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.

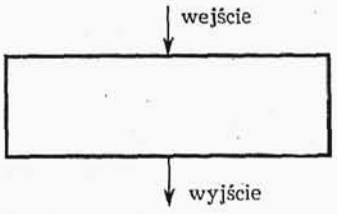
b. Warunek związania operatora. Jeżeli był wykonany pierwszy rozkaz operatora, to muszą być wykonane wszystkie pozostałe.

c. Warunek prostoty operatora. Operator realizuje tylko jeden rodzaj czynności w programie.

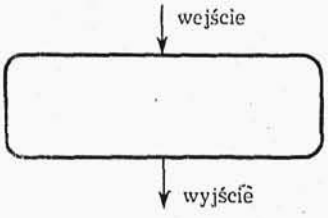
d. Warunek uporządkowania predykatu. Sterowanie z zewnątrz może być przekazane tylko pierwszemu rozkazowi predykatu, warunkowych rozkazów przekazanie sterowania na zewnątrz predykat musi mieć przynajmniej jeden.

Definicja 1. Będziemy mówili krótko — wejście operatora (predykatu), zamiast mówić pierwszy rozkaz operatora (predykatu).

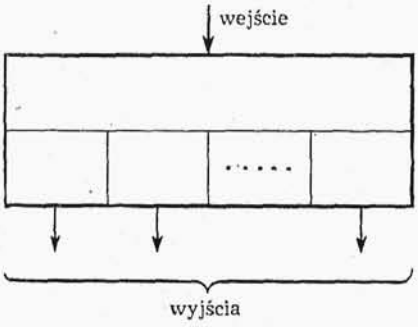
Definicja 2. Będziemy mówili krótko — wyjście operatora (predykatu), zamiast mówić rozkaz przekazania sterowania z operatora (predykatu) na zewnątrz.



Rys. 3-2. Graficzne przedstawienie operatora



Rys. 3-3. Graficzne przedstawienie operatora przeadresowywanego



Rys. 3-4. Graficzne przedstawienie predykatu

Graficznie operatory będziemy przedstawiali za pomocą prostokątów, wejście do operatora będziemy oznaczali za pomocą strzałki skierowanej do wnętrza prostokąta (z reguły będziemy rysowali ją nad górnym bokiem prostokąta), wyjście z operatora będziemy oznaczali za pomocą strzałki skierowanej na zewnątrz prostokąta (z reguły będziemy rysowali ją pod dolnym bokiem prostokąta). Schematycznie przedstawiliśmy ten zapis na rys. 3-2.

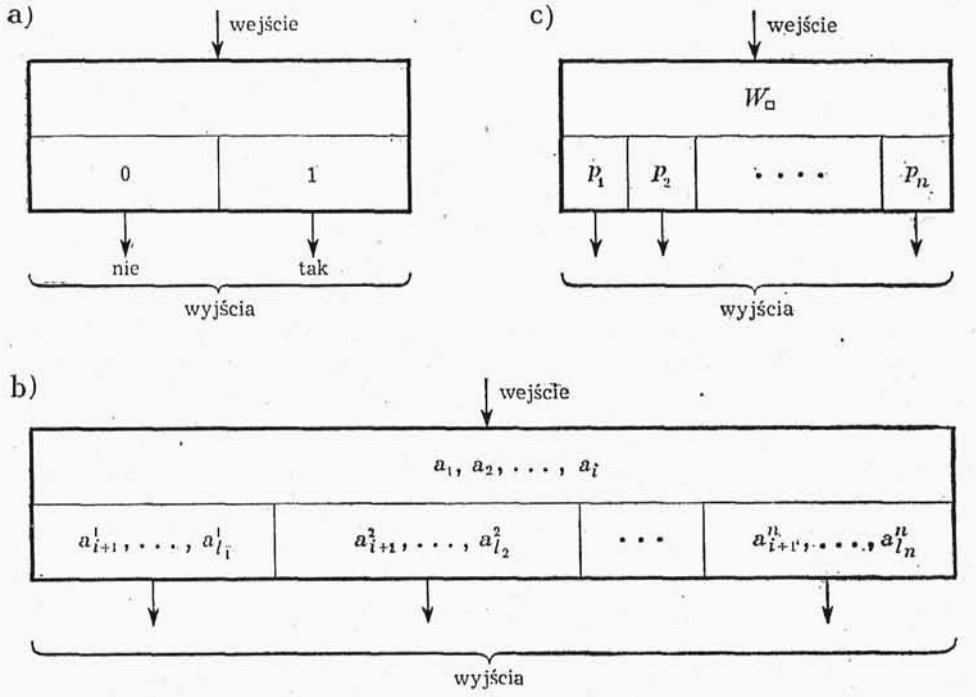
Te operatory, w których będziemy w trakcie dokonywania programu podstawiali zmienne parametry, będziemy przedstawiali graficznie jak na rys. 3-3.

Operatory realizujące w programie czynności organizacyjne będziemy w zależności od charakteru tych czynności nazywali operatorami podstawiania, odnowienia, formowania itp.

Predykaty będziemy przedstawiali graficznie w postaci prostokątów. Prostokąty te dzielimy linią poziomą na dwie części. Dolną część dzielimy pionowymi liniami na tyle prostokątów, ile elementów ma zbiór wyników czynności $\{s_j\}$ realizowanych przez predykat (rys. 3-4). Dowolny taki prostokąt odpowiadający wynikowi p czynności s_j nazywamy prostokątem kierunku p.

Będziemy wyróżniali trzy podstawowe typy predykatów w zależności od wyniku czynności s_j (klasyfikacja poniższa pochodzi od S. Paszkowskiego):

1. Wynikiem czynności s_j może być jedna z dwóch wartości, pierwsza z nich występuje wtedy i tylko wtedy, gdy jest spełniona własność W pewnych przedmiotów, druga



Rys. 3-5. Graficzne przedstawienie predykatów

zaś z nich występuje wtedy i tylko wtedy, gdy nie jest spełniona własność W. Tak określony predykat będziemy nazywali predykatem elementarnym i przedstawiali graficznie jak na rys. 3-5a.

2. Wynikiem czynności s_j może być jedna z n wartości, przy czym k-ta z nich ($k = 1, 2, \ldots, n$) występuje wtedy i tylko wtedy, gdy jest spełniona własność W_k pewnych przedmiotów, przy czym własności $W_1, W_2, \ldots, W_n$ wzajemnie się wyłączają i dopełniają. W_k formułujemy w symbolice teorii, do której należy rozwiązane zagadnienie, w postaci ciągu symboli $a_1, a_2, \ldots . a_i; a^k_{i+1}, \ldots, a^k_{l_k}$ ($a_1, a_2, \ldots, a_i$ nie zależą od k). Predykat o powyższej postaci przedstawiamy graficznie jak na rys. 3-5b.

3. Wynikiem czynności a może być jedna z n wartości i k-ta z nich ($k = 1,2..., n$) występuje wtedy i tylko wtedy, gdy jest spełniona własność W_{p_k} zależnie od parametru p_k, przy czym własności $W_{p_1}, W_{p_2}, ..., W_{p_n}$ wzajemnie się wykluczają i dopełniają. Tak określony predykat będziemy nazywali predykatem z parametrem i przedstawiali graficznie jak na rys. 3-5c.

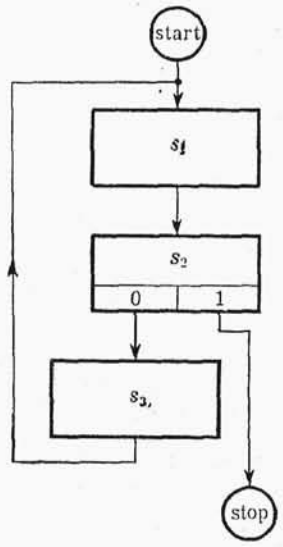
Schematy blokowe tworzymy łącząc wyjścia operatorów i predykatów z wejściami w kolejności zadanej nam przez formuły, które chcemy zrealizować. Punkty wspólne połączeń oznaczamy za pomocą kropek dla odróżnienia ich od skrzyżowań połączeń.

Przy układaniu schematów blokowych wyróżniamy dwa etapy:

1) tzw. schemat blokowy płaski, którym pokazujemy tylko związek logiczny między poszczególnymi operatorami a predykatami (schemat blokowy płaski, inaczej będziemy nazywali schematem logicznym);

2) tzw. schemat blokowy liniowy, w którym uwzględniamy nie tylko związek logiczny między poszczególnymi operatorami a predykatami, ale również kolejność elementarnych operacji, z których są zbudowane operatory i predykaty.

Inaczej mówiąc, schemat liniowy ustala system adresowania naszego programu.



Rys. 3-6. Rysunek do przykładu 3-2

Przykład 3-2(1). Jedną z metod rozwiązywania równań przestępnych jest metoda iteracji. Dla zastosowania tej metody musimy równanie przedstawić w postaci $x = g(x)$. Dla obliczenia $n + 1$ przybliżenia pierwiastka, znając n-te przybliżenie korzystamy ze związku $x_{n+1} = g(x_n)$. Oczywiście, postępowanie takie jest możliwe wtedy i tylko wtedy, gdy w otoczeniu pierwiastka funkcja $g(x)$ jest słabo zmienna, w przeciwnym przypadku proces iteracyjny będzie rozbieżny. Kolejne przybliżenie obliczamy tak długo, aż zostanie spełniony związek $|x_n - x_{n+1}| < \varepsilon$, gdzie ε jest z góry zadaną liczbą dodatnią. Rozwiązanie równania $x = g(x)$ metodą iteracyjną przedstawimy w postaci trzech następujących czynności (w tym przypadku $m = 3$):

1. Czynności s_1 — obliczenie nowego przybliżenia na podstawie poprzedniego przybliżenia.

2. Czynności s_2 — porównanie nowego przybliżenia z poprzednim i sprawdzenie, czy wartość bezwzględna ich jest mniejsza od ε. Jeśli tak jest, to obliczenie zakończymy, ponieważ otrzymaliśmy pożądaną dokładność, jeśli nie, to musimy wykonać czynność s_3.

3. Czynność s_3 — przygotowanie do czynności s_1 poprzez umieszczenie $n + 1$ przybliżenia na miejsce n-tego, po czym zostaje wykonana czynność s_1.

Schemat blokowy odpowiadający wyżej omówionym czynnościom s_1, s_2, s_3 jest podany na rys. 3-6.

(1) Przykład ten podał S. Paszkowski.

3.4. PROGRAMY LINIOWE

Programy liniowe są programami o najprostszej strukturze logicznej. W programach liniowych czynności $s_1, s_2, \ldots, s_m$, z których składa się program, są wykonywane kolejno, każda z tych czynności jest wykonywana dokładnie jeden raz. Zaletą programów liniowych jest prędkość ich wykonywania przez maszynę. Wadą tych programów jest ilość miejsca, jaką zajmują one w pamięci. Wyobraźmy sobie program liniowy składający się z 1000 rozkazów, załóżmy dalej, że maszyna pracuje z prędkością 100 operacji/s, wówczas cały program zostałby wykonany przez maszynę po upływie 10 s. Na to więc, aby zapewnić pracę maszyny przez jedną godzinę, potrzebna jest pamięć o pojemności 360 000 słów krótkich. Przykład programu liniowego podaliśmy w punkcie 3.1.

3.5. PROGRAMY Z ROZWIDLENIAMI

Problemy, w których w różnych przedziałach rozwiązania są dane za pomocą różnych formuł, rozwiązujemy za pomocą programu z rozwidleniami. Program z rozwidleniami zawiera predykaty sprawdzające warunki, operatory zwane wariantami i wreszcie operator wykonujący końcowe obliczenia. Wariantem nazywamy zespół operatorów i predykatów (lub jeden predykat), który może być lub nie być wykonany przy jednorazowym zastosowaniu programu. Rozpatrzymy obecnie dwa elementarne przykłady schematów blokowych programów z rozwidleniami. Pierwszy z tych przykładów został zaczerpnięty z książki Bondarenki, Płotnikowa i Połzowa (Bibliografia [3]). W drugim przykładzie po narysowaniu schematu blokowego zakodujemy jeszcze kolejne operatory i predykaty.

Przykład 3-3. Pewne zagadnienie z balistyki zewnętrznej sprowadza się do następującej formuły matematycznej:

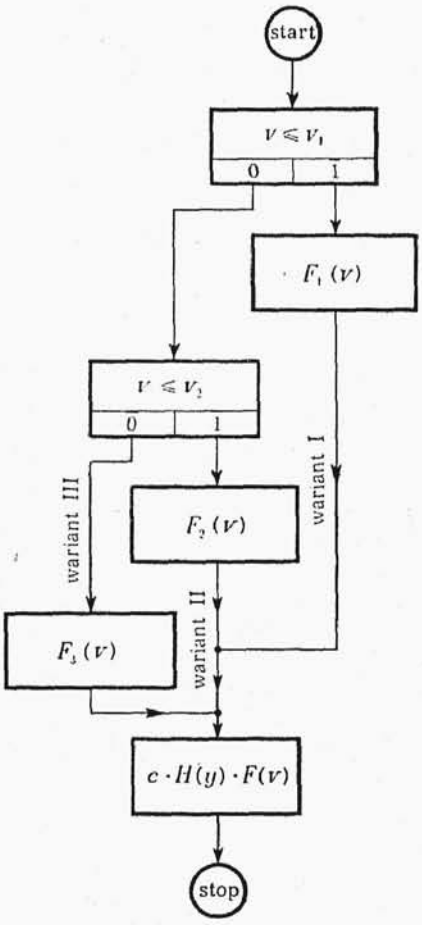
$$\mathcal{J} = c\,H(y)\;F(v).$$

Współczynnik balistyczny c i wartości funkcji gęstości powietrza $H(y)$ przyjmujemy za znane, a funkcję oporu powietrza $F(v)$ będziemy obliczali na podstawie wzorów przybliżonych:

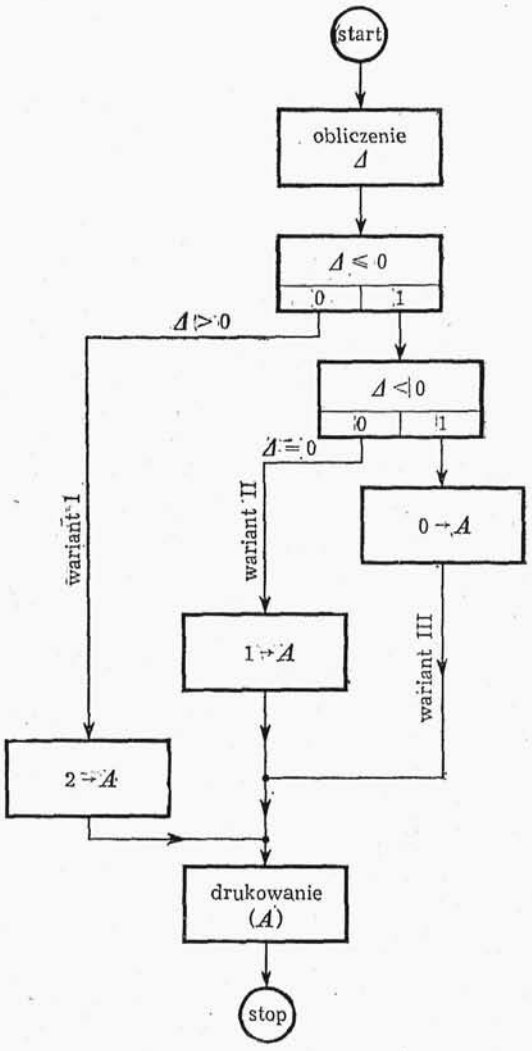
$$\begin{aligned} F(v) &= F_1(v) = a_1v^2 + b_1v + c_1 \quad \text{dla} \quad v \leqslant v_1, \\ F(v) &= F_2(v) = a_2v^2 + b_2v + c_2 \quad \text{dla} \quad v_1 < v \leqslant v_2, \\ F(v) &= F_4(v) = a_3v^2 + b_3v + c_3 \quad \text{dla} \quad v > v_2. \end{aligned}$$

Program dla obliczania wartości funkcji F będzie zawierał trzy warianty. Schemat blokowy tego programu podajemy na rys. 3-7.

Przykład 3-4. Przypuśćmy, że dla pewnych celów potrzebny jest nam program obliczający ilość pierwiastków rzeczywistych algebraicznego równania kwadratowego. Oznaczmy ilość pierwiastków rzeczywistych równania kwadratowego przez n (oczywiście n przebiega zbiór $\{0,1,2\}$). Dla uproszczenia przyjmiemy, że współczynniki rozpatrywanych przez nas równań są mniejsze co do modułu od jedności.



Rys. 3-7. Rysunek do przykładu 3-3



Rys. 3-8. Rysunek do przykładu 3-4

Schemat blokowy programu przedstawiliśmy na rys. 3-8. Rozmieszczając stałe jak w tabl. 3-4, możemy zakodować kolejne operatory i predykaty programu w postaci przedstawionej w tabl. 3-5.

Tablica 3-4

Adres	$p+4000$	$p+4002$	$p+4004$	$p+0006$	$p+0007$
Zawartość	a	b	c	2^{-4}	2^{-3}

Tablica 3-5

Kolejny adres	Kolejny rozkaz		Czynności wykonywane
$k+0000$	27	$p+4002$	$b \rightarrow M$
1	16	$p+4002$	b^2
2	21	0003	$\frac{1}{8} b^2$
3	15	0000	zaokrąglenie $\frac{1}{8} b^2$
4	14	$p+4002$	przesyłamy $\frac{1}{8} b^2$ na miejsce b
5	27	$p+4000$	$a \rightarrow M$
6	16	$p+4004$	$a \cdot c$
7	21	0001	$\frac{1}{2} ac$
$k+0010$	15	0000	zaokrąglenie $\frac{1}{2} ac$
1	11	$p+4002$	$\frac{1}{2} ac - \frac{1}{8} b^2 = -\frac{1}{8} \Delta$
2	03	$k+0021$	$\Delta > 0$ skok
3	26	0000	
4	06	$k+0017$	skok przy $\Delta = 0$
5	21	0004	} kładziemy $n=0$
6	02	$k+0022$	} wariant 3
7	12	$p+0006$	} kładziemy $n=1$
$k+0020$	02	$k+0022$	} wariant 2
1	12	$p+0007$	kładziemy $n=2$ wariant 1
2	25	0000	drukowanie
3	05	0000	stop

Przy założeniu, że k jest parzyste, otrzymamy

$$p = k + 0024;$$

przeadresowując według powyższego związku otrzymany program w ostatecznej postaci.

Powyższe przykłady mimo swojego elementarnego charakteru pokazują nam budowę typowego programu z rozwidleniami.

3.6. PROGRAMY CYKLICZNE I ITERACYJNE

Jeśli chcielibyśmy dodać n liczb (przy założeniu, że suma tych liczb jest mniejsza co do modułu od jedności) za pomocą programu liniowego, to program taki składałby się z $n + 1$ rozkazów i maszyna wykonywałaby ten program w $n + 1$ krokach. Program

taki dla dużych n zajmowałby zbyt wiele miejsca w pamięci. Dla $n > 13$ zamiast programu liniowego opłaca się stosować program cykliczny składający się z 13 słów krótkich. Zaletą takiego programu jest mała ilość miejsca, które zajmuje w pamięci maszyny. Wadą jest powolność wykonywania pracy przez maszynę; maszyna dla zsumowania n liczb za pomocą programu cyklicznego musi wykonać $8n + 4$ kroków, czyli blisko o rząd wielkości więcej niż przez sumowanie za pomocą programu liniowego.

Z punktu widzenia obliczeń na maszynach cyfrowych ważne jest rozróżnienie:

1) formuł iteracyjnych, dla których z góry znamy ilość powtórzeń,

2) formuł iteracyjnych, dla których nie znamy z góry ilości powtórzeń, dopiero zaś na podstawie wyników cząstkowych podejmujemy decyzję, czy wykonać następne powtórzenie, czy też przerwać iterację ze względu na osiągniętą już wystarczającą dokładność.

Programy opisujące formuły typu (1) będziemy nazywali krótko programami cyklicznymi, programy zaś opisujące formuły typu (2) będziemy nazywali krótko programami iteracyjnymi.

W programach cyklicznych wyróżniamy pięć zasadniczych elementów:

B — operator służący do przygotowania cyklu (lub do przygotowania przybliżenia początkowego),

C — operator służący do wykonania obliczeń kolejnego kroku (lub kolejnego przybliżenia),

D — operator służący do przygotowania operatora C do wykonania następnego kroku obliczeń, może to być tzw. operator przeadresowania,

E — predykat zwany krótko licznikiem kroków — pracujący w następujący sposób: jeśli krotność powtórzeń k jest mniejsza od z góry zadanej liczby n (n—tzw. krotność cyklu), to predykat E powoduje powtórzenie cyklu, jeśli zaś krotność powtórzeń k jest równa z góry zadanej liczbie n, to omawiany predykat E powoduje przekazanie sterowania kolejnemu operatowi programu,

F — operator służący do ostatecznego sformowania wyniku i przesłania go dalej.

Schemat blokowy jednej z możliwych organizacji cyklu jest pokazany na rys. 3-9.

W programach iteracyjnych podobnie jak w programach cyklicznych wyróżniamy pięć zasadniczych elementów:

B — operator służący do obliczenia przybliżenia początkowego,

C — operator wykonujący kolejny krok iteracyjny,

D — operator służący do przygotowania operatora C do wykonania następnego kroku iteracyjnego (może to być tzw. operator przeadresowania),

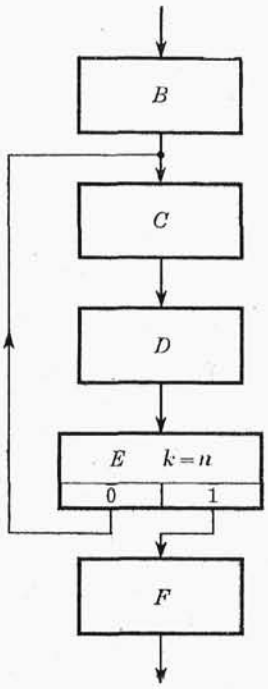
E — predykat sprawdzający dokładność wyniku po kolejnej iteracji, w zależności od tego powoduje wykonanie następnego powtórzenia iteracji albo przekazuje sterowanie kolejnemu operatorowi programu.

F — operator służący do ostatecznego sformowania wyniku i przesłania go dalej.

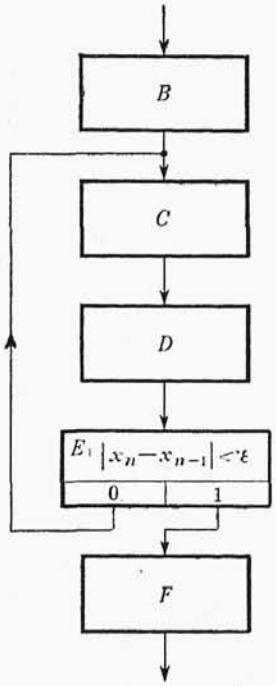
Schemat blokowy jednej z możliwych organizacji iteracji pokazany jest na rys. 3-10.

Zajmiemy się obecnie bliżej operatorami D i C w programach cyklicznych i programach iteracyjnych. Wydawać by się mogło, że wyróżnienie operatora D jest czymś sztucznym i że można by operatory C i D zastąpić jednym operatorem. Jednakże czyn-

ności wykonywane przez operator C i operator D są skrajnie różne. Operator D wykonuje wszelkie prace organizacyjne w omawianych programach. Operator C realizuje formułę iteracyjną i jest operatorem arytmetycznym.



Rys. 3-9. Schemat blokowy cyklu



Rys. 3-10. Schemat blokowy iteracji

Przez czynności organizacyjne będziemy rozumieli

1) przesyłanie danych liczbowych i rozkazów z jednych komórek pamięci do drugich,

2) przeadresowanie.

Punkt (2) wymaga szerszego omówienia; bardzo często np. przy sumowaniu szeregów potęgowych współczynniki szeregu znajdują się w kolejnych komórkach pamięci, wówczas opłaca się (jak już wspominaliśmy) stosować program cykliczny; w tym celu układamy program do obliczenia pierwszego wyrazu szeregu, a następnie po obliczeniu pierwszego wyrazu zastępujemy w naszym programie adres pierwszego współczynnika przez adres drugiego, obliczamy drugi wyraz — sumujemy go z pierwszym, dalej zastępujemy adres drugiego współczynnika przez adres trzeciego itd. Takie postępowanie nazywamy przeadresowaniem cyklu (lub iteracji). Oczywiście, postępowanie takie jest ekonomiczne tylko wówczas, gdy współczynniki są umieszczone pod kolejnymi adresami długimi, wówczas przeadresowanie polega na dodawaniu (lub odejmowaniu) 2^{-15} do części adresowych odpowiednich rozkazów. W pewnych przypadkach można uzyskać duże uproszczenie (lub oszczędność czasu) przez jednoczesne przeadresowanie

par rozkazów. Pamiętamy bowiem o tym, że jedno słowo długie składa się z dwóch słów krótkich.

Rozróżnienia programów cyklicznych od programów iteracyjnych dokonujemy z dwóch powodów:

1) ze względu na bardziej złożoną budowę predykatu w programie iteracyjnym,

2) ze względu na to, że czas potrzebny na wykonanie programu cyklicznego możemy dokładnie określić.

Programy cykliczne będziemy omawiali szczegółowo w punkcie 3.7, obecnie zaś podamy prosty przykład programu iteracyjnego.

Przykład 3-5. Dana jest liczba $1 > x > 0$, należy znaleźć przybliżenie pierwiastka z liczby x z dokładnością $\varepsilon > 0$ korzystając z algorytmu Newtona

$$y_{n+1} = \frac{y_n}{2} + \frac{x}{2y_n}, \tag{3-10}$$

z przybliżeniem początkowym

$$y_0 = 1 - 2^{-33}. \tag{3-11}$$

Tablica 3-6

Operator albo predykat	Kolejny adres	Kolejny rozkaz		Czynność wykonywana przez kolejny rozkaz
B	$k+0000$	12	$<1-2^{-33}>$	
	1	14	$b+4004$	$y_0 \longrightarrow b+4004$
D	2	12	$b+4004$	przesłanie y_n z $b+4004$
	3	14	$b+4002$	do $b+4002$
C	4	21	0001	obliczanie $\frac{1}{2}y_n$ i przesłanie do $b+4004$
	5	14	$b+4004$	
	6	12	$b+4000$	$x \longrightarrow A$
	7	17	$b+4002$	$x : y_n \longrightarrow A$
	$k+0010$	21	0001	$\frac{1}{2}\left(x : y_n\right)$
	1	10	$b+4004$	$\frac{1}{2}\left(x : y_n\right) + \frac{1}{2}y_n$
	2	14	$b+4004$	$y_{n+1} \longrightarrow b+4004$
E	3	11	$b+4002$	$y_{n+1} - y_n$
	4	26	0000	$\lvert y_{n+1} - y_n \rvert$
	5	11	$<\varepsilon>$	$\lvert y_{n+1} - y_n \rvert - \varepsilon$
	6	06	$k+0002$	skok przy $\lvert y_{n+1} - y_n \rvert - \varepsilon \geqslant 0$
F	7	12	$b+4004$	$\sqrt{x} \longrightarrow A$
	$k+0020$	05	$s+0000$	stop z pobraniem rozkazu $s+0000$

Zakładamy, że liczba x znajduje się pod adresem $b + 4000$. Adresów $b + 4002$ i $b + 4004$ będziemy używali jako komórek roboczych.

Program nasz będzie się znajdował pod adresami $k + 0000 \div k + 0021$. Kolejne rozkazy programu podajemy w tabl. 3-6.

3.7. STEROWANIE CYKLAMI

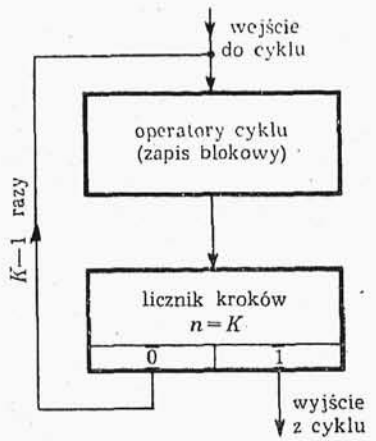
3.7.0. W niniejszym punkcie zajmiemy się szczegółowo sterowaniem cyklami; sterowanie rozwiedleniami omówimy dosyć marginesowo, przy okazji uwag nad poszczególnymi metodami sterowania cyklami.

W zasadzie rozróżniamy cztery metody sterowania cyklami;

1) sterowanie arytmetyczne,
2) sterowanie zmiennymi adresami albo zmiennymi rozkazami programu,
3) sterowanie za pomocą skal logicznych,
4) sterowanie za pomocą wielokrotnego wywoływania.

Ostatnią z wymienionych metod opłaca się stosować tylko wówczas, jeżeli organizacja danej UPMC pozwala w prosty sposób wywoływać podprogramy. W przyjętej przez nas przykładowej UPMC metoda ta daje się z powodzeniem stosować.

3.7.1. Sterowanie arytmetyczne. Sterowanie arytmetyczne opiera się na tzw. liczniku kroków; przez licznik kroków będziemy rozumieli predykat liczący krotność



Rys. 3-11. Rysunek do przykładu

powtórzeń wykonywania danego zespołu rozkazów, a gdy krotność ta jest równa z góry zadanej liczbie K, wówczas licznik kroków przekazuje sterowanie z góry ustalonemu operatorowi. Jedna z możliwych organizacji cyklu z licznikiem kroków jest przedstawiona na rys. 3-11.

Przykład licznika kroków:

$$\begin{array}{rll} k+0000 & 12 & s+0000, \\ 1 & 10 & <2^{-16}>, \\ 2 & 14 & s+0000, \\ 3 & 03 & p+0000, \end{array}$$

gdzie przez adres $p+0000$ oznaczamy adres pierwszego rozkazu operatora cyklu, zaś zawartość początkowa adresu $s+0000$ jest $-K\cdot 2^{-16}$ $(0<K\leqslant 2^{-16}-1)$.

Przedstawiony licznik kroków pracuje w następujący sposób: po każdym przejściu przez sekwencję operatorów cyklu liczba K zostaje zmniejszona o jedność (realizujemy to przez powiększenie liczby $-K$ o jedynkę), trwa to tak długo, aż otrzymamy liczbę zero, wówczas sterowanie zostaje przekazane rozkazowi $k + 0004$.

Oczywiście, że liczniki kroków można konstruować na bardzo wiele sposobów, przedstawiony wyżej licznik jest jednakże wygodny ze względu na swoją prostotę.

Może się zdarzyć, że omawiany przez nas cykl pracuje wewnątrz drugiego cyklu, wówczas istnieje konieczność odnawiania zawartości początkowej adresu $s + 0000$, dokonujemy tego za pomocą operatora odnowienia składającego się z dwóch rozkazów:

$$\left.\begin{array}{rrl} q + 0000 & 12 & s + 0001, \\ 1 & 14 & s + 0000, \end{array}\right\} \qquad (3\text{-}12)$$

gdzie pod adresem $s + 0001$ stawiamy liczbę $-K \cdot 2^{-16}$ (K-krotność cyklu).

Dla $K = 2$ zamiast licznika kroków i operatora odnowienia wartości początkowej tegoż licznika wygodniej jest użyć licznika samoodnawiającego się, tzw. sita. Sito takie przekazuje co drugi raz sterowanie rozkazowi następnemu, w kolejności: *skok, prosto, skok, prosto* itd., gdy początkową zawartością adresu $b + 0000$ jest zero

$$\left.\begin{array}{rrl} k + 0000 & 13 & b + 0000, \\ 1 & 11 & <\beta>, \\ 2 & 14 & b + 0000, \\ 3 & 03 & <\text{początek cyklu}>, \end{array}\right\} \qquad (3\text{-}13)$$

gdzie β jest to dowolny dodatni parametr lub któryś z rozkazów $00 \div 17$ (pod adresami $b + 0000$ znajdują się kolejne liczby $0, -\beta, 0, -\beta, 0, \ldots$).

Tablica 3-7

Kolejny adres	Kolejny rozkaz		Czynności wykonywane przez kolejny rozkaz
$p+0000$	14	$b+4000$	$x \cdot 2^{-33} \rightarrow$ komórki roboczej
1	27	$b+4000$	$x \cdot 2^{-33} \rightarrow M$
2	16	$b+4000$	obliczenie $x^2 \cdot 2^{-66}$
3	20	0041	obliczenie $x^2 \cdot 2^{-33}$
4	14	$b+4000$	$x^2 \cdot 2^{-33} \rightarrow$ komórki roboczej
5	12	$b+0002$	licznik kroków $\rightarrow A$
6	10	$<2^{-16}>$	powiększenie zawartości licznika kroków o „jedynkę“.
7	14	$b+0002$	
$p+0010$	03	$p+0002$	skok przy $n<K-1$, gdy $n=K-1$, przechodzimy do następnego rozkazu
1	12	$b+4000$	$x^k \cdot 2^{-33} \rightarrow A$
2	05	0000	

Gdybyśmy chcieli uzyskać odwrotną kolejność przekazywania sterowania (kolejność *prosto, skok, prosto, skok* itd.) początkową wartość $b + 0000$ musimy położyć równą $-\beta$.

Jako ilustrację do sterowania arytmetycznego pokażemy program dla obliczania K-tej potęgi liczby całkowitej, przy założeniu, że liczba całkowita x spełnia warunek

$$|x| \leqslant \sqrt[K]{2^{33}-1}. \tag{3-14}$$

Liczby całkowite x będziemy zapisywali w maszynie w postaci $x \cdot 2^{-33}$, musimy jednak pamiętać o konieczności przenormowania wyników działań arytmetycznych maszyny. Zakładając, że liczby $x \cdot 2^{-33}$ znajduje się w akumulatorze, otrzymujemy program postaci przedstawionej w tabl. 3-7.

3.7.2. Sterowanie zmiennymi adresami albo zmiennymi rozkazami programu. Najczęściej stosowaną metodą sterowania cyklami jest metoda zmiennych adresów. Idea tej metody jest następująca: w trakcie każdorazowego powtórzenia sekwencji rozkazów cyklu, przeadresowujemy jeden rozkaz (lub kilka rozkazów), np. ze względu na konieczność wykorzystania w każdym kroku innego współczynnika, otrzymany w ten sposób rozkaz porównujemy z ostateczną postacią tego rozkazu (postać ostateczna — postać tego rozkazu po ostatnim obiegu cyklu), w przypadku gdy postacie te są identyczne, następuje wyjście z cyklu. Przykład takiego postępowania podamy obecnie.

Przykład 3-6. Niech będzie dana liczba x co do modułu mniejsza od jedności, oraz niech będzie danych ciąg $n+1$ liczb $a_0, a_1, a_2, \ldots, a_n$, z których każda jest również co do modułu mniejsza od jedności; przy założeniu, że wszystkie wyniki pośrednie

Tablica 3-8

Adres	$a+4000$	$a+4002$	$a+4004$	$\cdots$	$a+4000+2n$
Zawartość	a_0	a_1	a_2	$\cdots$	a_n

Tablica 3-9

Operator lub predykat	Kolejny adres	Kolejny rozkaz	
B	$k+0000$	14	$b+4000$
	1	27	$b+4000$
	2	12	$a+4000+2n$
	3	14	$b+4000$
C	$\rightarrow k+0004$	16	$b+4000$
	5	15	0000
	6	10	$a+4000+2(n-1)$
	7	14	$b+4000$
D	$k+0010$	12	$k+0006$
	1	11	$<2^{-15}>$
	2	14	$k+0006$
E	3	11	$<10\ a+4000>$
	4	06	$k+0004$
F	5	12	$b+4000$
	6	05	0000

i wynik końcowy są co do modułu mniejsze od jedności, obliczamy wartość wielomianu w postaci Hornera.

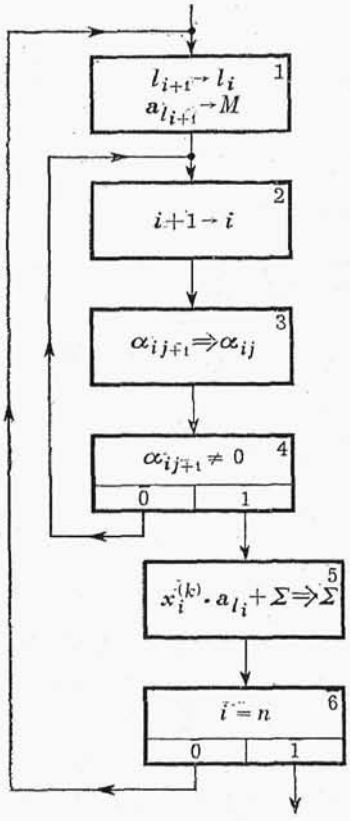
$$(\ldots(a_n x + a_{n-1})\,x + a_{n-2})\,x + \ldots a_1)\,x + a_0. \tag{3-15}$$

Załóżmy, że x znajduje się w akumulatorze, a ponadto wynik umieszczamy również w akumulatorze.

Miejsce robocze ma adres $b + 4000$, zaś liczby a_i są umieszczone w pamięci pod kolejnymi adresami (tabl. 3-8).

W tablicy 3-9 podajemy program podzielony na kolejne operatory i predykaty.

Metoda zmiennych rozkazów stosowana jest raczej dla sterowania rozwidleniami w programie, polega ona na tym, że w trakcie wykonywania programu zostaje sformowany pewien rozkaz, a następnie przesłany na dalsze miejsca w programie; w zależności od rodzaju tego rozkazu sterowanie zostaje przekazane określonemu wariantowi programu.



Rys. 3-12. Rysunek do przykładu 3-7

3.7.3. Sterowanie skalam logicznymi. Rozróżniamy dwa przypadki ze względu na sterowanie za pomocą skal logicznych:

1) sterowanie cyklami,

2) sterowanie cyklem pracującym w cyklu.

W pierwszym z rozważanych przypadków stosujemy sterowanie za pomocą skali logicznej tylko wówczas, gdy da nam to oszczędność w ilości operacji lub oszczędność w czasie (przykładem takim może być metoda „pilota" w postawowym programie wprowadzającym, punkt 5.1.1). W drugim z rozważanych przypadków korzystamy z metody skal logicznych tylko wówczas, gdy zewnętrzny i wewnętrzny cykl są powtarzane niewielką ilość razy i poprzez stosowanie skali logicznej unikamy odnawiania wartości początkowej licznika kroków wewnętrznego cyklu.

Najprostszą skalą logiczną będzie jedynka ustawiona na odpowiednim miejscu w słowie np. krótkim; po każdym wykonaniu sekwencji rozkazów cyklu następuje przesunięcie skali o jedną pozycję w lewo, jeśli na pierwszej pozycji akumulatora znajduje się zero, to następuje powtórzenie cyklu, jeśli zaś na pierwszej pozycji akumulatora znajduje się jedynka, to następuje przekazanie sterowania następnej sekwencji rozkazów. Oczywiście, można używać skali odwrotnej; jedynce odpowiada pewne wykonanie sekwencji rozkazów, zeru odpowiada przekazanie sterowania dalej.

W przypadku gdy mamy określić wybór jednej z kilku dróg (trzech, czterech itd), wówczas pojedyncza skala logiczna nie wystarcza, w tym przypadku stosujemy tzw.

skale logiczne złożone, np. dla wyboru jednej z trzech dróg stosujemy skalę logiczną złożoną z par liczb binarnych.

Przykład 3-7. Jeśli mamy rozwiązać na UPMC układ n równań algebraicznych, w którym znaczna część współczynników macierzy równa się zeru, to aby poprawić wykorzystanie pamięci maszyny, stosujemy skale logiczne. W tym celu tworzymy macierz zero-jedynkową α_{ij} opisującą w następujący sposób macierz współczynników równań a_{ij}: jeśli wyraz stojący na przecięciu i-tego wiersza i j-tej kolumny ($i, j = 1, 2, \ldots, n$) macierzy a_{ij} jest zerem, to kładziemy $\alpha_{ij} = 0$, jeśli zaś jest różny od zera, kładziemy $\alpha_{ij} = 1$. Następnie tworzymy ciąg niezerowych wyrazów macierzy a_{ij} ($a_{l_1}, a_{l_2}, \ldots, a_{l_n}$) poprzez ustawienie niezerowych kolejnych wyrazów macierzy a_{ij} jeden za drugim i skreślenie wyrazów zerowych. Na rysunku 3-12 pokazana jest część schematu blokowego dla rozwiązania metodą iteracji układu równań algebraicznych

Tablica 3-10

Operator lub predykat	Kolejny adres	Kolejny rozkaz		Uwagi
1	→ r+0000	12	r+0003	$l_{i+1} \rightarrow l_i$
	1	10	$<2^{-15}>$	
	2	14	r+0003	
	3	27	$<a_{l_i}>$	$a_{l_i} \rightarrow M$
2	→ 4	12	r+0013	
	5	10	$<2^{-15}>$	
	6	14	r+0013	
3	7	12	b+4000	badanie skali logicznej
	r+0010	23	0001	
	1	14	b+4000	
4	2	06	r+0004	
5	3	16	$<x_i^{(k)}>$	
	4	15	0000	
	5	10	b+4002	
	6	14	b+4002	
6	7	12	r+0013	
	r+0020	11	$<16<x_n^{(k)}>>$	
	1	03	r+0000	

liniowych $X = AX + B$ z macierzą, w której znaczna część elementów jest zerami. Dla uproszczenia rozważań przyjmiemy, że $n \leqslant 34$. Przedstawiony na rys. 3-12 fragment schematu dotyczy obliczenia $x_j^{(k+1)} = \sum_{i=1}^{n} a_{ij} x_i^{(k)} + b_j$ dla ustalonego j; j-ty wiersz macierzy α_{ij} znajduje się w odpowiedniej komórce roboczej. Ten właśnie wiersz macierzy α_{ij} jest przykładem skali logicznej, który chcemy pokazać. Dalej przyjęliśmy, że skala ta jest umieszczona pod adresem b + 4000, kolejne zaś sumy częściowo two-

rzymy przy obliczeniach $x_j^{(k+1)}$, oznaczamy przez $\sum$ i umieszczamy pod adresem $b + 4002$. Zakładając, że współczynniki również są tak dobrane, aby nie mogło nastąpić przekroczenie zakresu, możemy fragment schematu blokowego przedstawiony na rys. 3-12 zrealizować za pomocą układu rozkazów przedstawionych w tabl. 3-10.

3.7.4. Sterowanie za pomocą wielokrotnego wywołania. Idea tej metody jest następująca: przypuśćmy, że operatory cyklu, który mamy wykonać K-krotnie, zawierają łącznie p rozkazów, umieścimy je w pamięci pod adresami $n + 1 \div n + p$; pod adresem n zostawimy wolne miejsce (na ślad), pod adresem zaś $n + p + 1$ umieszczamy rozkaz $01\,n$ (skok z podstawieniem). W programie głównym umieszczamy jeden pod drugim rozkazy $04\,n$ (skok ze śladem).

Sterowanie cyklem przebiega następująco: sterowanie maszyny pobiera i wykonuje pierwszy z rozkazów $04\,n$, powoduje to umieszczenie zawartości licznika rozkazów pod adresem n, po czym zostają wykonane kolejne rozkazy znajdujące się pod adresami $n + 1 \div n + p$, następnie zostaje wykonany rozkaz $01\,n$ zapisany pod adresem $n + p + 1$, który spowoduje pobieranie i wykonanie drugiego z rozkazów $04\,n$ w programie głównym itd, aż zostanie wykonany K-ty rozkaz $04\,n$ w programie głównym, wówczas po ostatnim wykonaniu rozkazów znajdujących się pod adresami $n + 1 \div n + p$ rozkaz $01\,n$ znajdujący się pod adresem $n + p + 1$ spowoduje pobranie i wykonanie pierwszego z rozkazów stojącego za ciągiem rozkazów $04\,n$ znajdujących się w programie głównym.

Gdy $K = 3$, metoda ta jest co do ilości zajmowanego miejsca w pamięci konkurencyjna z metodami podanymi w punktach 3.7.1. i 3.7.3. Natomiast jeśli chodzi o prędkość jest ona szybsza od obu wyżej wymienionych metod. W punkcie 5.12 (stałoprzecinkowy program wprowadzająco - przeliczający dla liczb w systemie dziesiątkowym) pokażemy przykład sterowania za pomocą wielokrotnego wywołania. W przykładzie tym $K = 10$, a metodę tę zastosowaliśmy dla uzyskania większej prędkości wprowadzenia liczb w systemie dziesiątkowym z jednoczesnym przeliczeniem ich na system binarny.

Przykład 3-8. Przypuśćmy, że dla pewnego celu musimy obliczyć wyrażenie $y = \sin(\sin(\sin(\sin x)))$, gdzie $|x| < 1$. Załóżmy, że dysponujemy programem do obliczania funkcji $\sin x$. Program taki działa następująco: należy umieścić liczbę x (mniejszą co do modułu od jedności) w akumulatorze, po czym wykonać rozkaz $04\ s + 0000$; jeśli rozkaz ten znajdował się pod adresem $k + 0001$, to po zakończeniu obliczeń $\sin x$, zostaje umieszczony w akumulatorze, po czym poprzez ślad program obliczenia sinusa przekaże sterowanie rozkazowi znajdującemu się pod adresem $k + 0002$. Dla obliczenia wyrażenia $y = \sin(\sin(\sin(\sin x)))$, wystarczy więc czterokrotnie wywołać program $\sin x$, po czym otrzymujemy w akumulatorze wartość y.

$k + 0000$	12	x,
1	04	$s + 0000$,
2	04	$s + 0000$,
3	04	$s + 0000$,
4	04	$s + 0000$,
5	05	0000.