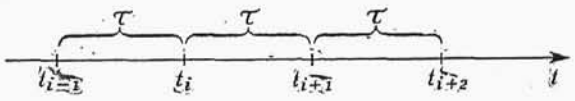
manych informacji, przyjęcie hipotezy co do informacji za bieżący przedział czasu, hipotez co do zachowania się układu w dalszych przedziałach czasu (predykcja) oraz podjęcie decyzji co do sterowania układu w następnym przedziale czasu, tak aby spełnił on narzucone warunki, τ_3 — czas potrzebny na przygotowanie odpowiedniego wykonawstwa przez układ sterowania.



Rys. 6-11.

Jak widać z powyższego, model używany do predykcji musi umożliwiać prześledzenie zjawiska w okresie krótszym od rzeczywistego zachodzenia tego zjawiska. Model ten musi być pewnym uproszczeniem zjawiska poprzez aproksymowanie funkcji opisujących proces dynamiczny pewnymi prostszymi funkcjami. Aproksymacji tej trzeba dokonać na gruncie teorii aproksymacji opartej o odległość funkcji na przedziale domkniętym:

$$\|f-g\| \underset{\text{df}}{=} \max_{t_i \leqslant t \leqslant t_{i+1}} |f(t)-g(t)|, \tag{6-16}$$

jak również na gruncie teorii aproksymacji opartej na odległości względnej funkcji na przedziale domkniętym:

$$\|f-g\| \underset{\text{df}}{=} \max_{t_i \leqslant t \leqslant t_{i+1}} \left|\frac{f(t)-g(t)}{f(t)}\right|. \tag{6-17}$$

7. KODY ZEWNĘTRZNE

(7.0. Uwagi wstępne, 7.1. Kody zewnętrzne typu interpreter, 7.2. Kody zewnętrzne typu compiler)

7.0. UWAGI WSTĘPNE

W punkcie 4.0 wyliczyliśmy etapy przygotowania problemu do obliczeń. Wymieniono tam między innymi punkty:

5) zaprogramowanie i zakodowanie poszczególnych operatorów i predykatów, na które rozbiliśmy program w wyniku przyjętej przez nas organizacji (punkt 4);

6) oparte na przyjętej przez nas organizacji zestawienia zakodowanego programu w jednolitym systemie adresowania rozkazów programu, analiza wykorzystania poszczególnych miejsc roboczych, rozmieszczenie programu, materiału liczbowego i miejsc roboczych w pamięci maszyny i wreszcie wydziurkowanie na taśmie programu i materiału liczbowego.

Oba wymienione punkty składają się z szeregu kroków łatwych do formalizacji. Czynności te z punktu widzenia możliwości człowieka są bardzo uciążliwe i są źródłem większości błędów w programach. Dlatego też w latach 1952—1953 rozpoczęto prace nad opracowaniem metod umożliwiających zapis programów w języku zbliżonym do języka używanego na codzień przez matematykę obliczeniową.

Prace te szły w dwóch kierunkach. Pierwszym kierunkiem stosunkowo prostym w realizacji było opracowanie języka, z którego w czasie rozwiązywania problemu UPMC tłumaczyłaby poszczególne wyrażenia na kod wewnętrzny i wykonywałaby je. Program tłumaczący oparty był o technikę interpretacyjną (punkt 6.1), skąd zresztą nazwa takich języków: „kody zewnętrzne typu *interpreter*" lub „kody interpretacyjne". Zasadniczą wadą interpretacyjnych kodów zewnętrznych jest konieczność wielokrotnego tłumaczenia poszczególnych wyrażeń zapisanych w kodzie zewnętrznym przy wykonywaniu obliczeń cyklicznych albo iteracyjnych. Jak czytelnik miał możność przekonania się na przykładzie podanym w punkcie 6.1, programy interpretacyjne działają stosunkowo wolno. Dlatego też interpretacyjne kody zewnętrzne w przypadku obliczeń cyklicznych zwalniają w zasadniczy sposób prędkość obliczeń. Zaletą ich jednak jest prostota kodowania i duża łatwość kontroli ułożonego programu.

Drugim kierunkiem było opracowanie języka, z którego wyrażenia UPMC w oparciu o specjalne programy tłumaczyłaby na język wewnętrzny, w którym to zapisywałaby program realizujący algorytmy analogiczne do algorytmów opisanych za pomocą języka zewnętrznego. Inaczej mówiąc, maszyna działa najpierw jako urządzenie tłumaczące, a następnie dopiero po przetłumaczeniu całego programu zaczyna działać jako urządzenie obliczeniowe. Programy tłumaczące język zewnętrzny na język wewnętrzny są programami kompilacyjnymi (punkt 6.2), skąd zresztą nazwa tych kodów — kody zewnętrzne typu *compiler*, bądź kompilacyjne kody zewnętrzne. Kompilacyjne kody zewnętrzne, w odróżnieniu od kodów interpretacyjnych, nie zmniejszają prędkości obliczeń cyklicznych. Przeciwnie, można by zaryzykować twierdzenie, że kody zewnętrzne typu *compiler* dają najlepsze rezultaty przy problemach cyklicznych o dużych ilościach wariantów obliczeń. Wynika to stąd, że raz przetłumaczony program z kodu zewnętrznego na wewnętrzny może być wykonywany dowolną ilość razy. Praktyka obliczeniowa wykazała, że dla większości problemów obliczeniowych przy korzystaniu ze średnich i dużych UPMC opłaca się stosować kompilacyjne kody zewnętrzne. Czas wykorzystania maszyny przy użyciu powyższych kodów dla problemów zawierających wielokrotne cykle niewiele przekracza czas wykorzystania maszyny przy programowaniu bezpośrednim w kodzie wewnętrznym, wliczywszy w to czas sprawdzenia programu na maszynie. Trzeba jednak podkreślić, że programy przekładające kody zewnętrzne są bardzo rozbudowane i zawierają około kilku tysięcy rozkazów. Wynika to przede wszystkim z faktu nieprzystosowania obecnie istniejących UPMC do realizowania

algorytmów automatycznego tłumaczenia. Ponadto ograniczenia wynikające ze struktury istniejących kodów nie pozwalają na stosowanie tych ostatnich przy rozwiązywaniu problemów stojących na granicy możliwości danej maszyny.

Należy podkreślić, że kody interpretacyjne prowadzą do znacznie krótszych programów tłumaczących (rzędu kilkuset rozkazów), aniżeli kody kompilacyjne. Ponadto korzystanie z kodów interpretacyjnych nie wymaga znajomości kodu wewnętrznego maszyny. Dlatego też kody zewnętrzne typu interpreter o prostej budowie i mnemotechnicznych nazwach operacji szczególnie nadają się do pracy obliczeniowej dla specjalistów różnych dziedzin nie znających organizacji danej maszyny. Umiejętność korzystania z takiego kodu wymaga zaledwie kilkudziesięciu godzin nauki.

7.1. KODY ZEWNĘTRZNE TYPU INTERPRETER

Jak już wspominaliśmy w punkcie 7.0, kody zewnętrzne typu *interpreter*, są tłumaczone i wykonywane kolejno symbol po symbolu. Dlatego też symbolika interpretacyjna kodów zewnętrznych musi być dostosowana do techniki interpretacyjnej. W przeciwnym przypadku otrzymalibyśmy bardzo niekorzystne stosunki czasów rozwiązywania zadań w kodzie zewnętrznym w porównaniu do czasów rozwiązywania tych zagadnień bezpośrednio w kodzie wewnętrznym maszyny.

Przedstawiony dalej kod zewnętrzny został opracowany pod kątem prostoty interpretacji przez S. Paszkowskiego (Bibliografia [14]). Odpowiednikami rozkazów w kodzie zewnętrznym są bloki, które to z kolei składają się z tzw. segmentów — odpowiedników części operacyjnych i adresowych rozkazów. Będziemy rozróżniali siedem typów segmentów.

Typ α. Do tego typu zaliczamy symbole działań i funkcji kodowane za pomocą symboli trzyliterowych:

1) ZER — zerowanie,
2) TRA — przesyłanie,
3) SQU — podnoszenie do kwadratu,
4) ADD — dodawanie
5) SUB — odejmowanie
6) MUL — mnożenie,
7) DIV — dzielenie,
8) SQR — obliczenie pierwiastka kwadratowego,
9) UNJ — skok bezwarunkowy,
10) EQJ — porównanie liczb i skok w przypadku, gdy porównane liczby są identyczne,
11) INA — powiększenie zawartości rejestru modyfikacji,
12) STO — stop,
13) SIN — obliczanie sinusa,
14) EXP — obliczanie wartości funkcji wykładniczej.

Tego typu symbole działań tworzymy w zależności od potrzeb obliczeniowych, dołączając do programu interpretacyjnego odpowiednie podprogramy.

Typ β. Do tego typu zaliczamy adresy liczb, na których wykonujemy działania zmiennoprzecinkowe. Adresy te kodujemy za pomocą trzycyfrowych liczb dziesiętnych.

$$i \quad j \quad k, \quad \text{gdzie} \quad i,j,k = 0,\ldots,9.$$

Typ γ. Są to symboliczne numery segmentów.

Uwaga: Numerujemy tylko te segmenty, do których przekazujemy sterowanie, bądź te, które przekształcamy w trakcie prowadzenia obliczeń, przy czym należy podkreślić, że segmentów modyfikowanych nie numerujemy. Numery segmentów kodujemy za pomocą symboli trzyznakowych, z których pierwszy jest gwiazdką, pozostałymi są dwie cyfry dziesiętne:

$$\star \quad i \quad j, \quad \text{gdzie} \quad i,j = 0,\ldots,9.$$

Typ δ. Są to symboliczne zapisy zmiennych. Zmienne są kodowane za pomocą pary liter, litery V i dowolnej litery

$$\mathrm{V} \quad \square,$$

np. VE i VD itp.

Typ ε. Adresy rejestrów modyfikacji. Symbole te składają się z litery M i jednej cyfry dziesiętnej.

$$\mathrm{M} \quad i, \quad \text{gdzie} \quad i = 0,\ldots 9,$$

Typ ζ. Trzycyfrowe dziesiętne liczby całkowite ze znakiem np.

$$+\,001, \quad -\,097, \quad +\,101,\ldots$$

Typ η. Symbole początku i końca cyklu. Kodowane odpowiednio za pomocą znaku otwarcia i zamknięcia nawiasu.

Programy w kodzie zewnętrznym zapisujemy na specjalnych formularzach (tabl. 7-1).

Tablica 7-1

Numer segmentu	Początek cyklu	Segment	Koniec cyklu	Modyfikacja segmentu		
1	2	3	4	5	6	7

Podamy obecnie elementarne przykłady korzystania z kodu zewnętrznego. Powiększenie zawartości modyfikatora np. piątego realizujemy jak w tabl. 7-2. Pomnożenie zawartości dwóch komórek pamięci, np. 108 i 271 umieszczeniem wyników w trzeciej, np. 902 realizujemy jak w tabl. 7-3.

Tablica 7-2

Numer segmentu	Początek cyklu	Segment	Koniec cyklu	Modyfikacja segmentu		
1	2	3	4	5	6	7
		INA +001 M5 STO				

Tablica 7-3

Numer segmentu	Początek cyklu	Segment	Koniec cyklu	Modyfikacja segmentu		
1	2	3	4	5	6	7
		MUL 108 271 902 STO				

Zsumowanie zawartości n kolejnych komórek pamięci, np. 126, z których pierwsza ma adres 273, i umieszczenie wyników w komórce 735 realizujemy jak w tabl. 7-4. Funkcję określoną wzorem (7-1) realizujemy jak w tabl. 7-5.

Tablica 7-4

Numer segmentu	Początek cyklu	Segment	Koniec cyklu	Modyfikacja segmentu		
1	2	3	4	5	6	7
	((	ZER 735 ZER M1 +126 ADD 273 735 735 INA +001 M1 STO	))	M1		

$$f(x,y)=\begin{cases} e^x, & \text{gdy } x=y \\ \dfrac{e^x \sin(x-y)}{x-y}, & \text{gdy } x \neq y, \end{cases} \tag{7-1}$$

zakładając, że $x = (000)$, $y = (001)$, zaś $f(x,y) = (002)$.

Tablica 7-5

Numer segmentu	Początek cyklu	Segment	Koniec cyklu	Modyfikacja segmentu		
1	2	3	4	5	6	7
		EXP				
		000				
		002				
		$\overline{\text{EQJ}}$				
		000				
		001				
		*01				
		$\overline{\text{SUB}}$				
		000				
		001				
		003				
		$\overline{\text{DIV}}$				
		002				
		003				
		002				
		$\overline{\text{SIN}}$				
		003				
		003				
		$\overline{\text{MUL}}$				
		002				
		003				
		002				
*01		$\overline{\text{STO}}$				

Czytelnik skonstruuje z łatwością bardziej skomplikowane programy w omawianym kodzie zewnętrznym. Obecnie przejdziemy do omówienia metody zapisu w kodzie zewnętrznym podprogramów, przeznaczonych do wielokrotnego wywołania. Podprogramy takie, zapisane w kodzie zewnętrznym będziemy nazywali blokami uogólnionymi. Przykład podany w tabl. 7-5 jest przykładem programu przeznaczonego do jednorazowego wykonania. W przypadku gdybyśmy ten program mieli wykonać wielokrotnie, musimy zapisać go w postaci bloku uogólnionego. W tym celu musimy symbolicznie oznaczyć podprogram, np. EXS, następnie zmiennej x przyporządkować symbol VB, zmiennej y symbol VC, znanej f zaś symbol VD. W tablicy 7-6 podana jest postać bloku uogólnionego, realizującego naszą funkcję.

Uwaga: Jak widać z przykładu podanego w tabl. 7-6, blok uogólniony zapisujemy podobnie do cyklu, z tym że w przypadku cyklu podajemy obok symbolu otwarcia cyklu

Tablica 7-6

Numer segmentu	Początek cyklu	Segment	Koniec cyklu	Modyfikacja segmentu		
1	2	3	4	5	6	7
	(	EXS				
		EXP				
		VB				
		VD				
		EQJ				
		VB				
		VC				
		*01				
		SUB				
		VB				
		VC				
		003				
		DIV				
		VD				
		003				
		VD				
		SIN				
		003				
		003				
		MUL				
		VD				
		003				
		VD				
*01		STO	)			

— krotność cyklu, w przypadku zaś bloku uogólnionego używamy wprawdzie też symboli „początek cyklu" i „koniec cyklu", ale obok symbolu początku cyklu podajemy trójliterową nazwę podprogramu.

Na zakończenie podamy jeszcze kilka uwag o programach tłumaczących nasz kod zewnętrzny. Program taki składa się z dwóch części:

1) służącej do wprowadzania programu w kodzie zewnętrznym do maszyny, zastąpienia numerów komórek i symboli zmiennych w programie rzeczywistymi adresami, według ustalonego słownika, przeliczenia stałych, zbudowania słownika nazw bloków uogólnionych itp.;

2) realizującej poprzez zainterpretowanie kolejnych symboli kodu zewnętrznego naszego programu.

7.2. KODY ZEWNĘTRZNE TYPU COMPILER

7.2.0. Pośród kilku współcześnie istniejących kodów zewnętrznych typu *compiler*, najbardziej znane są:

1) radziecki Programujący Program dla BESM,
2) amerykański FORTRAN dla maszyn IBM.

W dalszych rozważaniach ograniczymy się do omówienia zasad posługiwania się Programującym Programem dla BESM. Programujący Program dla BESM został opracowany w Centrum Obliczeniowym Akademii Nauk ZSRR w latach 1954—1955. Szczegółowa budowa tego programu jest omówiona w pracy A. P. Jerszowa (Bibliografia [7]).

7.2.1. Operatory arytmetyczne. W Programującym Programie operatorem arytmetycznym nazywamy jednokrotne obliczenia wykonane według skończonego ciągu formuł. Gdzie przez formułę rozumiemy następujący związek

$$F(x_1, \ldots, x_n, c_1, \ldots, c_m) = y, \tag{7-2}$$

w którym y oznacza symbol zmiennej, nazywanej wynikiem stosowania formuły, F zaś jest dowolną superpozycją dowolnych podstawowych operacji nad zmiennymi $x_1, \ldots, x_n$ i stałymi $c_1, \ldots, c_n$. „Zmienne" i „stałe" obejmujemy wspólną nazwą „wielkości".

Tablica 7-7

Operatory arytmetyczne programującego programu dla BESM

Lp.	Nazwa operacji	Symbol operacji	Uwagi
1	Dodawanie	$+$	
2	Odejmowanie	$-$	
3	Mnożenie	$\times$	
4	Dzielenie	$:$	
5	Kwadrat	x^2	
6	Sześcian	x^3	
7	Część całkowita liczby x	Ex	całkowicie wykonywane pojedynczymi rozkazami maszyny BESM
8	Moduł liczby x	mod x	
9	Zmienna cechy liczby x na n	$ИП_n x$	
10	Przesunięcie cyfrowej części liczby x o n pozycji	$\overleftarrow{n}x$	
11	Znak liczby x	sign x	
12	Oddzielenie cechy liczby x	$\downarrow x$	
13	Drukowanie (wyprowadzenie na zewnątrz) obliczonego wyrażenia F	$F, \Longrightarrow 0$	
14	Cotangens x	ctg x	obliczenie za pomocą podprogramów z pamięci stałej maszyny BESM
15	Tangens x	tg x	
16	Logarytm naturalny x	ln x	
17	Przeliczenie liczby x na postać dziesiętną	Выд x	
18	e^x	exp x	
19	Arcus sinus x	arcsin x	
20	Arcus tangens x	arctg x	
21	Sinus x	sin x	
22	Cosinus x	cos x	
23	Pierwiastek kwadratowy z x	$\sqrt{x}$	

Lewą stronę dowolnej formuły będziemy nazywali wyrażeniem. Programujący Program zakłada listę operacji podstawowych podanych w tabl. 7-7.

Operacje wykonywane nad dwiema zmiennymi x i y (np. $x+y$, $x:y$) nazywamy operacjami dwuargumentowymi, operacje zaś wykonywane na jednej zmiennej x, (np. Ex, $\ln x$, $\sin x$) nazywamy operacjami jednoargumentowymi.

Dla uniknięcia niejednoznaczności zapisu formuł np.

$$a+b=a,$$

która może być czytana zarówno „a dodaj do b i wynik umieść w a" (właściwe znaczenie powyższej formuły z punktu widzenia Programującego Programu), jak też „a plus b równa się a skąd b równa się zeru", będziemy zamiast znaku „=", używali znaku przesyłania „$=>$". Przy takim zapisie formuła

$$a+b=>a$$

określa jednoznacznie działanie.

Dla wszystkich operacji podstawowych musimy określić ich zakres działania. Jedną z metod określania zakresu działania operacji jest umieszczenie jej wyniku w nawiasach. W prawidłowo zapisanym wyrażeniu każdemu symbolowi otwarcia nawiasu odpowiada symbol zamknięcia nawiasu. Dla każdego symbolu operacji można więc znaleźć parę nawiasów, do których zawartości odnosi się dana operacja. Nawiasy te będziemy odpowiednio nazywali lewą i prawą granicą zakresu działania operacji. Niedogodnością takiego określenia zakresu jest nadmierna ilość informacji i mała przejrzystość wyrażeń. Z powyższych względów znacznie wygodniej określić jest zakres działania za pomocą reguł kolejności wykonywania poszczególnych operacji.

Taki sposób zapisu został przyjęty w Programującym Programie. W tym celu wszystkie podstawowe operacje dopuszczalne przez Programujący Program zostały podzielone na cztery klasy:

1) operacje obliczenia kwadratu i sześcianu liczby,
2) pozostałe operacje jednoargumentowe,
3) operacje mnożenia i dzielenia,
4) operacje dodawania i odejmowania.

Dla każdej z klas zostały określone reguły zapisu.

Reguła 1. Argumentem operacji pierwszej klasy są wyrażenia stojące z lewej strony od symbolu operacji aż do pierwszego występującego symbolu innej operacji drugiej, trzeciej albo czwartej klasy lub do lewej granicy zakresu działania operacji:

$$a^{2^3}+\sin b^2+c\times(b^2+a^{3^3})^2=>y \sim$$
$$((a^2)^3)+\sin(b^2)+c\times(((b^2)+((a^3)^3))^2)=>y.$$

Reguła 2. Argumentem operacji jednoargumentowej są wszystkie wyrażenia stojące z prawej strony od symbolu operacji aż do pierwszego symbolu operacji trzeciej albo czwartej klasy lub do prawej granicy zakresu działania operacji:

$$\sin b^2\times\cos\ln(a+b)^{2^3}=>y \sim$$
$$(\sin(b^2))\times(\cos(\ln((a+b)^2)^3))=>y.$$

Reguła 3. Lewym argumentem operacji trzeciej klasy są wszystkie wyrażenia, stojące po lewej stronie symbolu operacji aż do pierwszego symbolu operacji czwartej klasy lub do lewej granicy zakresu działania operacji. Prawym argumentem operacji trzeciej klasy są wszystkie wyrażenia, stojące po prawej stronie symbolu operacji aż do pierwszego znaku operacji trzeciej albo czwartej klasy lub do prawej granicy zakresu działania operacji:

$$a\times(b+c)\times d : e+\sin b^2\times c^3 ==> y \sim$$
$$(((a\times(b+c))\times d) : e)+(\sin b^2)\times(c^3) ==> y.$$

Reguła 4. Lewym argumentem operacji czwartej klasy są wyrażenia, stojące po lewej stronie symbolu operacji aż do lewej granicy zakresu działania operacji. Prawym argumentem operacji czwartej klasy są wyrażenia, stojące po prawej stronie symbolu operacji aż do pierwszego znaku operacji czwartej klasy lub do prawej granicy zakresu działania operacji:

$$a+(b-c)+d-e+(a\times b-c)\times d+f ==> y \sim$$
$$(((((a+(b-c))+d)-e)+((a\times b)-c)\times d)+f) ==> y.$$

Ponadto przyjęto następujące pomocnicze oznaczenia:
„(n" — n kolejnych symboli otwarcia nawiasu,
„) n" — n kolejnych symboli zamknięcia nawiasu,
„,==>" — znak równości (w sensie ==>) bez normalizacji wyniku.

W schemacie logicznym operator arytmetyczny przedstawiamy bądź za pomocą formuł, bądź za pomocą litery A z indeksem. W drugim przypadku formułę operatora zapisuje się oddzielnie i oznacza się literą A z przyjętym indeksem.

7.2.2. Predykaty. Programujący Program zakłada następującą postać predykatu. Dana jest zmienna x; w zależności od tego, na jakim odcinku prostej leży punkt o współrzędnej x, predykat przekaże sterowanie odpowiedniemu operatorowi. Operatory, którym przekazujemy sterowanie, numerujemy. Numer N operatora oznaczamy za pomocą specjalnego symbolu $\underset{N}{\llcorner}$, symbol ten stawiamy w schemacie logicznym programu przed wyróżnianym operatorem. Inaczej mówiąc wyrażenie $\underset{N}{\llcorner} A$ oznacza, że operatorowi A przyporządkowano numer N.

Omówimy bliżej strukturę predykatu. Niech $M_1, M_2, \ldots, M_k$ będą nieprzecinającymi się przedziałami i niech każdemu z przedziałów M_j odpowiada operator o numerze N_j, zbiorowi $\overline{M}$ zaś dopełnieniu sumy M przedziałów $M_1, M_2, \ldots, M_k$ odpowiada operator o numerze $\overline{N}$, oczywiście w przypadku gdy $\overline{M}$ jest zbiorem pustym, nie przyporządkowujemy temu zbiorowi żadnego operatora. Rodzaje przedziałów są przedstawione w tabl. 7-8

Informacje opisujące predykat P mają następującą postać:

1) podanie nazwy zmiennej x,

2) określenie przedziałów M_j ($j = 1,2, \ldots, k$) przez podanie współrzędnych końców a_j i b_j ($a_j \leqslant b_j$) i rodzaj przedziału (według klasyfikacji podanej w tabl. 7-8),

Tablica 7-8

Klasyfikacja przedziałów

Nazwa typu odcinka i jego postać geometryczna	Symboliczny zapis
lewa półprosta bez końca $\ldots \longrightarrow a$	$(-\infty, a)$
lewa półprosta z końcem $\ldots \longrightarrow\!\bullet\ a$	$(-\infty, a]$
prawa półprosta bez końca $a \longleftarrow \ldots$	(a, ∞)
prawa półprosta z końcem $a \bullet\!\longleftarrow \ldots$	$[a, \infty)$
przedział otwarty $a \longleftarrow\!\longrightarrow b$	(a, b)
przedział domknięty prawostronnie $a \longleftarrow\!\bullet\ b$	$(a, b]$
przedział domknięty lewostronnie $a \bullet\!\longrightarrow b$	$[a, b)$
przedział domknięty $a \bullet\!\longrightarrow\!\bullet\ b$	$[a, b]$

3) przyporządkowanie $M_j \rightarrow N_j$,

4) podanie numeru operatora $\overline{N}$.

W schemacie logicznym predykaty zapisujemy w sposób następujący:

$$P(x, \overline{N}, \overset{N_1}{\Gamma}\{a_1, b_1\}; \overset{N_2}{\Gamma}\{a_2, b_2\}; \ldots; \overset{N_k}{\Gamma}\{a_k, b_k\}). \tag{7-3}$$

Uwaga: Jeśli $\overline{M}$ jest zbiorem pustym, to symbol $\overline{N}$ opuszczamy. Nawiasy określające typ przedziałów oznaczamy jak w tabl. 7-8.

7.2.3. Niestandartowe operatory. W przypadku gdy pewne czynności maszyny trudno określić za pomocą operatorów arytmetycznych i predykatów, można określić te czynności za pomocą sekwencji rozkazów dla maszyny. Sekwencje takie nazywamy niestandartowymi operatorami i zapisujemy je bądź bezpośrednio w schemacie logicznym, bądź oznaczamy je literą H z indeksem. W drugim z przypadków sekwencję rozkazów niestandartowego operatora wypisujemy oddzielnie i oznaczamy tę sekwencję literą H z przyjętym indeksem. Rozkazy tych sekwencji zapisujemy w adresach względnych. W rozkazach skokowych do operatorów i predykatów schematu logicznego podajemy w części adresowej numery operatorów bądź predykatów, którym przekazujemy sterowanie. W przypadku gdy w programie występują rozkazy przekazania sterowania z jednego niestandartowego operatora do innego niestandartowego operatora,

konstruujemy specjalne operatory łączniki opatrzone odpowiednim numerem, przekazujące sterowanie wewnętrznym rozkazem niestandartowego operatora. Ponadto zamiast adresów wielkości podajemy ich symbole.

7.2.4. Cykle. Cyklem względem parametru i nazywamy część programu realizującą wielokrotne działanie danego operatora (lub grupy operatorów i predykatów wykonywanych kolejno) dla wszystkich wartości przyjmowanych przez parametr i oraz realizującą konieczne zmiany (od wartości parametru i) zmiennych adresów. Częścią roboczą cyklu nazywamy powtarzany wielokrotnie operator (lub grupę operatorów i predykatów). Wszystkie pozostałe rozkazy cyklu nazywamy rozkazami sterowania.

Cykl określamy za pomocą następujących informacji:

1) określenie granic cyklu, czyli grupy operatorów i predykatów, które są roboczą częścią cyklu,

2) wskazanie parametru, z którym związany jest dany cykl,

3) określenie rodzaju zmienności i granic zmienności parametru,

4) podanie związków zmiennych adresów z parametrami.

Informacje o granicach cyklu oraz wskazanie parametru cyklu i podajemy bezpośrednio w schemacie logicznym, ujmując operatory i predykaty będące częścią roboczą cyklu w nawiasy kwadratowe [ ], przy czym pod symbolem początku cyklu zapisujemy parametr i: $[_i$. Przy podaniu symbolu parametru należy pamiętać, że nie można używać tego samego symbolu parametru dla kilku cykli.

Przy określaniu granic cyklu należy pamiętać o następujących ograniczeniach:

1. Dowolne dwa cykle $[\,]_i$ oraz $[\,]_j$ mogą albo zawierać się jeden w drugim $[[\ ]]_{ij}$, albo nie mogą mieć wspólnych części $[\,]_i\ [\,]_j$.

W pierwszym z przypadków cykl względem parametru i nazywamy zewnętrznym, cykl zaś względem parametru j nazywamy wewnętrznym, parametr i zaś nazywamy starszym parametrem w stosunku do młodszego parametru j.

2. Cykl musi być *związany* w tym sensie, że wszystkie zmienne adresy zależne od parametru odnoszącego się do danego cyklu muszą należeć do roboczej części cyklu.

Pozostałe informacje opisujące cykl tworzą oddzielną część programu. Granice zmienności parametru określa się za pomocą pary wielkości i_{pocz}, i_{kon}, przy czym zakłada się, że przez i_{pocz} rozumiemy pierwszą wartość parametru, przy której cykl wykonujemy, przez i_{kon} zaś rozumiemy pierwszą wartość parametru, dla której cyklu już nie wykonujemy; i_{pocz} oraz i_{kon} mogą być stałymi, zmiennymi lub mogą być wynikami pewnych operatorów arytmetycznych.

7.2.5. Kolejność informacji dla Programu Programującego. Informacje o zagadnieniu zapisujemy w czterech listach.

1. Lista П — lista informacji dotycząca zmiennych adresów.
2. Lista P — lista informacji o parametrach.
3. Lista C — lista informacji o stałych i zmiennych.
4. Lista K — lista informacji o schemacie logicznym.

Przykład 7-1. Zadanie: 1. Obliczyć i wydrukować elementy macierzy 10 rzędu według wzorów

$$a_{ij} = \begin{cases} i-j, & i>j, \\ i-j+10, & i \leqslant j, \end{cases} \quad (i,j = 1, \ldots, 10). \tag{7-4}$$

2. Do otrzymanej w ten sposób macierzy znaleźć macierz odwrotną korzystając z następującego algorytmu:

a) krok początkowy, wyznaczenie macierzy $A^{(0)}$

$$A^{(0)} = A-E, \tag{7-5}$$

gdzie E macierz jednostkowa.

Następny krok wykonujemy $n = 10$ razy, dla $m = 1, \ldots, n$;

b) m-ty wiersz macierzy $A^{(m-1)}$ przesyłamy do dodatkowych n komórek, których zawartość będziemy oznaczali $d_1, \ldots, d_n$, na to miejsce zaś w macierzy $A^{(m-1)}$ wstawiamy m-ty wiersz macierzy jednostkowej. Otrzymujemy w ten sposób macierz $A^{(m-1)'}$;

c) z macierzy $A^{(m-1)'}$, tworzymy macierz $A^{(m)}$ korzystając z wzorów:

$$a_{ij}^{(m)} = a_{ij}^{(m)'} - \frac{a_{im}^{(m-1)'} d_j}{1+d_m}. \tag{7-6}$$

Macierz $A^{(n)} = X$, będzie macierzą odwrotną do macierzy A.

3. W czasie znajdowania macierzy odwrotnej będziemy prowadzili kontrolę obliczeń w następujący sposób:

a) sumując wyrazy macierzy A wierszami, otrzymamy wektor $\vec{b} = (b_1, \ldots, b_n)$;

$$b_k = \sum_{j=1}^{n} a_{kj} \quad (k = 1, \ldots, n); \tag{7-7}$$

b) sumując wyrazy macierzy X kolumnami, otrzymamy wektor $\vec{e} = (e_1, \ldots, e_n)$

$$e_k = \sum_{i=1}^{n} x_{ik} \quad (k = 1, \ldots, n); \tag{7-8}$$

c) obliczamy iloczyn skalarny wektorów $\vec{b}$ i $\vec{e}$:

$$R = \left(\vec{b}, \vec{e} \right), \tag{7-9}$$

który w przypadku prawidłowych obliczeń powinien być bliski n.

Przy układaniu schematu logicznego programu, musimy ustalić, z jakich operatorów składać się będzie nasz schemat. Poza tym należy określić części pamięci, jakie będą wykorzystane przy rozwiązywaniu zagadnienia, oraz określić zależność zmiennych adresów od parametrów.

Obliczenie elementów macierzy i wyprowadzanie na zewnątrz wykonamy za pomocą dwóch cykli względem parametrów α i β (parametr α odpowiada indeksowi wierszy, a parametr β odpowiada indeksowi kolumn macierzy). W tym cyklu znajdować się będzie predykat określający dla danych wartości α i β, według której części formuły (7-4) będziemy wyznaczali elementy $a_{\alpha\beta}$. Do tych cykli będzie można włączyć sumowa-

nie elementów macierzy wierszami. W celu zmniejszenia ilości zmiennych rozkazów w tych cyklach będzie wygodnie wszystkie czynności przy obliczaniu kolejnych elementów macierzy wykonywać w standartowej komórce a, następnie zaś przesłać wynik do odpowiedniej komórki pamięci.

Przed odwróceniem macierzy należy wykonać cykl względem parametru l odjęcia jedynek od elementów diagonalnych macierzy (wzór 7-5).

Odwracanie macierzy będzie miało postać cyklu względem parametru m, wewnątrz którego będzie znajdował się cykl względem parametru k, realizujący czynności punktu 2 b zadania, oraz dwa cykle względem parametrów i, j realizujące wzór (7-6).

Dalej umieścimy dwa cykle względem parametrów r, p, które zsumują macierz odwrotną kolumnami, a na końcu umieścimy cykl względem parametru s — obliczenia iloczynu skalarnego (7-9).

Ostatecznie informacje o zadaniu przyjmą postać.

1. Lista II (zmienne adresy):

a) część A (100 komórek)

$$< a_{\alpha\beta} > = 10 \cdot \alpha + 1 \cdot \beta,$$
$$< a_{ll} > = 11 \cdot l,$$
$$< a_{mk} > = 10 \cdot m + 1 \cdot k,$$
$$< a_{mm} > = 11 \cdot m,$$
$$< a_{im} > = 10 \cdot i + 1 \cdot m,$$
$$< a_{pr} > = 10 \cdot p + 1 \cdot r,$$
$$< a_{ij} > = 10 \cdot i + 1 \cdot j,$$

b) część b (10 komórek)

$$< b_{\alpha} > = 1 \cdot \alpha,$$
$$< b_{s} > = 1 \cdot s,$$

c) część d (10 komórek)

$$< d_{k} > = 1 \cdot k,$$
$$< d_{m} > = 1 \cdot m,$$
$$< d_{j} > = 1 \cdot j,$$

d) część e (10 komórek)

$$< e_{s} > = 1 \cdot s,$$
$$< e_{r} > = 1 \cdot r,$$

2. Lista P (parametry)

$$\beta : \beta_{\text{pocz}} = 0, \ \beta_{\text{kon}} = 10,$$
$$\alpha : \alpha_{\text{pocz}} = 0, \ \alpha_{\text{kon}} = 10,$$
$$l : l_{\text{pocz}} = 0, \ l_{\text{kon}} = 10,$$
$$k : k_{\text{pocz}} = 0, \ k_{\text{kon}} = 10,$$
$$j : j_{\text{pocz}} = 0, \ j_{\text{kon}} = 10,$$
$$i : i_{\text{pocz}} = 0, \ i_{\text{kon}} = 10,$$
$$m : m_{\text{pocz}} = 0, \ m_{\text{kon}} = 10,$$
$$p : p_{\text{pocz}} = 0, \ p_{\text{kon}} = 10,$$
$$r : r_{\text{pocz}} = 0, \ r_{\text{kon}} = 10,$$
$$s : s_{\text{pocz}} = 0, \ s_{\text{kon}} = 10.$$

3. Lista C (stałe i zmienne)

$$0,\ 10,\ 1,\ a,\ c,\ f,\ R.$$

4. Lista K (schemat logiczny)

$$[0 \Longrightarrow b\ \underset{\alpha}{[}\ \underset{\beta}{[}\alpha - \beta \Longrightarrow a;\ P(\beta,\ 0101;\ \overset{0102}{\urcorner}\ (-\infty, \alpha)),$$

$$\overset{0101}{\llcorner} a + 10 \Longrightarrow a \overset{0102}{\llcorner} \text{Выд } a, \Longrightarrow 0;\ b_\alpha + a \Longrightarrow b_\alpha;\ a \Longrightarrow a_{\alpha\beta}]]$$

$$[\ \underset{m}{[}\ \underset{k}{[}\ a_{mk} \Longrightarrow d_k;\ \ 0 \Longrightarrow a_{mk}];\ 1 \Longrightarrow a_{mm};\ d_m + 1 \Longrightarrow c;$$

$$\text{Выд } c \Longrightarrow 0\ \underset{i}{[}a_{im} : c \Longrightarrow f\ \underset{j}{[}\ a_{ij} - j \times d_j \Longrightarrow a_{ij}\ ]\ ]\ ]$$

$$\underset{r}{[}0 \Longrightarrow e_r\ \ \underset{p}{[}\ e_r + a_{pr} \Longrightarrow e_r]\ ];\ 0 \Longrightarrow R;\ \underset{s}{[}\ R + b_s \times e_s \Longrightarrow R\ ]$$

$$\text{Выд } R, \Longrightarrow 0;\ \text{Стоп};$$

(Przykład powyższy zaczerpnięto z pracy A.P. Jerszowa — Bibliografia [7]).