Część druga

PROGRAMOWANIE

3. ZASADY PROGRAMOWANIA I KODOWANIA

[3.0. Uwagi wstępne, 3.1. Metoda adresów względnych, 3.2. Działania arytmetyczne na rozkazach, 3.3. Schematy blokowe, 3.4. Programy liniowe, 3.5. Programy z rozwidleniami, 3.6. Programy cykliczne i iteracyjne, 3.7. Sterowanie cyklami]

3.0. UWAGI WSTĘPNE

Zanim przejdziemy do omawiania zasad programowania i kodowania, zastanówmy się nad definicją programowania i nad definicją kodowania. Przez programowanie będziemy rozumieli budowanie algorytmów postępowania maszyny dla rozwiązania określonego zadania. Warunkami, jakie muszą spełniać te algorytmy, zajmiemy się w rozdz. 4, obecnie ograniczymy się do omówienia zasadniczych elementów tych algorytmów oraz ich kodowania. Ostatnie pojęcie musimy wyjaśnić bliżej. Przez kodowanie będziemy rozumieli zapisywanie algorytmów w języku zrozumiałym dla maszyny — w tak zwanym kodzie zewnętrznym maszyny. Program wprowadzający, który omówimy w punkcie 5.1.1, tłumaczy kod zewnętrzny na kod wewnętrzny maszyny. W UPMC omówionej przykładowo kod wewnętrzny maszyny jest kodem binarnym (dwójkowym), kod zewnętrzny zaś jest kodem ósemkowym, wobec czego przejście od kodu zewnętrznego do wewnętrznego jest bardzo proste. Przyjęte przez nas rozwiązanie stwarza jedną zasadniczą trudność, mianowicie wymaga od programującego dużego nakładu pracy; a co za tym idzie czas potrzebny na zaprogramowanie i zakodowanie problemu, który chcemy rozwiązać na maszynie, jest stosunkowo długi. Zaletą przyjętego przez nas rozwiązania jest prostota programu wprowadzającego (program wprowadzający składa się z 23 rozkazów), co przy małej pojemności pamięci jest ważne, ponadto takie rozwiązanie pozwala na stosunkowo szybkie wprowadzanie programu do maszyny, co również jest bardzo korzystne przy małej prędkości liczenia.

Z kodem maszyny typowej zapoznaliśmy się w rozdz. 2; dopóki nie znamy jednak jeszcze na pamięć tego kodu, sporządzimy tzw. tablicę kodową, która ułatwi nam szybkie przechodzenie od zapisu symbolicznego do kodu (tabl. 3-1).

Dotychczas nie zastanawialiśmy się nad metodą kodowania liczb, obecnie omówimy jedną ze stosowanych przez nas metod kodowania liczb, umożliwiającą wprowadzenie liczb razem z rozkazami. Metoda ta pozwoli nam formalnie na nierozróżnianie przy

Tablica kodowa operacji

	0	1	2	3	4	5	6	7
0	$(n)\rightarrow D$	$(n)\rightarrow$L	$n\rightarrow L$	$n\rightarrow L$ $W=1$	$(L)\rightarrow n$ $n+1\rightarrow L$	stop n	$n\rightarrow L$ $W=0$	niewykorzystany
	—	—	—	—	—	—	—	—
	—	—	—	—	—	—	—	—
1	$(A)+(n)\rightarrow A$	$(A)-(n)\rightarrow A$	$(n)\rightarrow A$	$-(n)\rightarrow A$	$(A)\rightarrow n$	$z(A)\rightarrow A$	$(M)\cdot(n)\rightarrow A$	$(A):(n)\rightarrow A_L$ reszta$\rightarrow A_p$
	$(A)<0$	$(A)<0$	$(A)\neq 0$	$(A)\neq 0$	$(A)<0$	$(A)<0$	$(A)<0$	$(A)<0$
	$(A)\geqslant 1$ albo $(A)<-1$	$(A)\geqslant 1$ albo $(A)<-1$	—	$(A)=1$	—	$(A)\geqslant 1$ albo $(A)<-1$	$(A)=1$	$\|(A)\|\geqslant\|(n)\|$
2	$(A)\cdot 2^n\rightarrow A$	$(A):2^n\rightarrow A$	$(A):2_n\rightarrow A$	$(A)\cdot 2_n\rightarrow A$	czytaj	drukuj	$\|(A)\|\rightarrow A$	$(n)\rightarrow M$
	$(A)<0$	$(A)<0$	$(A)<0$	$(A)<0$	$Y=1$	$Z=1$	$A\neq 0$	$(M)<0$
	$(A)\geqslant 1$ albo $(A)<-1$	—	—	—	—	—	$(A)=1$	—
3	$(A)\cap(n)\rightarrow A$	niewykorzystany	niewykorzystany	niewykorzystany	niewykorzystany	niewykorzystany	niewykorzystany	$(L)\rightarrow n$ $n+1\rightarrow L$ $N=1$
	$A\neq 0$							—
	—							—

wprowadzaniu liczb i rozkazów. Dowolną liczbę co do modułu mniejszą od jedności o długości 33 B w rozwinięciu binarnym (33 B odpowiada dokładności około 10 cyfr dziesiętnych) będziemy przedstawiali za pomocą pary rozkazów, które dla odróżnienia będziemy nazywali *pseudorozkazami*. Liczbę zapisaną za pomocą pary pseudorozkazów będziemy nazywali *liczbą przedstawioną w kodzie rozkazowym*. Zastanówmy się najpierw nad algorytmem przejścia z systemu dziesiętnego do systemu o zmiennej podstawie, jakim jest kod rozkazowy.

Niech będzie dana liczba rzeczywista x spełniająca warunek $2 > x \geqslant 0$ i niech będzie dany skończony ciąg liczb naturalnych większych od jedności $g_1, g_2, \ldots, g_n$. Liczbę x możemy napisać wówczas jednoznacznie w postaci

$$x = C_0 + \frac{C_1}{g_1} + \frac{C_2}{g_1 g_2} + \frac{C_3}{g_1 g_2 g_3} + \ldots + \frac{C_n}{g_1 g_2 \ldots g_n} + \frac{x_{n+1}}{g_1 g_2 \ldots g_n}, \tag{3-1}$$

gdzie

$$\left.\begin{array}{ll} C_0 = [x], & x_1 = x - C_0, \\ C_1 = [g_1 x_1], & x_2 = g_1 x_1 - C_1, \\ C_2 = [g_2 x_2], & x_3 = g_2 x_2 - C_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \\ C_n = [g_n x_n] & x_{n+1} = g_n x_n - C_n. \end{array}\right\} \tag{3-2}$$

W interesującym nas przypadku $n = 12$, a wyrazy ciągu g_i przyjmują wartości:

$$g_1 = 2, \quad g_2 = g_3 = \ldots = g_6 = 8, \quad g_7 = 4, \quad g_8 = g_9 = \ldots = g_{12} = 8. \tag{3-3}$$

W ten sposób podzieliliśmy 34-bitowe słowa długie maszyny na 13 grup:
1B + 1B + 3B + 3B + 3B + 3B + 3B + 2B + 3B + 3B + 3B + 3B + 3B = 34B.
Jeśli teraz dla naszej liczby x, korzystając z algorytmu (3-2) z wartościami na g_1 podanymi w (3-3), obliczamy $C_0, C_1 \ldots, C_{12}$ (zaokrąglając przy tym C_{12}), to otrzymamy kolejne cyfry przybliżonego rozwinięcia liczby x w kodzie rozkazowym. Przy przeliczeniu liczby z systemu dziesiątkowego do systemu kodu rozkazowego wygodnie jest korzystać z następującego schematu zapisu:

$$\left.\begin{array}{cccccccccccc} C_0 & C_1 & C_2 & C_3 & C_4 & C_5 & C_6 & C_7 & C_8 & C_9 & C_{10} & C_{11} & C_{12} \\ 2 & 8 & 8 & 8 & 8 & 8 & 8 & 4 & 8 & 8 & 8 & 8 & 8 \end{array}\right\} \tag{3-4}$$

Obliczając pomocnicze wyrażenie

$$C_1' = 2C_0 + C_1,$$

możemy napisać liczbę x za pomocą pary pseudorozkazów, z których pierwszy umieścimy pod adresem parzystym, drugi zaś pod adresem nieparzystym:

$$\left.\begin{array}{cccccc} C_7 & C_8 & C_9 & C_{10} & C_{11} & C_{12} \\ C_1' & C_2 & C_3 & C_4 & C_5 & C_6 \end{array}\right\} \tag{3-5}$$

Przedział liczb x $(2 > x \geqslant 0)$ rozdzielimy na dwa podprzedziały: $1 > x \geqslant 0$ oraz $2 > x \geqslant 1$. Jeśli liczba y mniejsza co do modułu od jedności jest liczbą dodatnią, to przeliczamy z systemu dziesiętnego do systemu kodu rozkazowego liczbę $x = y$, $1 > x \geqslant 0$, jeśli zaś y jest liczbą ujemną, to przeliczamy liczbę $x = 2 + y$ $(2 > x \geqslant 1)$.

Przeliczając liczby x z podprzedziału $< 2{,}1 >$ musimy pamiętać, że dla dokonania zaokrąglenia odejmiemy 4 od C_{13} (przy założeniu, że $g_{13} = 8$), nie zaś jak w przypadku podprzedziału $< 1{,}0 >$, gdzie dodajemy 4 do C_{13}.

3.1. METODA ADRESÓW WZGLĘDNYCH

Zdajemy sobie sprawę z następującej trudności kodowania: wpisując kolejne rozkazy musimy znać adresy stałe i adresy robocze, na to zaś aby znać te adresy, musimy mieć wypisane wszystkie sekwencje rozkazów programu.

Trudność tę nieraz możemy ominąć stosując tzw. adresy symboliczne, mianowicie oznaczając nieznany adres liczby a przez $< a >$. Metoda ta nie jest jednak najwygodniejsza, gdyż

1) przy dużej ilości danych jest bardzo nieprzejrzysta,

2) dla rozróżnienia słów długich i krótkich musimy używać dodatkowych symboli.

Trudność ta usuwa stosowanie tzw. adresów względnych. Adresy rozkazów wykonywanych kolejno zapisujemy jako $k + 0000$, $k + 0001$, $k + 0002$ itd. Analogicznie postępujemy z materiałem liczbowym i adresami roboczymi. Na przykład liczby umieszczamy pod kolejnymi adresami $a + 4002$, $a + 4004$ itd., adresy robocze oznaczamy $b + 0000$, $b + 0001$, $b + 0002$, itd.

Uwaga: Liczby $k, a, b, \ldots$ będziemy nazywali stałymi przeadresowania.

Po ułożeniu programu w adresach względnych przystępujemy do rozmieszczania go w pamięci wewnętrznej i zewnętrznej maszyny (w przypadku omawianej UPMC taśmę perforowaną możemy traktować jako pamięć zewnętrzną).

Rozpatrzmy kolejność rozmieszczenia programu w przypadku, gdy cały program mieści się w pamięci wewnętrznej. Najpierw rozmieszczamy program, następnie dane i wreszcie adresy robocze. Doświadczenie uczy, że znacznie wygodniej adresy robocze rozmieszczać niezależnie w pewnych ustalonych częściach pamięci. W maszynie naszej przyjęliśmy ścieżkę nr 1 (adresy 0100÷0177) za miejsce rozmieszczenia adresów roboczych(1).

W przypadku gdy liczba potrzebnych nam adresów roboczych przekracza ilość adresów na ścieżce nr 1, używamy jeszcze ścieżki nr 2 itd.

Do rozmieszczania programu w pamięci maszyny używamy specjalnych formularzy podzielonych na 64 części; każdemu formularzowi odpowiada jedna ścieżka na bębnie magnetycznym.

Po rozmieszczeniu rozkazów i materiału liczbowego przyjmujemy jednolity system adresów roboczych dla rozkazów i danych liczbowych.

Rozmieszczając program należy unikać pełnego wykorzystania pamięci, zostawiając od kilku do kilkunastu adresów wolnych, ponieważ niejednokrotnie po kontroli programu w maszynie należy poprawić program. Jeśli w wyniku kontroli okaże się, że

(1) Przy rozmieszczaniu adresów roboczych należy sprawdzić, czy nie ma kolizji z adresami roboczymi podprogramów używanych dla rozwiązania danego zagadnienia.

między rozkazami k i $k+1$ należy umieścić dodatkową grupę m rozkazów, gdzie $m+2 \leqslant n$, zaś n oznacza ilość adresów niewykorzystanych, to pod adresem $k+1$ umieszczamy rozkaz 02 $s+0000$, ($s+0000$ — pierwszy adres rezerwowy), pod adresem $s+0000$ umieszczamy rozkaz stojący poprzednio pod adresem $k+1$, pod adresami $s+0001 \div s+m$ umieszczamy dodatkowe m rozkazów, zaś pod adresem $s+m+0001$ umieszczamy 02 $k+2$.

Rozmieszczenie programu w pamięci rozpatrzymy na prostym przykładzie.

Przykład 3-1. Obliczyć wartość funkcji przy założeniu, że wszystkie parametry, wielkości pośrednie i wyniki są co do modułu mniejsze od jedności,

$$y = \frac{c_0 x - c_1 x^3 + c_2 x^5}{c_0 - c_3 x^2 + c_4 x^4 + c_5 x^6} \tag{3-6}$$

dla $x = x_1$.

Przekształcając otrzymamy postać dogodniejszą do prowadzenia obliczeń

$$y = \frac{[(c_2 x^2 - c_1)x^2 + c_0]x}{[(c_5 x^2 + c_4)x^2 - c_3]x^2 + c_0}. \tag{3-7}$$

Stałe rozmieścimy pod adresami długimi w kolejności podanej w tabl. 3-2.

Tablica 3-2

Adresy	$a+4000$	$a+4002$	$a+4004$	$a+4006$	$a+4010$	$a+4012$	$a+4014$
Zawartość adresów	c_0	c_1	c_2	c_3	c_4	c_5	x_1

Adresy robocze będziemy numerowali począwszy od $b+4000$,...

Pierwszy rozkaz: *Przesyłamy do rejestru M wielkość* x_1

$$k+0000 \quad 27 \quad a+4014.$$

Drugi rozkaz: *Tworzymy kwadrat* x_1

$$k+0001 \quad 16 \quad a+4014.$$

Po wykonaniu tego rozkazu mamy w akumulatorze dokładny iloczyn (długi).

Trzeci rozkaz: *Zaokrąglamy liczbę zawartą w akumulatorze*

$$k+0002 \quad 15 \quad 0000.$$

Zaokrąglenie to zmniejszy nam błąd maksymalny przy obliczeniu wyrażenia (3-7), dalsze zaokrąglenia możemy pominąć ze względu na naprzemienne znaki wyrażenia (3-7).

Czwarty rozkaz: *Przesyłamy przybliżoną wartość* x_1^2 *pod pierwszy (długi) adres roboczy* $b+4000$

$$k+0003 \quad 14 \quad b+4000.$$

Piąty rozkaz: *Przesyłamy przybliżoną wartość* x_1^2 *do rejestru M*

$k + 0004 \quad 27 \quad b + 4000.$

Szósty rozkaz: *Obliczamy* c_5x^2 *dla* $x = x_1$

$k + 0005 \quad 16 \quad a + 4012.$

Siódmy rozkaz: *Obliczamy* $c_5x^2 + c_4$ *dla* $x = x_1$

$k + 0006 \quad 10 \quad a + 4010.$

Ósmy rozkaz: *Umieszczamy wynik pod drugim (długim) adresem roboczym* $b + 4002$

$k + 0007 \quad 14 \quad b + 4002.$

Dziewiąty rozkaz: *Obliczamy* $(c_5x^2 + c_4)x^2$

$k + 0010 \quad 16 \quad b + 4002.$

Dziesiąty rozkaz: *Obliczamy* $(c_5x^2 + c_4)x^2 - c_3$

$k + 0011 \quad 11 \quad a + 4006.$

Jedenasty rozkaz: *Umieścimy wynik pod adresem roboczym* $b + 4002$

$k + 0012 \quad 14 \quad b + 4002.$

Dwunasty rozkaz: *Obliczamy* $[(c_5x^2 + c_4)x^2 - c_3]x^2$

$k + 0013 \quad 16 \quad b + 4002.$

Trzynasty rozkaz: *Obliczamy* $[(c_5x^2 + c_4)x^2 - c_3]x^2 + c_0$

$k + 0014 \quad 10 \quad a + 4000.$

Czternasty rozkaz: *Umieścimy wynik pod adresem roboczym* $b + 4004$ (zapisujemy pod adresem $[(c_5x^2 + c_4)x^2 - c_3]x^2c_0$ dla $x = x_1$)

$k + 0015 \quad 14 \quad b + 4004.$

Piętnasty rozkaz: *Obliczamy* c_2x^2 *dla* $x = x_1$

$k + 0016 \quad 16 \quad a + 4004.$

Szesnasty rozkaz: *Obliczamy* $c_2x^2 - c_1$

$k + 0017 \quad 11 \quad a + 4002.$

Siedemnasty rozkaz: *Umieszczamy wynik pod adresem roboczym* $b + 4002$

$k + 0020 \quad 14 \quad b + 4002.$

Osiemnasty rozkaz: *Obliczamy* $(c_2x^2 - c_1)x^2$

$k + 0021 \quad 16 \quad b + 4002.$

Dziewiętnasty rozkaz: *Obliczamy* $(c_2x^2 - c_1)x^2 + c_0$

$$k + 0022 \quad 10 \quad a + 4000.$$

Dwudziesty rozkaz: *Umieszczamy wynik pod adresem roboczym* $b + 4002$

$$k + 0023 \quad 14 \quad b + 4002.$$

Dwudziesty pierwszy rozkaz: *Przesyłamy do rejestru M* x_1

$$k + 0024 \quad 27 \quad a + 4014.$$

Dwudziesty drugi rozkaz: *Obliczamy licznik wyrażenia* (3-7)

$$k + 0025 \quad 16 \quad b + 4002.$$

Dwudziesty trzeci rozkaz: *Obliczamy wartość y dla* $x = x_1$

$$k+0026 \quad 17 \quad b+4004\,.$$

Dwudziesty czwarty rozkaz: *Zatrzymujemy maszynę* (*stop*)

$$k+0027 \quad 05 \quad 0000$$

Zakładając, że k jest parzyste, z łatwością otrzymamy związek

$$a = 30+k, \tag{3-8}$$

$$\left.\begin{matrix} k+0030 \\ k+0031 \end{matrix}\right\} \quad k+4030 = \langle c_0 \rangle\,,$$

$$\left.\begin{matrix} k+0032 \\ k+0033 \end{matrix}\right\} \quad k+4032 = \langle c_1 \rangle\,,$$

$$\left.\begin{matrix} k+0034 \\ k+0035 \end{matrix}\right\} \quad k+4034 = \langle c_2 \rangle\,,$$

$$\left.\begin{matrix} k+0036 \\ k+0037 \end{matrix}\right\} \quad k+4036 = \langle c_3 \rangle\,,$$

$$\left.\begin{matrix} k+0040 \\ k+0041 \end{matrix}\right\} \quad k+4040 = \langle c_4 \rangle\,,$$

$$\left.\begin{matrix} k+0042 \\ k+0043 \end{matrix}\right\} \quad k+4042 = \langle c_5 \rangle\,,$$

$$\left.\begin{matrix} k+0044 \\ k+0045 \end{matrix}\right\} \quad k+4044 = \langle x_1 \rangle\,.$$

Adresy robocze rozmieścimy na ścieżce nr 1 począwszy od pierwszej komórki tej ścieżki, czyli

$$b = 100\,.$$

Pozostaje nam jeszcze przeanalizować wykorzystanie początkowych komórek roboczych. W tym celu sporządzamy harmonogram (tabl. 3-3).

IBJ

Dn.. 29.01.59

ARKUSZ PROGRAMOWY EM

kurs programowania

str. ◯

Program dla obliczania

$$y = \frac{c_2 x^2 - c_1 x^2 + c_0 x}{c_3 x^3 + c_2 x^2 + c_1 x^2 + c_0},$$

dla $x = x_1$, $\langle x_1 \rangle = k + 4044$,

Uwaga: k — parzyste; adresy robocze 4100 ÷ 4104.

k+000 0	27	4044	k	k+004 0			c_3	
1	16	4044	k	1				
2	15	0000		2			c_0	
3	14	4100		3				
4	27	4100		4			x_1	
5	16	4042	k	5				
6	10	4040	k	6				
7	14	4102		7				
k+001 0	16	4102		0				
1	11	4036	k	1				
2	14	4102		2				
3	16	4102		3				
4	10	4030	k	4				
5	14	4100		5				
6	16	4034	k	6				
7	11	4032	k	7				
k+002 0	14	4102		0				
1	16	4102		1				
2	10	4030	k	2				
3	14	4102		3				
4	27	4044	k	4				
5	16	4102		5				
6	17	4100		6				
7	05	0000		7				
k+003 0		c_0		0				
1				1				
2		c_1		2				
3				3				
4		c_2		4				
5				5				
6		c_3		6				
7				7				

Rys. 3-1. Arkusz programowy EM

Tablica 3-3

Harmonogram wykorzystania komórek roboczych

Kolejny rozkaz programu	4100	4102	4104
k+0000			
1			
2			
3	× × ×		
4	× × ×		
5			
6			
7		× × ×	
k+0010		× × ×	
1			
2		× × ×	
3		× × ×	

Kolejny rozkaz programu	4100	4102	4104
k+0014			
5			× × ×
6			× × ×
7			× × ×
k+0020		× × ×	× × ×
1		× × ×	× × ×
2			× × ×
3		× × ×	× × ×
4		× × ×	× × ×
5		× × ×	× × ×
6			
7			

Uwaga: Zakreślone zostały pola w naszym harmonogramie odpowiadające okresowi czasu, w którym korzystamy z danej komórki roboczej.

Na podstawie sporządzonego harmonogramu widzimy, że możemy ograniczyć ilość komórek roboczych do dwóch. Mianowicie zamiast używać komórek 4100 i 4104, możemy korzystać tylko z komórki 4100. Zastępujemy więc we wszystkich rozkazach programu, w których występuje adres 4104, adresem 4100. Otrzymany w ten sposób program możemy jednolicie adresować używając adresów względnych ze stałą przeadresowania k (przy założeniu, że k jest — parzyste). Otrzymujemy w ten sposób:

k+0000	27	k+4044
1	16	k+4044
2	15	0000
3	14	4100
4	27	4100
5	16	k+4042
6	10	k+4040
7	14	4102
k+0010	16	4102
1	11	k+4036
2	14	4102
3	16	4102
4	10	k+4030
5	14	4100
6	16	k+4034
7	11	k+4032

$k+0020$	14	4102
1	16	4102
2	10	$k+4030$
3	14	4102
4	27	$k+4044$
5	16	4102
6	17	4100
7	05	0000

Obecnie możemy zapisać nasz program na arkuszu programowym. W paru słowach postaramy się omówić stosowany przez nas arkusz programowy. Dalej będziemy nazywali ten arkusz — arkuszem programowym EM (E oznacza, że jest to arkusz przystosowany do kodu ósemkowego, *M* — oznacza, że arkusz ten jest przystosowany do zapisania rozkazów w adresach względnych). Na stosowanym arkuszu programowym liczby są zapisywane tak, jak to wspominaliśmy w punkcie 3.0, tj. w postaci dwóch pseudorozkazów. Ze względu na stosowany przez nas program wprowadzający na arkuszu programowym EM stałą przeadresowania w części adresowej rozkazu piszemy w odwrotnej kolejności niż dotychczas, mianowicie najpierw piszemy część liczbową adresu, a następnie stałą przeadresowania, znak dodawania opuszczamy. Prawidłowo wypełniony arkusz programowy EM jest przedstawiony na rys. 3-1. Jako przykładu do wypełnienia arkusza używaliśmy zakodowanego programu przedstawionego w niniejszym punkcie. Przyjęty przez nas program wprowadzający (punkt 5.1) pozwala na jednoczesne korzystanie z dwunastu stałych przeadresowania, oznaczonych literami *X, N, H, L, M, S, G, B, D, F, J, K.*

3.2. DZIAŁANIA ARYTMETYCZNE NA ROZKAZACH

Niejednokrotnie w czasie obliczeń zachodzi konieczność zmieniania wartości rozkazu (jego części adresowej lub też jego części operacyjnej). Oznaczając przez r część operacyjną rozkazu, a przez n część adresową rozkazu, liczbę odpowiadającą rozkazowi możemy zapisać w postaci

$$2^{-4}r+2^{-16}n; \tag{3-9}$$

przy czym r jest liczbą pięciobitową, n zaś jest liczbą dwunastobitową.

Dla powiększenia części adresowej rozkazu o 1 należy do liczby postaci (3-9) dodać 2^{-16}, dla powiększenia zaś części adresowej o 2, należy dodać do liczby postaci (3-9) 2^{-15}. Podobnie dla zmniejszenia części adresowej rozkazu o 1 należy dodać do liczby postaci (3-9) $2-2^{16}$, zaś dla zmniejszenia części adresowej rozkazu o 2 należy dodać do liczby postaci (3-9) $2-2^{-15}$.

Rozkazy o kodach części operacyjnej $00\div17$ są traktowane jako liczby dodatnie, rozkazy zaś o kodach części operacyjnej $20\div37$ są traktowane jako liczby ujemne. Ze względu na omawiane w punkcie 2.1 własności dodawania w arytmetyce uzupełnienio-

wej, nie zachodzi potrzeba rozróżniania przy wykonywaniu działań arytmetycznych na rozkazach znaku liczb odpowiadających rozkazom.

Przypuśćmy, że w trakcie pewnych obliczeń musimy zmienić część operacyjną rozkazu, np. rozkaz dodawania $10\,n$, musimy zastąpić rozkazem odejmowania $11\,n$; w tym celu dodajemy do liczby odpowiadającej rozkazowi $10\,n$ 2^{-4}.

3.3. SCHEMATY BLOKOWE

Celem schematów blokowych jest przedstawienie zasadniczych czynności programu przy użyciu języka graficznego. W dalszym ciągu zajmiemy się omówieniem zasad tworzenia tych schematów. Warto podkreślić, że schematy te mają dwojaką przydatność:

1) ułatwiają kodowanie problemu,

2) umożliwiają łatwe zorientowanie się w organizacji programu.

Jak wiadomo, program zakodowany jest tak mało czytelny, że nawet osoba, która układała ten program po upływie pewnego czasu zapomina wiele szczegółów i dla ponownego zorientowania się w programie musi poświęcić wiele czasu.

Podziału programu na pewne prostsze części możemy dokonać na wielu drogach; chodzi jednak o to, aby podział ten spełniał pewne warunki. Warunki te są następujące:

1. Elementy podziału muszą być niezależnie kodowalne.

2. Podział musi uwzględniać specyfikę obliczeń automatycznych, tzn. musi odróżniać kroki algorytmów numerycznych od pomocniczych operacji, służących np. do przekształcenia tych algorytmów.

3. Podział musi uwzględniać wszystkie dopuszczalne w programie kolejności wykonywania sekwencji rozkazów lub zespołów tych sekwencji oraz kryteria wyboru tych kolejności.

W dalszym ciągu będziemy rozróżniali trzy rodzaje czynności występujących w programie:

1) czynności arytmetyczne — obliczenia arytmetyczne wykonywane na liczbach lub rozkazach,

2) czynności logiczne — sprawdzanie warunków, kryteria wyboru kolejności itp.,

3) czynności organizacyjne — przesyłanie liczb i rozkazów, wprowadzanie i wyprowadzanie z maszyny itp.

Rozwiązanie zagadnienia, dla którego chcemy zbudować schemat blokowy, składa się z czynności $a_1, a_2, \ldots, a_m$. Każdą z tych czynności możemy zakwalifikować jako czynność arytmetyczną, logiczną albo organizacyjną. Ze względu na podobną strukturę czynności arytmetyczne i organizacyjne obejmiemy wspólną nazwą operatorów. Natomiast czynności logiczne nazwiemy predykatami. Operatory i predykaty są reprezentowane w programie jako pewne zespoły rozkazów. Na to aby operatory i predykaty spełniały podane trzy warunki, powinny być spełnione następujące wymagania:

a. Warunek uporządkowania operatora. Sterowanie z zewnątrz może być przekazane tylko pierwszemu rozkazowi operatora, przekazanie sterowania na zewnątrz (koniec realizowania operatora) może znajdować się tylko na końcu operatora.