrzebuje maximum tylko 8 pozycji bitowych i, że dla ilości wanien pozostaje reszta, 16 bitów.

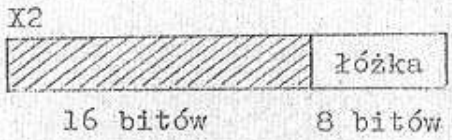
Funkcje przesunięć cyklicznych.

SLC /Przesuń cyklicznie w lewo/

Zobaczymy jak dokonywane jest to pakowanie.

Na początek dwie ilości są w oddzielnych komórkach, które nazwiemy BEDS i BATHS.

Możemy przenieść zawartość słowa BEDS do akumulatora 2 za pomocą rozkazu LDX, aby liczba łóżek zajmowała nie więcej, niż 8 najmniej znaczących pozycji.

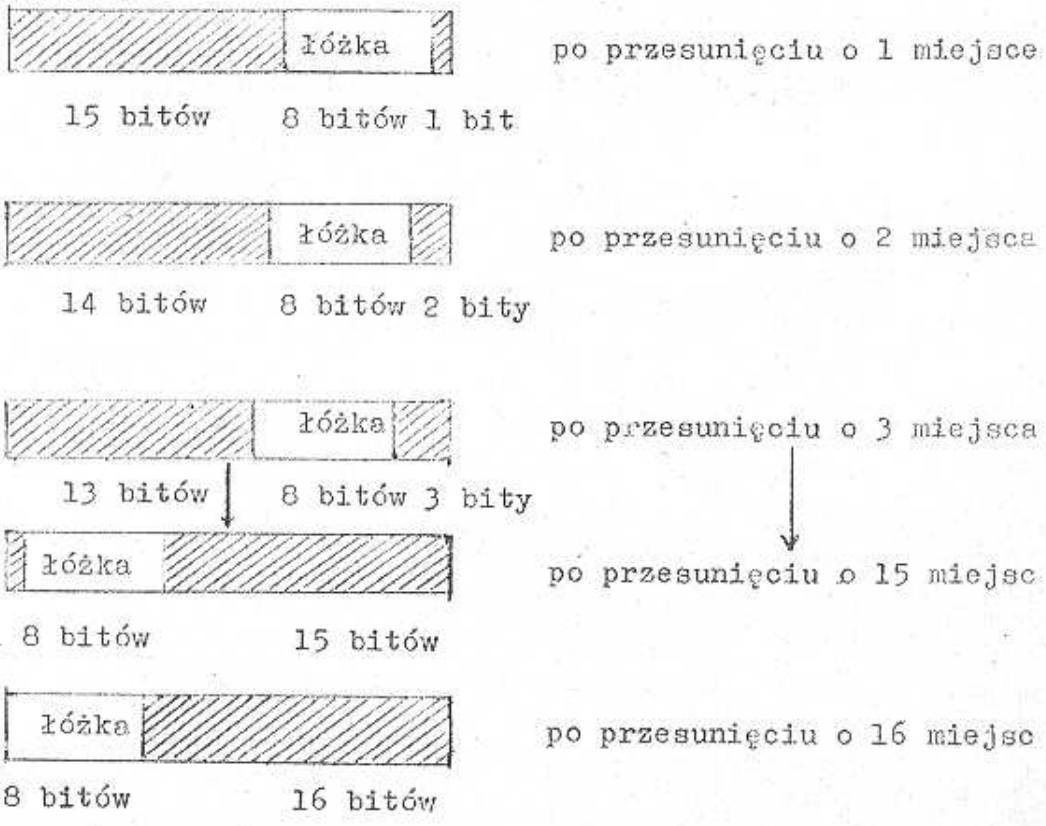


Następnie może być użyty rozkaz przesunięcia, aby spowodować zajęcie przez liczbę łóżek najwięcej znaczących miejsc akumulatora 2. Piszemy

```
SLC     2     16
```

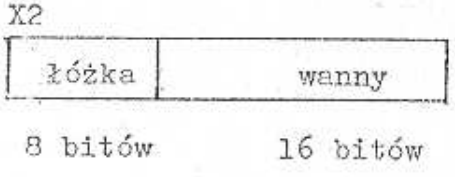
SLC jest mnemonicznym odpowiednikiem dla "przesuń cyklicznie w lewo". Cyfry w każdej pozycji bitowej akumulatora 2 są przesuwane o 1 miejsce w lewo. Wypadający bit z krańca lewej strony jest natychmiast wprowadzany z prawej strony. Wyżej napisany rozkaz wymaga powtórzenia czynności przesuwania 16 razy.

Efekt rozkazu jest łatwo zrozumiany, jeżeli widzimy go w "zwolnionym tempie".



Rozkazy przesuwania działają tylko na liczbach przechowywanych w akumulatorach i dlatego nie ma sensu przesuwanie o więcej, niż 23 miejsca.

Aby zakończyć pakowanie, możemy teraz dodać zawartość komórki BATHS. Ponieważ ilość wanien wymaga nie więcej jak 16 pozycji bitowych, dodanie tej liczby nie będzie miało wpływu na liczbę łóżek.

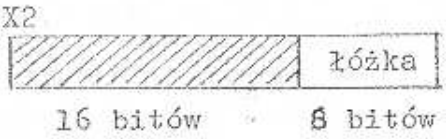


Oto pełny kod dla operacji pakowania.

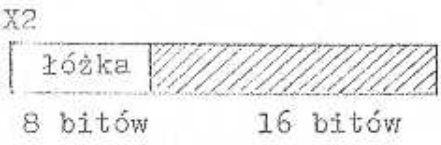
	LDX	2	BEDS		
	SLC	2	16		
	ADX	2	BATHS		

Na wynik przesunięcia cyklicznego w lewo o 16 miejsc można spojrzeć w inny sposób.

Tu początkowy stan akumulatora.

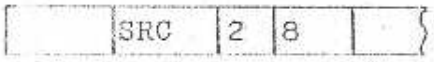


a tu, stan końcowy



Rozkaz spowodował, że grupa 16 bitów z lewej strony zamieniła miejsca z grupą /24-16/ bitów z prawej strony.

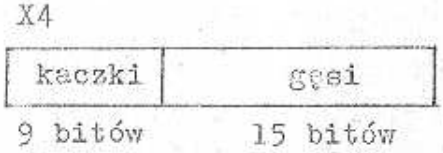
Ponieważ istnieje w Kodzie Rozkazów rozkaz, który pozwala na cykliczne przesuwanie w prawo /SRC/, ten sam efekt może być osiągnięty przez danie rozkazu,



powodując, że grupa 8 bitów z prawej strony zmieni miejsca z grupą /24-8/ bitów z lewej strony. W tym przypadku rozkaz SRC ma tę przewagę nad rozkazem SLC, że przesuwanie o 8 pozycji wymaga krotnego czasu, niż przesunięcie o 16 pozycji.

Przykłady:

1. Liczba kaczek zajmuje do 9 pozycji bitowych, a liczba gęsi zajmuje do 15 pozycji. Upakuj te dwie liczby do akumulatora, jak to pokazano w przykładzie. Wykonaj ćwiczenie najpierw używając rozkazu SLC, a następnie powtórz ćwiczenie, używając rozkazu SRC.



2. Trzy liczby w komórkach DBEAR, MBEAR i BBEAR mogą być reprezentowane odpowiednio przez 12, 8 i 4 bity. Upakuj je do akumulatora 5 jak pokazano na stronie następnej. Wykonaj ćwiczenie najpierw używając rozkazu SLC, a następnie powtórz ćwiczenie, używając rozkazu SRC.

X5

Odpowiedzi:

1.

	LDX	4	DUCKS
	SLC	4	15
	ADX	4	GEESE

lub

	LDX	4	DUCKS
	SRC	4	9
	ADX	4	GEESE

2.

	LDX	5	DBEAR
	SLC	5	8
	ADX	5	MBEAR
	SLC	5	4
	ADX	5	BBEAR

lub

	LDX	5	BBEAR
	SRC	5	4
	ADX	5	MBEAR
	SRC	5	8
	ADX	5	DBEAR
	SRC	5	12

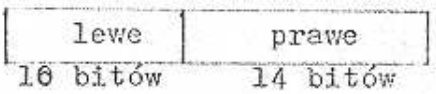
Funkcje "logiczne" przesuwania

SLL /Przesuń w lewo logicznie/

Funkcje przesuwania cyklicznego znalazły swoje zastosowanie w pakowaniu. Przesunięcia "logiczne" używane są do rozpakowania t.j. wydzielania oddzielnych ilości pakowanych w jedno słowo.

Załóżmy, że komórka RECE zawiera dwie liczby, które mają być wydzielone i zapamiętane w komórkach LEWE i PRAWE.

RECE



Zawartość komórki RECE umieszczana jest w akumulatorze przy pomocy rozkazu LDX.

X1

lewe	prawe
10 bitów	14 bitów

Następnie użyty jest rozkaz SLL /przesuń w lewo logicznie/

	SLL	1	10	

aby dać w efekcie

X1

prawe	0000000000
14 bitów	10 bitów

Pozostawiamy nasz problem w tym miejscu.

Jeżeli umieścimy znów wydzieloną ilość z prawej strony akumulatora, możemy ją znów zapamiętać. Kodowanie jest następujące:

	SRC	1	10	
	STO	1	PRAWE	

SRL /Przesuń w prawo logicznie/

Aby wydzielić pozostałą ilość wprowadzamy znów zawartość komórki RECE do akumulatora

X1

lewe	prawe
10 bitów	14 bitów

używając rozkazu SRL /przesuń logicznie w prawo/,

	SRL	1	14	

otrzymujemy w efekcie

X1

00000000000000	lewe
14 bitów	10 bitów

Ilość "lewa" pojawia się po prawej stronie akumulatora, podczas gdy ilość "prawa" została wyzerowana. Ilość "lewa" może być teraz zapamiętana bez używania dalszych przesunięć.

Oto kodowanie dla wydzielenia i zapamiętania obu ilości:

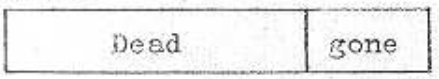
```
        LDX   1   RECE
        SLL   1   10
        SRC   1   10
        STO   1   PRAWE
        LDX   1   RECE
        SRL   1   14
        STO   1   LEWE
```

Teraz nastąpi kilka przykładów.

Przykłady:

1. Wydzielić dwie wielkości upakowane w komórce GRAVE

GRAVE

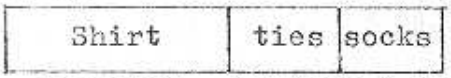


17 bitów 7 bitów

i zapamiętać w komórkach DEAD i GONE. /0/

2. Wydzielić trzy ilości upakowane w komórce DRESS

DRESS



13 bitów 6 bitów 5 bitów

i umieścić te ilości w komórkach SHIRT, TIES i SOCKS. /1/

Odpowiedzi

1.

```
        LDX   0   GRAVE
        SLL   0   17
        SRL   0   17
        STO   0   GONE
        LDX   0   GRAVE
        SRL   0   7
        STO   0   DEAD
```

2.

```
      LDX   1   DRESS
      SLL   1   19
      SRL   1   19
      STO   1   SOCKS
      LDX   1   DRESS
      SLL   1   13
      SRL   1   18
      STO   1   TIES
      LDX   1   DRESS
      SRL   1   11
      STO   1   SHIRT
```

Arytmetyczne funkcje przesuwania

Jest dobrze znanym faktem, że pobranie liczby dziesiętnej i przesunięcie jej cyfr o 1 miejsce w lewo, mnoży liczbę przez 10.

Efekt przesuwania osiągany jest przez umieszczanie zera na końcu liczby. Tak więc, umieszczając zero na końcu 587 np. 5870 daje odpowiedź na 587 x 10.

Przesunięcie w lewo o 2 miejsca mnoży przez 100. Dowolne mnożenie przez potęgę 10 jest możliwe przez przesuwanie odpowiednią ilość razy. Dzielenie przez 10 dokonywane jest przez przesuwanie w prawo.

Ten sam efekt występuje w systemie binarnym, chociaż oczywiście przesunięcie cyfr liczby binarnej o jedno miejsce w lewo, mnoży liczbę przez 2, a nie przez 10. Przesuwanie o dwa miejsca w lewo mnoży przez 4 i podobnie dla innych potęg 2.

SLA, SRA /Przesuń arytmetycznie w lewo i przesuń arytmetycznie w prawo/

Przesuwanie przedstawia odpowiednią metodę mnożenia /i do pewnego stopnia dzielenia/ przez potęgę 2.

Okazyjnie, przesunięcia cykliczne i logiczne mogą być używane do mnożenia i dzielenia, ale najlepiej pozostawić dzielenie i mnożenie przesunięciom arytmetycznym SLA i SRA /przesuń arytmetycznie w lewo i przesuń arytmetycznie w prawo/.

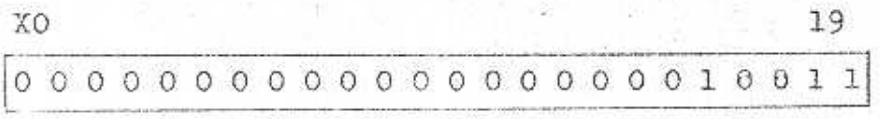
Nie ma potrzeby wchodzić w szczegóły tych dwu funkcji, ale zapew-

niają one, że liczba, która jest dzielona lub mnożona, zachowuje swój początkowy znak. Dokonywane jest to przez trzymanie bitu znaku ustalonego tak, że nie bierze on udziału w przesuwaniu. W operacji dzielenia otrzymujemy także wynik zaokrąglony.

W przykładzie tym 19 jest dzielone przez 4 przez przesunięcie wszystkich cyfr 19 o dwa miejsca w prawo. 19 jest w akumulatorze, ponieważ funkcje przesuwania mogą operować tylko na liczbach będących w akumulatorach. Chociaż możemy otrzymać tylko odpowiedź w liczbach całkowitych, odpowiedź jest zaokrąglona, aby mogła być tak dokładna, jak to jest możliwe. W tym przypadku otrzymujemy 5 zamiast dokładnej odpowiedzi $4\frac{3}{4}$.

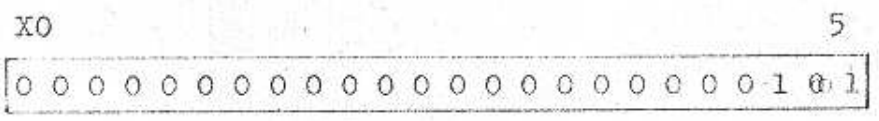
Zaokrąglenie jest dokonywane w ten sposób, że ostatni bit wysuwany jest dodawany z powrotem do wyniku.

przed



	SRA	0	2	

po



Przy mnożeniu należy uważać, aby odpowiedź nie przekraczała pojemności jednego słowa, w przeciwnym przypadku otrzymywana jest odpowiedź nieprawidłowa.

Przykłady:

1. Pomnóż liczbę w X7 przez 2. /Nie będzie przepełnienia/.

2. Pomnóż liczbę w X6 przez 32. /Nie ma przepełnienia/.

3. Podziel liczbę w X0 przez 8. Jeżeli tą liczbą jest 70, jaką otrzymamy odpowiedź?

4. Podziel liczbę w X5 przez 128. Jeżeli tą liczbą jest 260, jaką otrzymamy odpowiedź?

Odpowiedzi

1. | | SLA | 7 | 1 | |

2. | | SLA | 6 | 5 | |

3. | | SRA | 0 | 3 | |

odpowiedź = 9

4. | | SRA | 5 | 7 | |

odpowiedź = 2

Przesunięcia podwójnej długości

SLC i SRC /Przesuń cyklicznie w lewo i przesuń cyklicznie w prawo - podwójnej długości/

Wszystkie przesunięcia omówione dotychczas mogą operować na 2 akumulatorach. Używane są te same skróty mnemoniczne, jak dla przesunięć pojedyńczej długości, ale w formacie rozkazu muszą być wymienione 2 akumulatory. Możliwe jest przyjęcie, jak zrobiliśmy to w tym przykładzie, że akumulator 7 i akumulator 0 znajdują się obok siebie. Zakładamy, że akumulatory te zawierają osiem sześcio-bitowych znaków, reprezentujących litery A do H.

przed

X7				X0			
A	B	C	D	E	F	G	H
6	6	6	6				

	SLC	70	12	

po

X7				X0			
C	D	E	F	G	H	A	B
6	6	6	6				

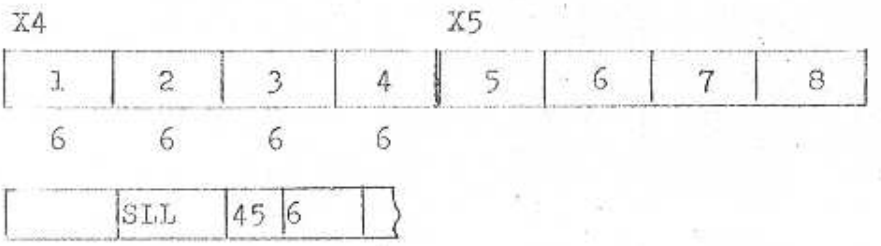
SRC operuje w podobny sposób.

SLL /Przesuń logicznie w lewo - podwójnej długości/

Znów używamy 6 bitowych znaków dla zilustrowania przesunięć lo-

gicznych podwójnej długości. Tym razem reprezentowane są cyfry od 1 do 8.

przed

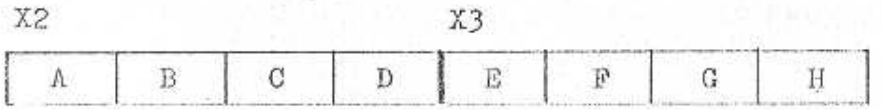


po

Sześć pozycji bitowych na prawym krańcu akumulatora 5 jest wyzerowanych. W schemacie piszemy zero, ponieważ 6 bitowy kod dla zera jest 000000 /patrz załącznik C/.

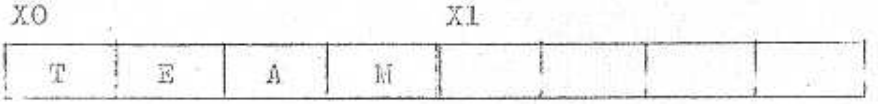
Przykłady

Pierwsze cztery przykłady zaczynają się od tego, że akumulatory 2 i 3 zawierają sześcio-bitowe znaki A do H.



Użyj funkcji przesuwania, aby zamienić zawartość na

1. D E F G H A B C
2. G H A B C D E F
3. B C D E F G H zero
4. zero zero zero zero zero A B C
5. Akumulator 0 zawiera sześcio-bitowe znaki reprezentujące litery T, E, A, M. Akumulator 1 zawiera zera.



Przegrupuj litery w akumulatorze 0 tak, aby otrzymać MEAT.

Odpowiedzi

1.

	SLC	23	18	

2.

	SRC	23	12	

3.

	SLL	23	6	

4.

	SRL	23	30	

5. Jest to jedno z wielu możliwych rozwiązań.

					TSAM- - -	
	SLC	0	6		EAMT- - - -	
	SRC	01	6		-EAMT- -	
	SRC	0	6		M-EAT- - -	
	SRC	01	12		- -M-EAT-	
	SRC	0	6		- - -MEAT-	
	SLC	01	18		MEAT- - - -	
					X0	X1

SLA, SRA /Przesuń w lewo i Przesuń w prawo arytmetycznie - podwójnej długości/

Jak można było oczekiwać arytmetyczne funkcje przesunięć, gdy operują na dwóch akumulatorach, traktują zawartość akumulatorów jako liczbę podwójnej długości. Oznacza to, że pozycja bitowa znaku w słowie mniej znaczącym jest ignorowana. Jak to miało miejsce w przypadku przesunięć arytmetycznych pojedyńczej długości, bit znaku liczby podwójnej długości jest zachowany w jego ustalonej początkowej wartości.

przed

X6 X7

000000000000000000000000	010000000000000000000000

	SLA	67	1	

po

X6	X7
00000000000000000000001	0000000000000000000000

W przykładzie tym 4.194.304 zostało pomnożone przez 2 przez przesunięcie o 1 miejsce w lewo. Pozorne przesunięcie o 2 miejsca /patrz schemat powyżej/ jest wyjaśnione przez fakt, że pozycja bitu znaku w mniej znaczącym słowie jest ignorowana i nie bierze udziału w przesuwaniu. Rozkaz SRA podwójnej długości tworzy odpowiedzi zaokrąglone w ten sposób jak jego pojedyńczy odpowiednik.

Przykłady

1. Pomnóż przez 2 liczbę podwójnej długości zapamiętaną w komórkach SUM, SUM+1 /3.4/

2. Podziel przez 8 liczbę podwójnej długości zapamiętaną w komórkach TOTAL, TOTAL+1. /7.0/

Odpowiedzi

1.

	LDX	4	SUM+1		
	LDX	3	SUM		
	SLA	34	1		

2.

	LDX	0	TOTAL+1		
	LDX	7	TOTAL		
	SRA	70	3		

Omówiliśmy teraz wszystkie rozkazy przesuwania oprócz specjalnych przesunięć SRAV /pojedyńczej i podwójnej długości/, które w tej książce nie będą omawiane.

Funkcje logiczne

Logiczne AND /i/

Funkcje logiczne otrzymują ich nazwę dzięki podobieństwu z operacjami używanymi w logice matematycznej.

Patrz na to zdanie -

Róże są czerwone "i" fiołki są niebieskie.

"I" między dwiema częściami zdania uważa się za podobne do znaku + między liczbami w wyrażeniu arytmetycznym 5 + 3.

Jednak 5 + 3 oznacza operację i daje wynik: 5 + 3 = 8.

Jeżeli "I" w powyższym zdaniu uważane jest za operację podobną do operacji arytmetycznych, to jaki tworzy ono rodzaj wyniku?

Logika zajmuje się rozumowaniem i tym, czy wnioski są prawdziwe, czy fałszywe.

Czy możemy dojść do jakichkolwiek wniosków, czy zdanie

"Róże są czerwone "i" fiołki są niebieskie"

jest prawdziwe, czy fałszywe?

Prawda lub fałsz zdania zależy oczywiście od prawdy lub fałszu obu części rozdzielonych przez "I".

Ponieważ każda część musi być albo prawdziwa albo fałszywa, nie jest trudno wyliczyć wszystkie możliwe przypadki i zdecydować, czy całe zdanie jest prawdziwe, czy fałszywe.

Od operacji I, więc, oczekiwane są tylko dwa możliwe wyniki: prawdziwy lub fałszywy.

Róże są czerwone I fiołki są niebieskie.

Jeżeli pierwsza część jest prawdziwa, a druga część jest fałszywa, całe zdanie jest fałszywe. Ten sam rezultat otrzymujemy, gdy pierwsza część jest fałszywa, a druga część jest prawdziwa i gdy obie części są fałszywe. Tylko, gdy obie części są prawdziwe, całe zdanie jest prawdziwe.

Zestawiając te wyniki w formie tablicy i używając T i F dla określenia prawdy lub fałszu, otrzymujemy

T I F = F

F I T = F

F I F = F

T I T = T

Jeżeli zastąpimy T przez 1, a F przez 0, otrzymamy

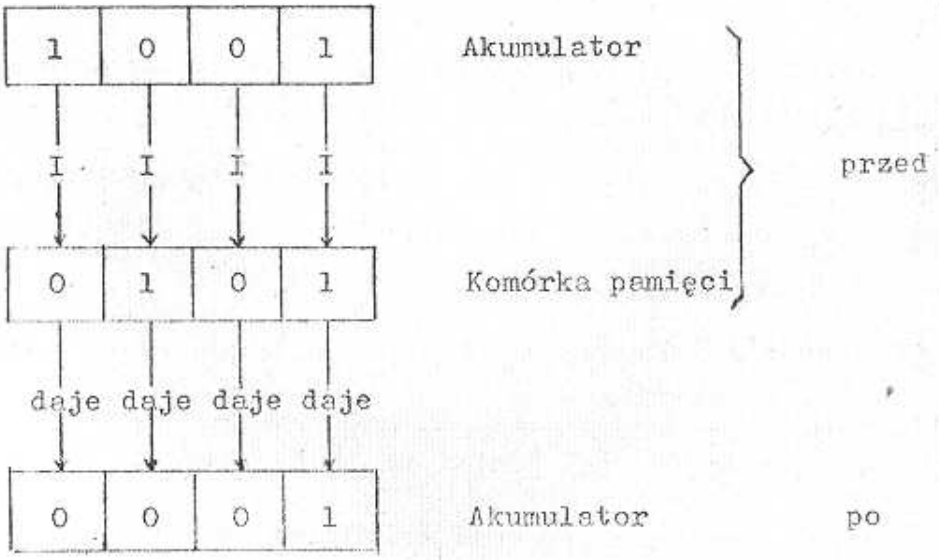
1 I 0 = 0

0 I 1 = 0

0 I 0 = 0

1 I 1 = 1

Ostatnia tablica pokazuje, co zdarzy się, gdy korespondujące bity dwóch słów maszyny porównywane są w czasie wykonywania operacji logicznego I.



ANDX /Logiczne I do akumulatora/

Dla operacji tej jedna z liczb musi być w akumulatorze. Korespondujące bity w akumulatorze i w komórce pamięci są porównywane, a wynik każdego porównania umieszczany jest w akumulatorze. Zawartość komórki pamięci jest niezmieniona.

przed

X3

1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 0 0 0 1 0 0 1 1 0

MASK

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1

	ANDX	3	MASK		

po

X3

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0

Funkcje logiczne są głównie używane w pakowaniu i rozpakowywaniu jako alternatywy lub funkcje posiłkowe dla funkcji przesuwania. Załóżmy, że komórka pamięci PACK zawiera dwie ilości, lewą i prawą.

PACK

lewa	prawa
10 bitów	14 bitów

PACK

0111011100	11110111101111
10 bitów	14 bitów

Ilość prawa ma być wydzielona i zapamiętana w komórce RIGHT.

Załóżmy także, że akumulator jest załadowany następująco:

X7

0000000000	11111111111111
10 bitów	14 bitów

Układ tego rodzaju może być zapamiętany na początku programu jako stała.

Następnie dany jest rozkaz ANDX

	ANDX	7	PACK		

i akumulator 7 będzie zawierał

X7

0000000000	11110111101111

t.j. tylko ilość prawą, która ma być zapamiętana.

Ta operacja wydzielania lub wyciągania bitów przy pomocy funkcji logicznego I znana jest jako maskowanie.

Układ bitów w akumulatorze 7 używany do wydzielania nazywa się maską.

Przykłady:

1. X1

1 1 0 0 1 0 1 1 0 0 1 0 1 1 0 0 1 0 1 1 0 0 1 0

MASK

0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1

zapisz wartość akumulatora 1 po wykonaniu następującego rozkazu

	ANDX	1	MASK		

2. Akumulator 5 i komórka CELL przedstawiają się następująco:

X5

0 0 0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1 1 1 1

13 bitów

CELL

chleb	woda

11 bitów 13 bitów

Wydziel ilość "woda" i zapamiętaj w komórce WATER.

Odpowiedzi

1. X1

0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0

2.

	ANDX	5	CELL			
	STO	5	WATER			

ANDS

Operacja logicznego I może być także wykonana przy użyciu rozkazu

ANDS /logiczne I do pamięci/ z umieszczeniem wyniku w komórce pamięci. Wynik pojawia się w komórce pamięci, a zawartość akumulatora jest niezmieniona.

przed

XO

1 1 1 0 0 0 1 1 1 0 0 1 1 1 1 0 0 0 1 1 1 0 0 1

MASK

1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

	ANDS	0	MASK		

po

MASK

1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Przykłady

1. Akumulator 3 i komórka pamięci CUT przedstawiają się następująco:

X3

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

8 bitów

CUT

nóż	widelec	łyżka

8 bitów 8 bitów 8 bitów

Oczyść komórkę CUT z ilości noży i widelców.

Odpowiedzi

1.

	ANDS	3	CUT		

Logiczne OR /Lub/

Róże są czerwone "lub" fiołki są niebieskie.

Zdanie, z którym zaczęliśmy omawiać funkcje logiczne, zostało zmienione tylko przez podstawienie LUB zamiast I.

LUB jest rozważane także dla określenia operacji między dwiema częściami zdania.

Znów rysowana jest tablica dla pokazania prawdy lub fałszu całego zdania w zależności od prawdy lub fałszu dwu części. Podstawienie LUB zamiast I czyni zdanie fałszywym tylko wtedy, gdy obie części są fałszywe

T LUB F = T

F LUB T = T

F LUB F = F

T LUB T = T

i podstawiając 1 zamiast T i 0 zamiast F, widzimy wyniki osiągnięte przez maszynę, gdy porównuje ona korespondujące bity w dwóch słowach za pomocą operacji logicznego LUB.

1 LUB 0 = 1

0 LUB 1 = 1

0 LUB 0 = 0

1 LUB 1 = 1

ORX /Logiczne LUB do akumulatora/

Wynik pojawia się w akumulatorze, a zawartość komórki pamięci pozostaje niezmieniona.

przed

X5

1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0

STORE

0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1

	ORX	5	STORE		

po

X5

1	1	1	0	0	0	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1

Rozkaz logicznego LUB może być używany w pakowaniu, jak pokazano w następnym przykładzie, gdzie chcemy upakować dwie 12-bitowe ilości pantofli i butów przechowywanych w komórkach BOOT i SHOE do komórki BOX.

Pakowanie dokonane jest przez kodowanie:

	LDX	6	BOOT		
	SLC	6	12		
	ORX	6	SHOE		
	STO	6	BOX		

W przykładzie tym rozkaz ADX spowodowałby dokładnie to samo, co ORX. Niestety, zbyt szybko zaszlibyśmy za daleko, gdybyśmy usiłowali dać przykład, gdzie ORX nie mógłby być zastąpiony przez ADX.

ORS /Logiczne LUB do pamięci/

ORS wykonuje operację logicznego LUB między zawartością akumulatora i zawartością komórki pamięci. Odpowiedź pojawia się w komórce pamięci, a zawartość akumulatora pozostaje niezmieniona.

Przykłady

1. X7

1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0

STORE

1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0

Napisz zawartość akumulatora 7 po wykonaniu następującego rozkazu:

	ORX	7	STORE	

2. TOPSY

0000000000000000	topsy
	8 bitów

TURWY

00000000	turwy
	16 bitów

XO

topsy	turwy
8 bitów	16 bitów

Użyj rozkazu ORX dla zapakowania topsy i turwy do akumulatora 0, jak pokazano powyżej.

Odpowiedzi

1. X7

1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0

2.

	LDX	0	TOPSY	
	SLC	0	16	
	ORX	0	TURWY	

Logiczne EXKLUZYWNE LUB /Wyłączające/

Hitler zmarł LUB Hitler żyje.

LUB w tym zdaniu nie jest takie samo jak LUB inkluzywne /włączające/ uprzednio omawiane.

Użycie LUB wyłączającego oznacza, że całe zdanie jest prawdziwe tylko wtedy, gdy jedna część jest prawdziwa, ponieważ prawda jednej części wyklucza prawdę drugiej części. Niemożliwe przypadki, że dwie części są prawdziwe i dwie części są fałszywe, oznaczane są jako fałszywe.

Dla odróżnienia między LUB włączającym i wyłączającym, to ostatnie oznaczamy przez ER.

T ER F = T

F ER T = T

F ER F = F
T ER TT = F

i podstawiając 1 zamiast T i 0 zamiast F otrzymujemy.

1 ER 0 = 1
0 ER 1 = 1
0 ER 0 = 0
1 ER 1 = 0

ERX /Logiczne LUB wykluczające do akumulatora/

Wynik pojawia się w akumulatorze, a zawartość komórki pamięci jest niezmieniona.

przed

X6

1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0

STORE

1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0

	ERX	6	STORE

po

X6

0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0

Korzystając zarówno z ERX jak i ERS /logiczne wyłączające LUB do pamięci/ następujące kodowanie wymienia zawartość akumulatora i komórki pamięci, używając tylko dwóch zaangażowanych komórek.

	ERX	2	STORE
	ERS	2	STORE
	ERX	2	STORE

Dla większej jasności, w schemacie wyjaśniającym poniżej zakładamy, że słowo Maszyny jest 4-bitowe.

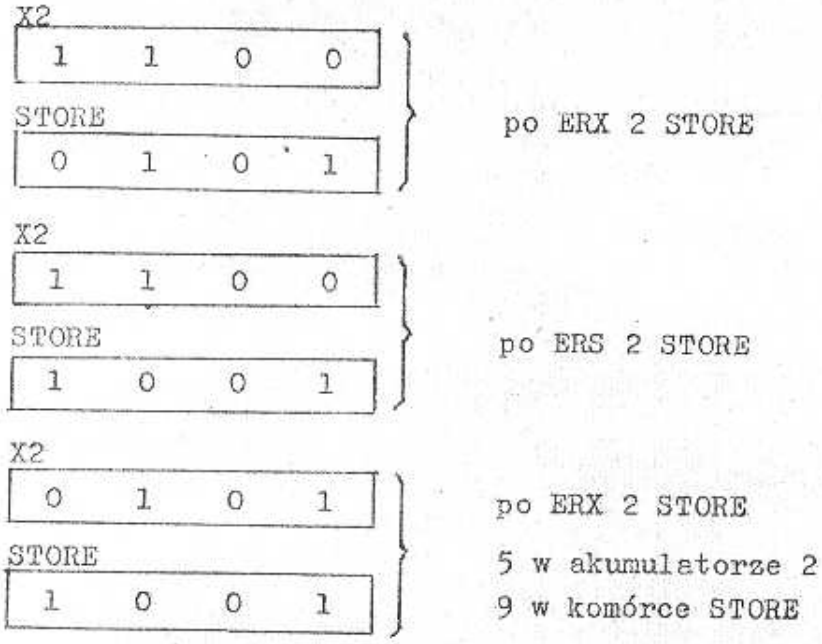
X2 9

1	0	0	1

STORE 5

0	1	0	1

Początkowo:
9 w akumulatorze 2
5 w komórce STORE



Przykłady

1. X3

0 0 1 1 0 0 0 0 1 1 0 0 0 1 0 0 1 0 0 1 0 0 1 0

STORE

1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0

Zapisz zawartość akumulatora 3 po wykonaniu następującego rozkazu

	ERX	3	STORE	

2. Wymień zawartość akumulatora 5 i komórki SWOP, używając rozkazów wyłączającego LUB.

Odpowiedzi

1. 1 1 0 0 1 1 0 0 1 1 0 0 1 0 1 1 0 1 0 1 0 0 1 0

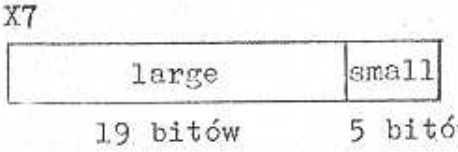
2.

	ERX	5	SWOP	
	ERS	5	SWOP	
	ERX	5	SWOP	

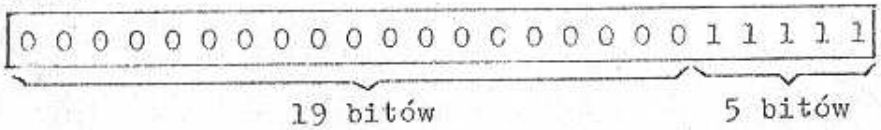
Funkcje logiczne z operandami w formie literali ANDN, ORN, ERN

Są to zwykłe funkcje logicznego I, logicznego LUB i logicznego wyłączającego LUB z tą różnicą, że mają operandy w formie literali. Jak inne funkcje literalne /ADN, SBN, LDN i t.d./ operują one tylko na akumulatorach.

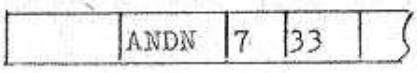
Załóżmy, że akumulator 7 przedstawia się następująco:



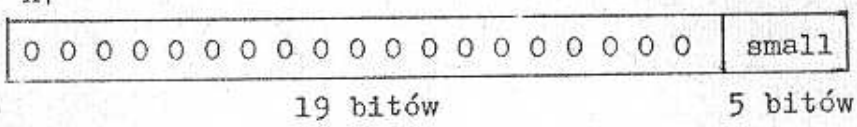
Jeżeli następnie na tym układzie bitów wykonamy logiczne I do akumulatora



wydzielona jest ilość, small. Wartością numeryczną wymaganego układu bitów jest 33. Jeżeli więc napiszemy



otrzymamy w akumulatorze 7



Zauważ, że w ostatnim przykładzie, chociaż byliśmy zainteresowani w układzie bitów, musieliśmy obliczyć ich dziesiętny ekwiwalent, aby użyć ANDN.

Może to być raczej nużące, lecz na szczęście jest na to sposób.

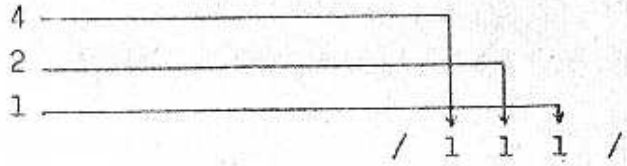
Weźmy inny układ bitów, który ma być użyty z ANDN.

0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0

Zamiast wypracowywać jego dziesiętną wartość, wyobraźmy sobie układ rozbity na grupy, składające się z 3 cyfr,

/0 0 0/0 0 0/0 0 0/0 0 0/1 1 1/1 1 1/0 0 0/0 0 0/

Pozycjom wewnątrz przegródek nadane są następujące znaczenia:



Przegródka ma wtedy wartość

4+2+1=7,

a powyższy układ możemy napisać jako

00007700

albo prościej jako 7700

Dla układu bitów rozbitego w ten sposób jest 8 możliwych wartości.

0 0 0	=	0	1 0 0	=	4
0 0 1	=	1	1 0 1	=	5
0 1 0	=	2	1 1 0	=	6
0 1 1	=	3	1 1 1	=	7

Liczby, do których dochodzimy przez rozbijanie układów binarnych na grupy 3 /trójek/, zwane są liczbami ósemkowymi, ponieważ każda pozycja cyfrowa przybiera jedną z ośmiu wyrtości w przedziale 0 - 7.

Dla uniknięcia pomieszania z liczbami dziesiętnymi, liczby ósemkowe są poprzedzane znakiem =. Tak więc liczba 7700, o której już była mowa, pisana jest jako =7700.

Jeżeli układ ten byłby użyty np. w akumulatorze 1, napisalibyśmy:

	ANDN	1	#7700	

1. Opisz następujące układy bitów jako liczby ósemkowe:

a/	000	000	000	000	111	000	111	000
b/	000	000	000	000	101	000	101	000
c/	000	000	000	000	111	111	100	000
d/	000	000	000	000	000	001	111	111

2. Użyj ANDN i operandów ósemkowych.

a/ X4

ink	pen
17 bitów	7 bitów

Wydziel pen i umieść w komórce PEN.

b/ X5

brush	mop	broom
12 bitów	7 bitów	5 bitów

Wydziel mop i umieść w komórce MOP.

Odpowiedzi

1. a/ #7070 c/ #7740
   b/ #5050 d/ #177

2. a/

	ANDN	4	#177		
	STO	4	PEN		

b/

	ANDN	5	#7740		
	SRC	5	5		
	STO	5	MOP		

Jest granica odnośnie wielkości liczb ósemkowych, które mogą być używane z ANDN, ORN i ERN.

Pamiętaj, że część operanda słowa rozkazu może zawierać tylko 12 bitów.

1 1 1 1 1 1 1 1 1 1 1 1

napisane jako liczba ósemkowa jest #7777; tak więc 7777 jest największą liczbą ósemkową, która może być użyta.

Liczby ósemkowe mogą być używane w operandach wcześniej omówionych funkcji literalnych /ADN, SBN, LDN i t.d./, ale przez ich użycie nie osiąga się prawie żadnych korzyści z wyjątkiem przypadku, gdy LDN jest używane w połączeniu z funkcją logiczną.

W tym miejscu rozstajemy się chwilowo z funkcjami logicznymi.

Funkcje różne

Zerowanie

Przed rozpoczęciem przetwarzania jest często konieczne zerowanie akumulatora lub komórki pamięci. Gdy taka konieczność zaistniała poprzednio, używaliśmy LDN.

Zerowanie akumulatora wymagało tylko jednego rozkazu

	LDN	6	0	

ale zerowanie komórki pamięci, powiedzmy, JANEK, wymagało dwóch rozkazów

	LDN	4	0	
	STO	4	JANEK	

STOZ /Pamiętaj zero/

Lepszą metodą jest używanie rozkazu STOZ, który działa na akumulatorach i na komórkach pamięci.

Aby wyzerować komórkę JANEK, piszemy

	STOZ		JANEK	

i aby zerować akumulator 6

	STOZ		6	

STOZ jest jednym z kilku rozkazów w kodzie, gdzie kolumna akumulatora pozostaje pusta /blank/.

Przykłady

1. Zeruj X2.

2. Zeruj komórkę HOUR.

3. Zeruj komórkę PENCE jak i komórkę po PENCE.

Odpowiedzi

1.

	STOZ		2	

2.

	STOZ		HOUR	

3.

	STOZ		PENCE	
	STOZ		PENCE+1	

MOVE /Przesuń/

Rozkaz MOVE kopiuje liczbę słów z jednego zestawu komórek pamięci do drugiego. Może być używany np. przy przenoszeniu rekordów z obszaru wejścia do obszaru roboczego lub z obszaru roboczego do obszaru wyjścia.

Kopiowanie zawartości 50 słów, zaczynających się od komórki WORK do 50 słów, zaczynających się od komórki OUTPU zaczyna się od wprowadzenia do akumulatora faktycznego numeru komórki WORK. Chociaż nie wiemy jaki jest ten numer, mimo to może on być wprowadzony do akumulatora.

	LDN	2	WORK

Przyjrzyj się dokładnie temu użyciu LDN. Rozkaz ten umieszcza w akumulatorze adres słowa WORK, a nie jego zawartość.

Do następnego akumulatora wprowadzamy adres komórki OUTPU, znów używając LDN.

	LDN	3	OUTPU

Teraz wszystko jest gotowe dla rozkazu MOVE.

	MOVE	2	50	

Akumulatorem wymienionym w rozkazie jest akumulator, zawierający początkowy adres 50 słów, które mają być kopiowane /przenoszone/.

W ten sposób jednym rozkazem MOVE może być przenoszona każda liczba słów aż do 512.

Przykłady

1. Przenieś 66 słów, zaczynając od REC do 66 słów, zaczynających się od PROC. /1,2/

Odpowiedź

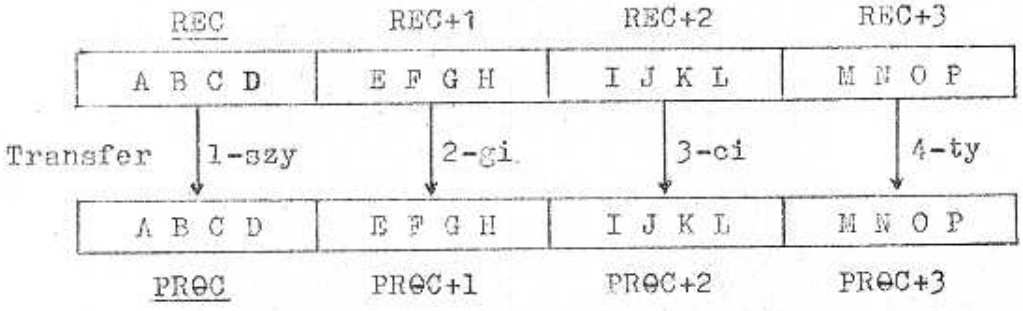
1.

	LDN	1	REC		
	LDN	2	PROC		
	MOVE	1	66		

Zazwyczaj MOVE używany jest do przenoszenia grupy słów z jednego odległego obszaru pamięci do drugiego. Transfer odbywa się następująco:

zawartość REC przenoszona jest do PRØC
zawartość REC+1 przenoszona jest do PRØC+1
zawartość REC+2 przenoszona jest do PRØC+2

i t.d.

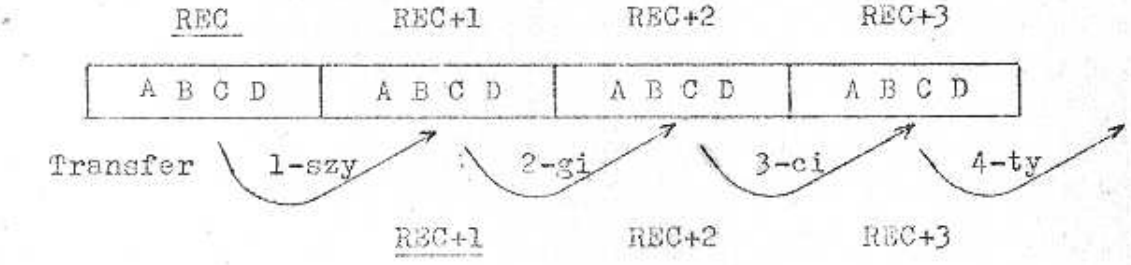


Nie ma jednak powodu, dla którego te dwa obszary nie miałyby zachodzić na siebie.

Przenoszenie słów z obszaru, zaczynającego się w REC do obszaru, zaczynającego się w REC+1 wyglądałoby następująco:

zawartość REC przenoszona do REC+1
zawartość REC+1 przenoszona do REC+2
zawartość REC+2 przenoszona do REC+3

i t.d.



Wynikiem tej operacji jest ciąg słów, zaczynający się od REC, z których wszystkie mają tę samą zawartość.

Zachodzenie na siebie dwu obszarów jest szczególnie użyteczne, gdy mają być zerowane wielkie obszary pamięci.

Zera wprowadzane są do pierwszego słowa rozkazem STOZ, a rozkaz MOVE przenosi zera przez resztę obszaru.

Cztery rozkazy poniżej wyzerują 100 słów pamięci, zaczynając od komórki OUTPT

	STØZ		ØUTPT		
	LDN	0	ØUTPT		
	LDN	1	ØUTPT	+1	
	MØVE	0	99		

Od rozkazu MOVE wymaga się przeniesienia tylko 99 słów, inaczej byłoby zerowane 101 zamiast 100 komórek.

Przykład

1. Zeruj 64 słowa w pamięci, zaczynając od komórki ARRAY. /6,7/

Odpowiedzi

1.

	STØZ		ARRAY		
	LDN	6	ARRAY		
	LDN	7	ARRAY	+1	
	MØVE	6	63		

SUM

Rozkaz SUM kończy chwilowo naszą naukę Kodu Rozkazów.

Efektem tego rozkazu jest umieszczenie w akumulatorze sumy zawartości grupy kolejnych komórek pamięci. Liczba komórek jest nie większą od 512.

Zanim rozkaz może działać, musimy umieścić adres pierwszego słowa, które ma być sumowane w akumulatorze po akumulatorze, w którym ma pojawić się odpowiedź.

Jeżeli wynik przekracza pojemność jednego słowa, a SUM nie ustawia ani rejestru przepełnienia, ani rejestru przeniesienia, otrzymywana jest odpowiedź nieprawidłowa.

Sumowanie zawartości 100 komórek, zaczynających się od CENT, wykonane jest następująco:

	LDN	3	CENT		
	SUM	2	100		

Odpowiedź pojawia się w akumulatorze 2.

Przykład

1. Sumuj zawartość 144 komórek, zaczynających się od GROSS. Wynik nie przekroczy pojemności jednego słowa. Zapamiętaj wynik w komórce TOTAL.

Odpowiedź

	LDN	7	GROSS	
	SUM	6	144	
	STO	6	TOTAL	

Pamięć

Opuszczając na chwilę rozkazy programowe, popatrzmy na urządzenia zrobione dla przydzielania obszaru pamięci danych.

Następny schemat pokazuje jak program przechowywany jest w pamięci w czasie przetwarzania.

Akumulatory
Zarezerwowana dla Execitive
Niższa Pamięć Danych
Pamięć na Program

Widzieliśmy już, że akumulatory zajmują 8 pierwszych słów.

Wszystkie następne 27 słów /komórki 8 - 34/ z wyjątkiem jednego, używane są przez Executive i nie są dozwolone do użytku przez programistę. Chwilowo nie musimy o tym myśleć.

Bezpośrednio interesuje nas Niższa Pamięć Danych.

Niższa Pamięć Danych

Niższa pamięć danych zaczyna się w komórce 35 i może się rozciągać aż do komórki 4095. Liczba 4095 pojawia się znów z tego samego, co uprzednio powodu - jest ona największą liczbą całkowitą, która może pojawić się w części operanda słowa rozkazu.

Jeżeli cały obszar pamięci, aż do komórki 4095 został przydzielony, a więcej obszaru nadal jest wymagane, wówczas tworzona jest WYŻSZA pamięć danych po pamięci na program.

Wyższą pamięcią danych będziemy zajmowali się później, chwilowo możemy przyjąć, że cały nasz obszar jest bezpiecznie schowany gdzieś między komórkami 35 i 4095.

W PLAN 1 wszystkie komórki wymagane dla danych muszą być przedstawione asemblerowi, aby można było przydzielić adres absolutny.

Np. rozkaz STO 7 TONS

nie ma żadnego znaczenie, jeżeli komórka nazwana TONS nie została dla asemblera uprzednio określona. Jeżeli dla TONS przydzielono komórkę 69, rozkaz zażąda, aby zawartość akumulatora 7 została zapamiętana w komórce 69.

Wszystkie obszary danych muszą być więc określone w programie przed rozpoczęciem rozkazów programu.

Dyrektywa LOWER

Aby dokonać przydziału pamięci, piszemy rozkaz, mówiący asemblerowi, że nazwy, które następują, są obszarami pamięci, którym należy przydzielić adresy. Asembler kontynuuje tę pracę, dopóki nie otrzyma rozkazów przeciwnych.

Wszystkie rozkazy dla asemblera nazywane są Dyrektywami.

Dyrektywą używaną do przydzielania obszaru pamięci w niższej pamięci danych jest dyrektywa LOWER.

Jest ona pisana na arkuszu programowym następująco:

⌗LOWER				

Znak # sygnalizuje, że słowo LOWER jest dyrektywą.

Tak więc, wracając do komórki TONS, na arkuszu programowym piszemy dyrektywę LOWER i TONS w polu operanda w następnym wierszu.

#LOWER				
			TONS	

Kompiler przechodzi przez dyrektywę LOWER i gdy napotyka TONS, TONS otrzymuje adres absolutny, powiedzmy, 69. Dalsze odnoszenia do TONS w programie PLAN powodują, że program wynikowy odnosi się do komórki 69.

Przykład

Przydziel komórkę pamięci w niższej pamięci danych dla PENCE.

Odpowiedź

#LOWER				
			PENCE	

Nie ma potrzeby używania oddzielnej dyrektywy LOWER dla każdej komórki. Możemy pisać serie komórek w jednym wierszu, stawiając przecinek po każdej, z wyjątkiem ostatniej.

np.

#LOWER							
			POUND,SHILL,PENCE				

Nie stawiamy przecinka po ostatniej komórce w każdym wierszu, jeżeli seria komórek zajmuje więcej, niż jeden wiersz.

Np.

#LOWER									
			BED,SHEET,RAKE,MOVER,SPADE						
			HUT,COSTA,COSTB,COSTC						

Przykład

Przydziel obszar pamięci w niższej pamięci danych dla SUMA, SUMB,

SUMC, SUMD, SUME, BUN, LOAF, CAKE, TAB1, TAB2, TAB3, SAUCE, SUGAR, SALT.

Odpowiedź

```
#LOWER
      SUMA,SUMB,SUMC,SUMD,SUME,BUN,LOAF,CAKE
      TAB1,TAB2,TAB3,SAUCE,SUGAR,SALT
```

W poprzednich przykładach każdej wymienionej nazwie była przydzielana tylko jedna komórka. Jeżeli potrzeba więcej komórek, żądana liczba pisana jest w nawiasach po nazwie komórki.

Np. przydzielenie po 50 słów dla INPUT i OUTPT i 25 słów dla PROC dokonywane jest następująco:

```
#LOWER
      INPUT(50),OUTPT(50),PROC(25)
```

W programie PLAN do pierwszego słowa INPUT odnosimy się przez INPUT, do drugiego przez INPUT+1, do trzeciego przez INPUT+2 i t.d.

Przykład

1/ Przydziel 10 komórek dla TOES i po 2 dla ARMS i LEGS.

2/ Obszar pamięci, zaczynając od komórki PROC, ma 20 słów. Napisz odpowiedni rozkaz LDX, aby przenieść zawartość trzynastego słowa w obszarze do X4.

Odpowiedzi

1.
```
#LOWER
      TOES(10),ARMS(2),LEGS(2)
```

2.
```
      LDX   4  PROC+12
```

Ograniczenia odnośnie użycia nazw

Należy zwrócić uwagę na kilka ograniczeń odnośnie nazw, używanych dla komórek w PLAN 1.

1. Nazwa nie może mieć więcej, niż 5 znaków.

2. Użyte znaki muszą być albo literami A - Z, lub cyframi 0-9.

3. Każda nazwa musi zaczynać się od litery.

4. Różnym komórkom nie można dawać tej samej nazwy.

Przykład

1. Zadecyduj, która z poniższych nazw nie odpowiada zasadom:

#L0WER											
			P0UND,	0UNCES,	LEVEL,	CF#40,	3PLY,	H912			
			P0UND,	SHILL,	PENCE						

Odpowiedzi

0UNCES ma więcej, niż 5 znaków.
CF#40 zawiera symbol niedozwolony "#".
3PLY nie zaczyna się od litery.
Drugi P0UND jest niedozwolony, ponieważ już jeden jest.

Wszystkie zapisy dokonane na arkuszu programowym pod dyrektywą LOWER nazywane są Oświadczeniami o Danych.

Rozróżniamy dwa typy:

Oświadczenia o danych zmiennych
Oświadczenia o danych stałych.

Oświadczenia o danych, z którymi już spotykaliśmy się, są przykładami pierwszego typu, gdzie komórce dany jest adres bezwzględny w niższym obszarze pamięci.

Oświadczenia drugiego typu wołają o to samo, ale wymagają nadto, aby przydzielona komórka zawierała jakąś określoną wartość. One

.także są pisane pod dyrektywą LOWER, ale w trochę odmienny sposób, jak to zobaczymy za chwilę.

Jeżeli chcemy, aby komórce CF12 przydzielono w obszarze niższej pamięci adres bezwzględny, i aby komórka ta zawierała także 12, piszemy:

#LOWER				
CF12			+12	

W oświadczeniu o danych stałych przyjęło się pisanie nazwy od lewego krańca arkusza programowego. Stałą oznacza się także znakiem + lub -.

Przykłady

1. Dokonaj zapamiętania 144 w komórce GROSS.

Odpowiedź

#LOWER				
GROSS			+144	

W jednym wierszu arkusza programowego może być napisanych więcej, niż jedna stała.

Np. to oświadczenie o danych stałych -

#LOWER							
DATA			+3,	-177,	+408312,	+32	

spowoduje, że: -

komórka DATA będzie zawierała +3
komórka DATA+1 będzie zawierała -177
komórka DATA+2 będzie zawierała +408312
komórka DATA+3 będzie zawierała +32

Gdyby do powyższych wartości stałych odnoszono się przez nazwy różne, wówczas każda nazwa musiałaby pojawić się w oddzielnym wierszu.

#LOWER					
TRID			+3		
CONS			-177		
KAY			+402312		
COUNT			+32		

Przykłady

1. Zapamiętaj stałe: 100, -10 i -319.000 w komórkach CONST, CONST+1, CONST+2.

2. Zapamiętaj stałe z przykładu 1 w komórkach: CENT, FAC, NUM.

Odpowiedzi

1.

#LOWER							
CONST			+100,-10,-319000				

2.

#LOWER							
CENT			+100				
FAC			-10				
NUM			-319000				

Stałe o podwójnej długości mogą być zapamiętane przez pisanie wnnawiasach cyfry 2.

N.p.

#LOWER							
LIMIT			+10000000(2)				

10,000,000 jest zapamiętane w komórkach LIMIT, LIMIT+1 w formie podwójnej długości.

Przykład

1. Zapamiętaj -63,535,207 jako liczbę podwójnej długości w komórkach NEG, NEG+1.

Odpowiedź

#LOWER					
NEG			-63535207(2)		

Liczby ósemkowe

Stałe mogą być podawane także jako liczby ósemkowe. Jak wykazano uprzednio, jest to użyteczne, gdy stałe ma być użyta jako maska. Dopuszczalna jest każda liczba ósemkowa do 77777777. Liczby ósemkowe wyższe od tej nie mogą być umieszczane w jednym słowie.

Aby zapamiętać postać

0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1

jako maskę w komórce MASK, piszemy:

#LOWER					
MASK			#777733		

Przykład

Zapamiętaj następujący układ bitów jako maskę w MASK.

1	1	1	0	1	0	1	1	1	0	1	0	1	1	1	1	0	0	1	1	1	1	0	0

Odpowiedź

#LOWER						
MASK			#72727474			

Ułamki

Ułamki dziesiętne mogą być podawane jako stałe. Będą one przechowywane w ułamkowej formie binarnej z założeniem, że kropka binarna jest w tym miejscu

0.75

0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Po kropce może być pisane 8 cyfr dziesiętnych, ale dokładność jest w przybliżeniu do 7 cyfr.

#LOWER						
FRACT			+.00004872			

Przykład

1. Zapamiętaj 0.98375552 w formie ułamkowej w komórce PROP.

Odpowiedź

1.

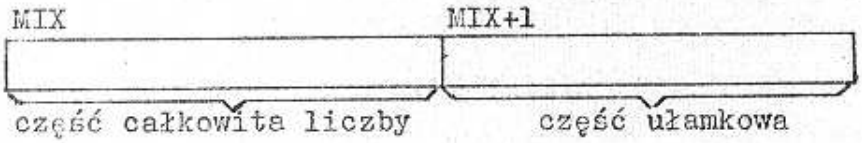
#LOWER						
PROP			+.98375552			

Mieszane liczby dziesiętne

Takie liczby jak 30.46 przechowywane są zazwyczaj w ten sposób, że całkowita część liczby jest w jednym słowie, a ułamkowa część w słowie sąsiednim.

Za każdą mieszaną liczbą dziesiętną na arkuszu programowym muszą następować dwie liczby napisane w nawiasach. Pierwsza liczba określa ile ma być użytych komórek pamięci. Liczbą tą będzie zazwyczaj 2, ale może być i 1. Druga liczba określa ile pozycji cyfr binarnych ma być użytych dla zapamiętania części ułamkowej. Druga liczba może mieć każdą wartość w zakresie 1 do 46. Gdy część ułamkowa umieszczona jest w drugim słowie, ma ona wartość 23.

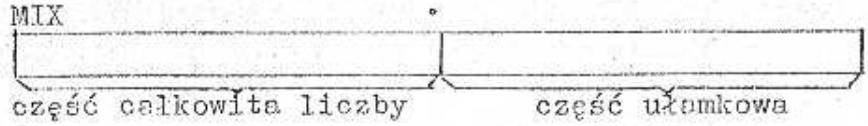
Aby zapamiętać 13.56 w sposób następujący



piszemy:

#LOWER						
MIX			+13.56(2.23)			

ale, aby zapamiętać ją w jednej komórce, w której po połowie zajmują liczba całkowita i część ułamkowa



piszemy

⌗LOWER						
MIX			+13.	56(1	.12)	

Zauważ, że liczby w nawiasach oddzielane są kropką /kropką dziesiętną/, anmie przecinkiem.

Przykłady

1. Zapamiętaj $35\frac{3}{4}$ z częścią całkowitą liczby w FRAG i częścią ułamkową w FRAG+1.

2. Zapamiętaj $35\frac{3}{4}$ z częścią całkowitą liczby zajmującą 18 pozycji od lewej FRAG i z częścią ułamkową, zajmującą pozostałe 6 pozycji.

Odpowiedzi

1.

⌗LOWER						
FRAG			+35.	75(2	.23)	

2.

⌗LOWER						
FRAG			+35.	75(1	.6)	

Znaki

Jeżeli litery A do Z i numeryki 0 do 9 mają być zapamiętane w formie znakowej, wówczas znaki te muszą być poprzedzone przez literę H, która z kolei jest poprzedzana przez liczbę znaków.

Znaki A, 1, B i C mogą być zapamiętane w jednym słowie jak następuje:

FIRST

A	1	B	2

⌗LOWER						
FIRST			4HA1	B2		

Kropki, przecinki, spacje, gwiazdki i t.p., wszystkie one mają swoje 6-bitowe kody i muszą być włączone do licznika znaków przed H.

Komórki MARKS, MARKS+1 są ładowane jak następuje:

MARKS				MARKS+1			
spacja	spacja	$	!	spacja	>	=	+

przez oświadczenie:

#LOWER						
MARKS			8Hvv$!v>=+			

Symbol V używany jest dla oznaczenia spacji na arkuszu programowym.

Jeżeli liczba podanych znaków nie zapełnia całkowitej liczby słów, wówczas pozycje znakowe pozostałe w ostatnim słowie są zapełniane sześcio-bitowym kodem spacji.

Np. oświadczenie:

#LOWER							
MESS			14HINCORRECT DATA				

daje następujący efekt

MESS	I	N	C	O
MESS+1	R	R	E	C
MESS+2	T	spacja	D	A
MESS+3	T	A	spacja	spacja

Przykłady

1. Napisz prawidłowe oświadczenie o danych, aby wypełnić podane komórki tak, jak w schemacie.

STOP	S	T	O	P
STOP+1	spacja	#	2	4
STOP+2	*	*	*	*

2. Wypisz zawartość TRAP, TRAP+1, TRAP+2 po wykonaniu poniższego oświadczenia o danych.

```
#LOWER
TRAP    1OHLONG▽BLOCK
```

Odpowiedzi

1.
```
#LOWER
STOP    12HSTOP▽#24****
```

2.

TRAP	L	O	N	G
TRAP+1	spacja	B	L	O
TRAP+2	C	K	spacja	spacja

Dyrektywa PROGRAM

Dyrektywa PROGRAM pojawia się tylko w dwóch miejscach.

Po pierwsze jest ona używana dla przydzielenia programowi jego nazwy i numeru priorytetu.

Nazwa może składać się z 1 - 4 liter A do Z, podczas, gdy numer priorytetu musi mieć dwie cyfry od 01 do 99.

Jeżeli mamy programowi nazwanemu CALC przydzielić numer priorytetu 77, pierwszym oświadczeniem napisanym w programie musi być

```
#PROGRAM    CALC77
```

Jeżeli program jest pisany na maszynę, która nie jest wieloprogramową, wówczas numer priorytetu może być opuszczony.

Przykłady

1. Napisz oświadczenie, które daje programowi nazwę SORT i numer priorytetu 42.

2. Które z następujących nie mogą być użyte jako nazwy programu?

READ	PIL
AX	SAK2
WRITE	Q

Odpowiedzi

1.

#PRØGRAM			SØRT42		

Jest to pierwsze zdanie /oświadczenie/ w programie. Nic go nie poprzedza.

2. Dwie nazwy są nieprawidłowe:

WRITE ma więcej, niż 4 litery.
SAK2 zawiera symbol 2, który nie jest dozwolony.

Dyrektywa PROGRAM musi pojawić się znowu bezpośrednio zanim zaczną się rozkazy programu. Tym razem nie ma żadnej potrzeby wymieniać nazwę programu i numer priorytetu. Funkcją dyrektywy PROGRAM jest tu po prostu wskazanie końca oświadczeń o danych i początku rozkazów programu.

#PRØGRAM					
	LDX	7	CØN1		
	ADS	7	CØN2		
	STØZ		5		
	SDX	5	DAYS		
	etc.				

Przykłady

1. Trzema pierwszymi rozkazami w programie zwanym FILE, z numerem priorytetu 60, są:

	LDX	4	REC		
	ANDX	4	MASK		
	STØ	4	WØRK		

Napisz te trzy rozkazy razem z poprzedzającym je oświadczeniem.

Odpowiedź

#PROGRAM					
	LDX	4	REC		
	ANDX	4	MASK		
	STØ	4	WØRK		

Nazwa programu i numer priorytetu występują tylko na początku w innej dyrektywie "PROGRAM".

Dyrektywy komentujące /Komentarze/

Gdy program jest napisany, musi on być przed oddaniem go do użytku sprawdzony /testowany/. Testowanie programu powoduje, co zdarza się dosyć często, wprowadzanie zmian.

Zmiany programu są znacznie łatwiejsze, jeżeli jest on gęsto opatrzony komentarzami, wyjaśniającymi cel różnych rozkazów i oświadczeń. Jest to szczególnie ważne, gdy program był sporządzany przez dłuższy okres i musi być zmieniony w związku ze zmianą warunków. Zupełnie możliwe w takim przypadku, że osoba, lub osoby dokonujące zmian nie brały żadnego udziału w początkowym opracowaniu.

Ciąg oświadczeń o danych i rozkazów programowych może być przerwany w dowolnym miejscu i po dyrektywie komentującej mogą być wprowadzone komentarze. Dyrektywa ta jest symbolizowana przez znak #. W polu operamda nie musi być pisane nic innego z wyjątkiem, oczywiście, komentarze.

```
        ADM   3  65
        STO   3  ANS
#               SEKCJA TA OBLICZA RABAT
        LDX   5  PRICE
        LDX   6  PRICE+1
```

Dyrektywa komentująca jest przykładem dyrektywy INTERRUPT /przerwania/. Dyrektywa przerwania ma wpływ tylko na jedno oświadczenie.

W przykładzie pokazanym powyżej, rozkazy nastąpiły za komentarzem, ale nie musiały one być wprowadzane przez inną dyrektywę programową, aby nie zostały potraktowane jako komentarz.

Dyrektywy LOWER i PROGRAM są przykładami dyrektyw MAJOR /główne/.

Dyrektywa główna ma wpływ na wszystkie oświadczenia na arkuszu programowym, znajdujące się między nią i następną dyrektywą główną, z wyjątkiem tych oświadczeń wprowadzonych na podstawie dyrektywy przerwania.

Gdy ma miejsce konwersja Programu Zródłowego na Program Wynikowy, bardzo ważnym produktem ubocznym jest Listowanie. Jest to wydruk, pokazujący oświadczenia w PLANie i ich maszynowe odpowiedniki.

Fragment listowania podany poniżej pokazuje dokładnie istotę przerywania dyrektywy komentującej. Następujące bezpośrednio rozkazy PLAN zostały wszystkie przetłumaczone na język w kodzie maszyny.

```
        LDCH   6   0 3                236   024 6 3       0
        SRC    6   3                  237   112 6 0       3
        SLC    67  3                  238   111 6 0       3
        BNZ    6   ++2                239   052 6        241
        BCHX   3   +-4                240   064 3        236
        SLL    67  21                 241   111 6 0   1045
        EXIT   1   0                  242   072 1          0
#               CHECKSUM ROUTINE
CSUM    BCHX   3   ++1                243   064 3        244
        LDCH   2   0 3                244   024 2 3        0
        BZE    2   LM1                245   050 2          0
        LDN    5   0 3                246   100 5 3        0
        SUM    4   0 2                247   127 4 2        0
        BZE    44  LM1                248   050 4          0
        LDCT   2   20                 249   124 2 0       20
        LDX    4    4H                250   000 4 0        0
        TXU    4   BUFFER 2           251   026 4 2        0
```

ROZKAZY PLANu

ROZKAZY W KODZIE MASZYNY

NUMERY ROZKAZOW PROGRAMU

Alternatywną metodą pisania komentarzy przy poszczególnych oświadczeniach i rozkazach jest pozostawienie po rozkazie kilku spacji i rozpoczęcie komentarze po nawiasie kwadratowym. Ze względów porządkowych, praktykuje się pisanie wszystkich komentarzy tego rodzaju od kolumny 36.

```
      MPY   5  CONVA                    [ZAMIEN FUNTY NA DOLARY
      STO   5  BANK
      MPY   3  CONVB                    [ZAMIEN DOLARY NA DRACHMY
```

Przykład

1. Napisz na arkuszu programowym rozkaz zapamiętania zawartości akumulatora 4 w komórce TOTAL, następnie napisz komentarz "ta sekcja oblicza podatek" i w końcu napisz rozkaz przeniesienia zawartości komórki GROSS do akumulatora 2.

2. Napisz komentarz "oblicz różnicę" obok rozkazu odjęcia zawartości NUMA z akumulatora 7.

Odpowiedzi

```
      STO   4  TOTAL
#             SEKCJA TA OBLICZA PODATEK
      LDX   2  GROSS
```

```
      SBX   7  NUMA                     [OBLICZ RÓŻNICE
```

ETYKIETA POCZĄTKOWA /16 słów/

1	2	3 4	5	6	7	8	9	10	11-16
HDDR	Numer serynny taśmy	Nazwa zbioru		Numer sekwencyjny krążka	Numer generacji	Okres przechowywania	Data zapisania	Informacje dla inżyniera	——
4 znaki	binarny	12 znaków		binarny	binarny	binarny	binarna		słowa wolne

ZNACZNIK PUNKTU ZAŁADOWANIA /znacznik odblaskowy/

Opis:

Identyfikacja etykiety początkowej	Numer sekwencyjny taśmy w ramach instalacji. Do końca używania taśmy	Identyfikacja danych na taśmie		Plus jeden za każdym zapisywania zbioru	Ilość dni po dacie zapisywania przez które taśma musi być przechowywana	Data zapisania taśmy /ilość dni od 1.1 1900/	Używane przez DYRYGENTA dla zanotowania ilości razy zapisu taśmy i użycia jednost. taśm.
		Nazwa zbioru powinna być unikalna dla instalacji	1-szy krążek = 0, do maksymalnie 511				

OBSZARY DEFINICJI ZBIORU /20 słów/

Otwórz zbiór wejściowy z daną etykietą

Słowo indeksowe	Słowo odpowiedzi						
261/ 400	Ustawione na zero	Nazwa zbioru	Ustawione na zero	Ustawione na zero	Ustawione na zero	Ustawione na zero	10-20 ustawione na zero

Opis działania DYRYGENTA

Używane przez DYRYGENTA dla ustalenia rodzaju i trybu		Informacje dostarczone przez użytkownika w obszarze definicji zbioru porównywane są przez DYRYGENTA z etykietą początkową		DYRYGENT przenosi informacje z etykiety początkowej do obszaru definicji zbioru	
			Powiększane przez MTM przed otwarciem krążków kontynuacji	Sprawdzane przez MTM z nr generacji dostarczonym przez program	Używane przez system MTM

Działanie dodatkowe: DYRYGENT bada nieobecność krążka zapisu

Otwórz zbiór wyjściowy i zapisz daną etykietę

Słowo indeksowe	Słowo odpowiedzi						
261/ /400	Ustawione na zero	Nazwa zbioru	Ustawione na zero	/patrz niżej/	Okres przechowywania	Ustawione na zero	10-20 ustawione na zero

Opis działania DYRYGENTA

Używane przez DYRYGENTA dla ustalenia rodzaju i trybu		DYRYGENT zapisuje informacje dostarczone w obszarze definicji zbioru do etykiety początkowej/gdy znaleziona jest odpowiednia taśma/				
			Powiększane przez MTM przed otwarciem krążków kontynuacji	Ustawiane przez program przed otwarciem zbioru	Data skasowania na etykiecie początkowej sprawdzane przez DYRYGENTA dla znalezienia taśmy do zapisu	Używane przez system MTM

Działanie dodatkowe: DYRYGENT sprawdza obecność krążka zapisu

ETYKIETA KOŃCOWA /20 słów/

Identyfikacja etykiety końcowej	Licznik bloków danych	Zero	Zero	Słowa od 5 do 20 zarezerwowane dla użytku sowftwaru

Przerwy międzyblokowe

Wskazuje czy koniec krążka czy koniec zbioru	Wskazuje Ilość bloków zapisanych na taśmie

ZNACZNIK TAŚMY /blok 1-znakowy/

ZNACZNIK KOŃCA TAŚMY /znacznik odblaskowy tylko w końcu krążka/

System manipulowania taśmą magnetyczną zapisze znacznik taśmy i
a/ automatycznie etykietę końcową krążka, gdy wykryty jest znacznik końca taśmy
b/ etykietę końcową zbioru, gdy takie jest polecenie instrukcji MTEND

MAKROROZKAZY SYSTEMU MANIPULOWANIA TASMĄ MAGNETYCZNĄ

NA POZIOMIE FIZYCZNYM

WEJŚCIE

Pole operacji	Pole akumulatora	Pole operanda 1	2	3	4
MTDEF	Numer jednostki taśmy magnetycznej programu	ustawione na zero	ustawione na zero	Adres początkowy obszaru definicji zbioru	Etykieta pierwszego rozkazu podprogramu wyjątkowego
MTRDB	Numer jednostki taśmy magnetycznej programu	Adres początkowy obszaru wejścia /może być modyfikowane/	lub a/Akumulator Zawierający b/Adres symboliczny max. długość bloku lub c/max. długość bloku jako literał		

WYJŚCIE

MTDEF	Numer jednostki taśmy magnetycznej programu	ustawione na zero	ustawione na zero	Adres początkowy obszaru definicji zbioru	Opcjonalne - w tym tekście nieużywane
MTWRB	Numer jednostki taśmy magnetycznej programu	Adres początkowy obszaru wyjścia /może być modyfikowany/	jak dla MTRDB		

NA POZIOMIE LOGICZNYM

WEJŚCIE

MTDEF	Numer jednostki taśmy magnetycznej programu	1 = bufor pojedynczy 2 = bufory alternatywne	Maksymalna długość buforu	Adres początkowy obszaru definicji zbioru	Etykieta pierwszego rozkazu podprogramu wyjątkowego
MTRD	Numer jednostki taśmy magnetycznej programu	A Adres początkowy obszaru rekordu	← może być modyfikowane		
MTRDP	Numer jednostki taśmy magnetycznej programu	B Akumulator, w którym adres początkowy następnego rekordu ma być umieszczony			

WYJŚCIE

MTDEF	Numer jednostki taśmy magnetycznej programu	1 = bufor pojedynczy 2 = bufor alternatywny	Maksymalna długość buforu	Adres początkowy obszaru definicji zbioru	Opcjonalne - w tym tekście nieużywane
MTWR	Numer jednostki taśmy magnetycznej programu	A Adres początkowy rekordu do wyprowadzenia lub B Akumulator, w którym przechowywany adres początkowy rekordu.			

CENTRALNY OŚRODEK DOSKONALENIA KADR KIEROWNICZYCH

w y d a j e:

- Miesięcznik „Doskonalenie Kadr Kierowniczych"
- Bibliografię bieżącą „Organizacja, Zarządzanie i Doskonalenie Kadr Kierowniczych"
- Zestawienia dokumentacyjne książek i czasopism
- Materiały i studia
- Materiały szkoleniowe

Wydawnictwa CODKK są do nabycia lub zaprenumerowania w Dziale Administracyjno-Ekonomicznym, Warszawa, ul. Wawelska 56, tel. 25-29-59