$d + 0010$		
1	30	$< 007777 >$
2	23	0003
3	25	0000
4	01	$d + 0010$

6.2. Technika kompilacyjna

6.2.0. Rozważania dotychczasowe dotyczyły programów wykonujących bezpośrednio obliczenia. Obecnie zajmiemy się inną klasą programów, mianowicie klasą programów przeznaczonych do przekładu wyrażeń jednego języka na wyrażenia drugiego języka. Czytelnik zauważy, że terminy „język" i „przekład" użyliśmy w sensie ogólniejszym niż zwykle używanym w mowie potocznej. Tak na przykład przez przekład z języka na język będziemy rozumieli zarówno przekład z języka angielskiego na rosyjski, jak też układanie programów dla UPMC realizujących z góry zadane wyrażenie algebraiczne (czyli tłumaczenie wyrażeń sformalizowanego języka opisującego formuły algebraiczne na kod wewnętrzny maszyny). Programy realizujące przekład wyrażeń jednego języka na drugi nazywamy programami kompilacyjnymi lub składającymi. Pierwsza z tych nazw wydaje się trafniejsza.

6.2.1. Podobnie jak zrobiliśmy to w punkcie 6.1, technikę kompilacyjną wyłożymy na możliwie najprostszym przykładzie, który jednak pozwoli wyjaśnić zasadnicze elementy tej techniki. Wprawdzie w dotychczasowych rozważaniach mieliśmy do czynienia z bardzo prostym programem kompilacyjnym, a mianowicie z podstawowym programem wprowadzającym (punkt 5.1), jednakże przykład ten jest zbyt prosty na to, żeby pozwolił wyłożyć zasadnicze elementy techniki kompilacyjnej.

Weźmy pod uwagę klasę wyrażeń arytmetycznych zawierających tylko działania dodawania i mnożenia. Na przykład do klasy tej należy wyrażenie:

$$(((a \cdot b + c) \cdot d + e + f \cdot g) \cdot h + (i \cdot j + k) \cdot l. \tag{6-5}$$

Jeżeli odrzucimy konwencję, że dodawanie wiąże słabiej argumenty niż mnożenie, oraz przyjmiemy, że wynik każdego działania arytmetycznego jest ujmowany w nawiasy, otrzymamy wtedy tzw. pełną symbolikę nawiasową zapisu wyrażeń arytmetycznych. Przyjmiemy ponadto, że na zakończenie wyrażenia postawimy znak równości. Przy tak przyjętych regułach zapisu wyrażeń arytmetycznych wyrażenie (6-5) przyjmie postać:

$$(((((((a \cdot b) + c) \cdot d) + e) + (f \cdot g)) \cdot h) + (((i \cdot j) + k) \cdot l)) = . \tag{6-6}$$

Sformalizujemy obecnie proces zaprogramowania i zakodowania wyrażeń arytmetycznych dla dwu działań zmiennoprzecinkowych: dodawania i mnożenia, zapisanych w omawianej wyżej symbolice. Formalizacji tej dokonamy na drodze podania reguł postępowania, czyli dyrektyw przy budowaniu programu realizującego wyrażenia arytmetyczne w przyjętej symbolice.

Wprowadzimy obecnie pewne pomocnicze definicje.

Definicja 1. Rozważane wyrażenia arytmetyczne składają się z pięciu rodzajów symboli: symboli otwarcia nawiasu oznaczonych znakiem (, symboli zamknięcia nawiasu oznaczonych znakiem), symboli końca oznaczonych znakiem =, symboli zmiennych oznaczonych znakiem □, symboli operacji oznaczonych ○.

Definicja 2. Proces formalnego programowania wyrażeń arytmetycznych rozbijamy na takty. W każdym takcie rozpatrywanych jest n kolejnych symboli, gdzie $n = 1,2,\ldots$ Symbole rozpatrywane w każdym takcie numerujemy w kierunku od lewej strony do prawej, począwszy od symbolu numer jeden. Przez symbol numer zero k-tego taktu będziemy rozumieli ostatni symbol taktu $(k-1)$-ego.

Definicja 3. Przez kierunek normalny badania symboli będziemy rozumieli kierunek od lewej strony do prawej. Przez kierunek odwrotny będziemy rozumieli kierunek od prawej strony do lewej.

Z kolei wprowadzimy następujące dyrektywy:

Dyrektywa 1. Pobierz pierwszy z symboli, jeśli jest to symbol otwarcia nawiasu, to postąp ponownie według dyrektywy 1; jeśli jest symbol zamknięcia nawiasu, to postąp według dyrektywy 6; jeśli jest to symbol końca, to postąp według dyrektywy 7. Jeśli jest to symbol operacji, to pobierz drugi symbol i zbadaj go, jeśli drugi symbol jest symbolem zmiennej, to postąp według dyrektywy 3, jeśli zaś drugi symbol jest symbolem otwarcia nawiasu, to postąp według dyrektywy 4. Jeśli pierwszy symbol jest symbolem zmiennej, to pobierz trzeci symbol i zbadaj go, jeśli trzeci symbol jest symbolem zmiennej, to postąp według dyrektywy 2, jeśli zaś trzeci symbol jest symbolem otwarcia nawiasu, to postąp według dyrektywy 5.

Omawiane w dyrektywie 1 postępowanie w zależności od wyników testowania można scharakteryzować za pomocą tabl. 6-20.

Dyrektywa 2. Programuj

$s+0$	12	$<\square>$	(adres pierwszego symbolu),
1	14	P[1]	
2	12	$<\square>$	(adres trzeciego symbolu),
3	04	„○“	(adres podprogramu odpowiadającego drugiemu symbolowi),

po czym postaw znaki wykorzystania, przy symbolach: zerowym, pierwszym, drugim, trzecim i czwartym. Następnie postąp według dyrektywy 1, począwszy od piątego symbolu.

Dyrektywa 3. Programuj

$s+0$	12	$<\square>$	(adres drugiego symbolu),
1	04	„○“	(adres podprogramu odpowiadającego pierwszemu symbolowi,)

po czym postaw znaki wykorzystania, przy symbolach pierwszym, drugim, trzecim. Po czym posuwając się w kierunku odwrotnym postaw znak wykorzystania przy pierw-

---

(1) Przez P oznaczyliśmy komórkę roboczą programu zmiennego przecinka (punkt 4.3).

Tablica 6-20

Kombinacje symboli i odpowiadające im dyrektywy

Symbol					Stosowana dyrektywa
0	1	2	3	4	
(	(				1
(	□	○	□	)	2
)	○	□	)		3
)	○	(			4
(	□	○	(		5
)	)				6
)	=				7

szym napotkanym symbolu otwarcia nawiasu. Następnie postąp według dyrektywy 1, począwszy od czwartego symbolu.

Dyrektywa 4. Programuj

$$s+0 \quad 14 \quad R_i,$$

przy czym weź niewykorzystaną komórkę roboczą o najniższym adresie. Następnie postąp według dyrektywy 1, począwszy od trzeciego symbolu.

Dyrektywa 5. Postąp według dyrektywy 1 począwszy od czwartego symbolu.

Dyrektywa 6. Przejrzyj w odwrotnej kolejności poprzednio opracowane symbole, aż natrafisz na pierwszy symbol operacji, przy którym nie ma symbolu wykorzystania, następnie sprawdź, czy symbol stojący bezpośrednio przed odszukanym symbolem operacji ma znak wykorzystania, jeśli nie ma znaku wykorzystania, to programuj

$s+0$	12	$<\square>$	(adres symbolu stojącego przed odszukanym symbolem operacji),
1	04	„○“	(odszukany symbol operacji),

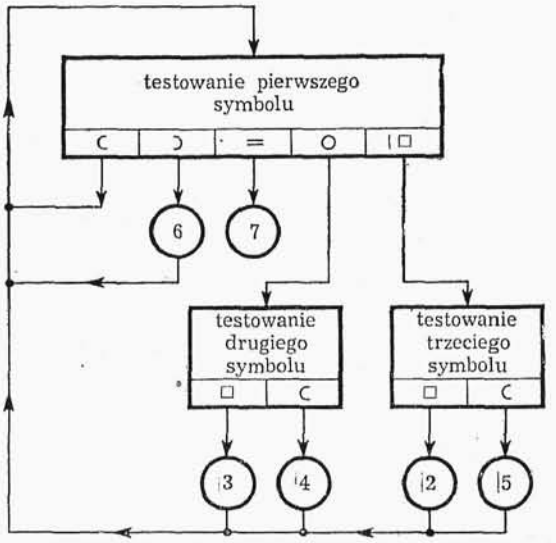
jeśli ma znak wykorzystania, to programuj

$s+0$	12	R_i	(odszukany adres komórki roboczej),
1	04	„○“	(odszukany symbol operacji),

po czym postaw znaki wykorzystania, przy symbolu pierwszym, przy odszukanym symbolu operacji, w pierwszym przypadku, ponadto przy symbolu zmiennej, stojącym bezpośrednio przed symbolem operacji. Po czym posuwając się dalej w kierunku przeciwnym postaw znaki wykorzystania przy pierwszym napotkanym symbolu otwarcia nawiasu, który nie miał znaku wykorzystania. Następnie postąp według dyrektywy 1, począwszy od drugiego symbolu.

Dyrektywa 7. Przerwij postępowanie.

Dla pełniejszego wyjaśnienia przypadków stosowania poszczególnych dyrektyw po dyrektywie 1 zapoznajemy się z wykresem przedstawionym na rys. 6-3.



Rys. 6-3. Schemat blokowy wyboru następnej dyrektywy w wyniku zastosowania dyrektywy 1

Zastosujemy obecnie dyrektywy $1 \div 7$ do wyrażenia (6-6).

1. Siedmiokrotnie stosujemy dyrektywę 1.
2. Stosujemy dyrektywę 2

$$\begin{array}{rll} s+0 & 12 & <a>\\ 1 & 14 & P,\\ 2 & 12 & <b>,\\ 3 & 04 & \text{„}\cdot\text{“}. \end{array}$$

3. Następnie stosujemy dyrektywę 3.

$$\begin{array}{rll} s+4 & 12 & <c>,\\ 5 & 04 & \text{„}+\text{“}. \end{array}$$

4. Następnie stosujemy powtórnie dyrektywę 3.

$$\begin{array}{rll} s+6 & 12 & <d>,\\ 7 & 04 & \text{„}\cdot\text{“}. \end{array}$$

5. Następnie stosujemy po raz trzeci dyrektywę 3.

$$\begin{array}{rll} s+10 & 12 & <e>,\\ 1 & 04 & \text{„}+\text{“}. \end{array}$$

6. Z kolei stosujemy dyrektywę 4

$$s+12 \quad 14 \quad R.$$

7. Następnie stosujemy dyrektywę 2

$$\begin{array}{rll} s+13 & 12 & <f>,\\ 4 & 14 & P,\\ 5 & 12 & <g>,\\ 6 & 06 & \text{„}\cdot\text{“}. \end{array}$$

Tablica 6-21

Numer symbolu	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20	21	22	23	24	25	26	27	30	31	32	33	34	35	36	37	40	41	42	43	44	45	46	47	50	51	52	53	54	55
Kolejne symbole	(	(	(	(	(	(	(	*a*	×	*b*	)	+	*c*	)	×	*d*	)	+	*e*	)	+	(	*f*	×	*g*	)	)	×	*h*	)	+	(	(	(	*i*	×	*j*	)	+	*k*	)	×	*b*	)	)	=
takt 1	→	→	→	→	→	→	→	→																																							
takt 2							*	*	*	*	*																																					
							→	→	→	→	→																																					
takt 3						*						*	*	*																																	
						←	←	←	←	←	←	→	→	→																																	
takt 4					*										*	*	*																															
					←	←	←	←	←	←	←	←	←	←	→	→	→																														
takt 5				*														*	*	*																											
				←	←	←	←	←	←	←	←	←	←	←	←	←	←	→	→	→																										
takt 6																					→	→																									
takt 7																						*	*	*	*	*																				
																							→	→	→	→																				
takt 8			*																		*						*																		
			←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	→																		
takt 9		*																										*	*	*																
		←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	→	→	→															
takt 10																															→	→	→	→												
takt 11																																	*	*	*	*	*									
																																		→	→	→	→									
takt 12																																	*					*	*	*						
																																	←	←	←	←	←	←	→	→	→					
takt 13																																*									*	*	*			
																																←	←	←	←	←	←	←	←	←	←	→	→	→		
takt 14	*																																											*		
	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	←	→		
takt 15																																														→

Objaśnienia:

→ znak badania symbolu przy postępowaniu w kierunku normalnym

← znak badania symbolu przy postępowaniu w kierunku odwrotnym

* znak wykorzystania symbolu

Tablica 6-22

Podprogram sformułowany przy korzystaniu z dyrektyw 1÷7

$s+0000$	12	$<a>$	$s+0020$	04	„+“
1	14	P	1	12	$<h>$
2	12	$<b>$	2	04	„·“
3	04	„·“	3	14	R
4	12	$<c>$	4	12	$<i>$
5	04	„+“	5	14	P
6	12	$<d>$	6	12	$<j>$
7	04	„·“	7	04	„·“
$s+0010$	12	$<e>$	$s+0030$	12	$<k>$
1	04	„+“	1	04	„+“
2	14	R	2	12	$<l>$
3	12	$<f>$	3	04	„·“
4	14	P	4	12	R
5	12	$<g>$	5	04	„+“
6	04	„·“			
7	12	R			

8. Następnie stosujemy dyrektywę 6

$s+17$ 12 R,
20 04 „+“.

9. Z kolei zastosujemy dyrektywę 3

$s+21$ 12 $<h>$,
2 04 „·“.

10. Ponownie stosujemy dyrektywę 4

$s+23$ 14 R.

11. Następnie stosujemy dyrektywę 2

$s+24$ 12 $<i>$,
5 14 P,
6 12 $<j>$,
7 04 „·“.

12. Następnie stosujemy dyrektywę 3

$s+30$ 12 $<k>$,
1 04 „+“.

13. Znowu stosujemy dyrektywę 3

$s+32$ 12 $<l>$,
3 04 „·“.

14. Następnie stosujemy dyrektywę 6

$$s + 34 \quad 12 \quad R,$$
$$5 \quad 04 \quad „+“.$$

15. Na zakończenie stosujemy dyrektywę 7.

W tablicy 6-21 przedstawiliśmy kolejne czynności wykonywane przy tworzeniu programu przy korzystaniu dyrektyw. W tablicy 6-22 podany jest utworzony program zapisany w adresach symbolicznych.

Zajmiemy się obecnie opracowaniem programu realizującego podane przez nas dyrektywy. Program ten będzie układał programy obliczenia wyrażeń arytmetycznych (postaci podanej wyżej) o zmiennym przecinku.

6.2.2. Kodowanie symboli. Pojedyncze symbole formuł arytmetycznych będziemy kodowali w 17B słowach. W dalszym ciągu przyjmiemy, że dysponujemy maksymalnie 32 zmiennymi, oznaczonymi kolejnymi literami alfabetu (przyjęty kod nie ma nic wspólnego z kodem typowej UPMC i jest wprowadzony jedynie na użytek niniejszego punktu), ponadto zakładamy, że opracowane przez nas formuły arytmetyczne wymagają w obliczeniach nie więcej niż 16 komórek roboczych. W słowie 17B przyporządkowujemy poszczególnym bitom części w następujący sposób:

α_0 — miejsce na znak wykorzystania,
$\alpha_0 = 0$ — symbol niewykorzystany przy opracowaniu,
$\alpha_0 = 1$ — symbol wykorzystany;
$\alpha_1, \alpha_2, \alpha_3, \alpha_4$ — rodzaj symbolu (tabl. 6-23)

Tablica 6-23

Kodowanie symboli

α_1	α_2	α_3	α_4	Rodzaj symbolu
1	0	0	0	(
0	1	0	0	)
0	0	1	0	=
0	0	0	1	○
0	0	0	0	□

Dla symboli „(“, „)“, „=“ bitów $\alpha_5 \div \alpha_{16}$ nie wykorzystujemy. Do symbolu "○„ wykorzystujemy bit α_5 dla określenia rodzaju operacji: $\alpha_5 = 0$ dla dodawania, $\alpha_5 = 1$ dla mnożenia, bity zaś $\alpha_6 \div \alpha_9$ są przeznaczone dla zapisania numeru komórki roboczej, w przypadku korzystania z komórki. Bitów $\alpha_{10} \div \alpha_{16}$ nie wykorzystujemy.

Dla symbolu □ bity $\alpha_5 \div \alpha_9$ są przeznaczone dla zapamiętania numeru (symbolu) zmiennej.

Wprowadzimy następujące oznaczenia:

N — bieżący adres symbolu opracowywanego wyrażenia arytmetycznego,
M — pomocniczy adres symbolu opracowywanego wyrażenia arytmetycznego,
s — kolejny adres układanego programu realizującego zadane wyrażenie,
β — 16-bitowa skala logiczna wykorzystania komórek roboczych. Inaczej mówiąc,

Tablica 6-24

Operator lub predykat	Kolejny adres	Kolejny rozkaz		Uwagi
1	a+0000	12	$<14\ s_0>$	
	1	14	$<14\ s>$	
2	2	12	$<12\ N_0>$	
	3	14	a+0006	$(a+0006) = 12\ N$
3	4	12	0077	$(0077) = 0$
	5	14	$<\beta>$	
4	6	12	N	← (G)
	7	23	0001	
	a+0010	03	a+0025	skok, gdy badamy symbol „(“
	1	23	0001	
	2	03	a+0031	skok, gdy badamy symbol „)“ → (A)
	3	23	0001	
	4	03	a+0177	skok, gdy badamy symbol „=“
	5	23	0001	
	6	03	a+0075	skok, gdy badamy symbol „○“
8	7	12	a+0006	$(a+0006) = 12N$
	a+0020	10	0067	$(0067) = 00\ 0002$
	1	14	a+0006	
9	2	00	a+0006	$(N) \rightarrow A$
	3	23	0001	skok, gdy badamy symbol „□“
	4	06	a+0154	→ (D) → (E)
5	5	12	a+0006	← (E) $(a+0006) = 12\ N$
	6	10	0076	$(0076) = 00\ 0001$
	7	14	a+0006	
	a+0030	02	a+0006	
10	1	04	k+0000	← (A) wywołanie podprogramu 4
11	2	04	h+0000	wywołanie podprogramu 1
12	3	04	t+0000	wywołanie podprogramu 6
13	4	12	$<14\ s>$	
	5	11	0076	$(0076) = 00\ 0001$
	6	14	$<14\ s>$	
14	7	12	k+0004	$12M \rightarrow A$
	a+0040	11	0076	$(0076) = 00\ 0001$
	1	14	a+0042	
	2	12	M	
	3	23	0000	
	4	03	a+0055	
20	5	04	l+0000	wywołanie podprogramu 2
21	6	04	v+0000	wywołanie podprogramu 5

Tablica 6-24 (d. c.)

Operator lub predykat	Kolejny adres	Kolejny rozkaz		Uwagi
22	$a+0047$	04	$r+0000$	wywołanie podprogramu 3
23	$a+0050$	04	$h+0000$	wywołanie podprogramu 1
24	1	12	$a+0006$	$(a+0006) = 12\ N$
	2	14	$k+0004$	$(k+0004) = 12\ M$
25	3	04	$r+0000$	wywołanie podprogramu 3
	4	02	$a+0025$	
15	5	00	$k+0004$	
	6	30	$<00\ 3600>$	
	7	22	0006	
	$a+0060$	10	$<12\ R_0>$	$R_i = R_0 + 2i$
16	1	00	$<14\ s>$	
17	2	04	$h+0000$	wywołanie podprogramu 1
18	3	00	$k+0004$	
	4	30	$<00\ 3600>$	
	5	10	$<21\ 0000>$	
	6	14	$a+0070$	
	7	12	$<20\ 0000>$	
	$a+0070$	21	0000	
	1	10	$<\beta>$	
	2	14	$<\beta>$	
19	3	04	$k+0000$	wywołanie podprogramu 4
	4	02	$a+0047$	
6	5	12	$a+0006$	$a+0006 = 12N$
	6	10	0076	$(0076) = 00\ 0001$
	7	14	$a+0006$	
7	$a+0100$	00	$a+0006$	
	1	23	0002	skok przy „(“ → C
	2	03	$a+0122$	
26	3	12	$a+0006$	$(a+0006) = 12\ N$
	4	14	$k+0004$	$(k+0004) = 12\ M$
27	5	04	$v+0000$	wywołanie podprogramu 5
28	6	04	$t+0000$	wywołanie podprogramu 6
29	7	12	$k+0004$	
	$a+0110$	10	0067	$(0067) = 00\ 0002$
	1	14	$k+0004$	
30	2	04	$r+0000$	wywołanie podprogramu 3
31	3	04	$k+0000$	wywołanie podprogramu 4

Tablica 6-24 (d. c.)

Operator lub predykat	Kolejny adres	Kolejny rozkaz		Uwagi
32	$a+0114$	04	$r+0000$	wywołanie podprogramu 3
33	5	04	$h+0000$	← (H) wywołanie podprogramu 1
34	6	12	$a+0006$	$(a+0006) = 12\ N$
	7	10	0067	$(0067) = 00\ 0002$
	$a+0120$	14	$a+0006$	
	1	02	$a+0006$	→ (G)
35	2	12	<22 0000>	← (C)
	3	14	$a+0125$	
36	4	12	$<\beta>$	
37	5	22	0000	
38	6	06	$a+0135$	
42	7	12	$a+0125$	
	$a+0130$	10	0076	$(0076) = 00\ 0001$
	1	14	$a+0125$	
43	2	11	<22 0020>	
	3	03	$a+0124$	
44	4	04	< 🔔 >	wywołanie podprogramu dzwonka
	5	12	$a+0125$	$(a+0125) = 22\,i$
	6	23	0001	$(A) = 04\ 2\,i$
	7	10	$<10\ R_0>$	$+\ \frac{10\ R_0}{14\ R_i}$
40	$a+0140$	00	$<14\ s>$	
41	1	12	$a+0006$	$(a+0006) = 12\ N$
	2	11	<02 0001>	
	3	14	$a+0151$	
	4	10	<02 0000>	
	5	14	$a+0152$	
	6	12	$a+0125$	
	7	30	<00 0017>	7B
	$a+0150$	23	<00 0007>	$\alpha_6\alpha_7\alpha_8\alpha_9$ $\boxed{\alpha_{10}\alpha_{11}\alpha_{12}}$ $\alpha_{13}\alpha_{14}\alpha_{15}\alpha_{16}$
	1	10	N-1	←
	2	14	N-1	→ (H)
	3	02	$a+0115$	
45	4	12	$a+0006$	$(a+0006) = 12N$
	5	10	0076	← (D) $(0076) = 00\ 0001$
	6	14	$k+0004$	$(k+0004) = 12M$
46	7	04	$r+0004$	wywołanie podprogramu 3

Tablica 6-24 (d. c.)

Operator lub predykat	Kolejny adres	Kolejny rozkaz		Uwagi
47	a+0160	04	l+0000	wywołanie podprogramu 1
48	1	04	v+0000	wywołanie podprogramu 5
49	2	12	<14 P>	
	3	00	<14 s>	
50	4	12	<14 s>	
	5	10	0067	(0067) = 00 0002
	6	14	<14 s>	
51	7	04	z+0000	wywołanie podprogramu 6
52	a+0170	12	<14 s>	
	1	11	0076	(0076) = 00 0001
	2	14	<14 s>	
53	3	04	l+0000	wywołanie podprogramu 2
54	4	04	v+0000	wywołanie podprogramu 5
55	5	04	r+0000	wywołanie podprogramu 3 → (H)
	6	02	a+0114	
	7	05	0000	stop

jeśli w i-tej komórce roboczej jest przechowywany wynik pośredni przeznaczony do dalszych obliczeń, to na i-tym miejscu skali logicznej β znajduje się jedynka.

Na rysunku 6-4 przedstawiony jest schemat blokowy naszego programu. Jak widać z tego schematu, wielokrotnie powtarzają się w nim następujące podprogramy:

1. Powiększenie zawartości $< 14\,s >$ o jeden

h + 0000		
1	12	$< 14\,s >$,
2	10	$< 00\,0001 >$,
3	14	$< 14\,s >$,
4	01	h + 0000.

2. Zmniejszenie zawartości $< 12\,M >$ o jeden

l + 0000		
1	12	$< 12\,M >$,
2	11	$< 00\,0001 >$,
3	14	$< 12\,M >$,
4	01	l + 0000.

3. Postawienie znaku wykorzystania, przy symbolu zapisanym pod adresem M

$r + 0000$		
1	12	$< 12\,N >$,
2	10	$< 02\,0000 >$,
3	14	$r + 0006$,
4	00	$< 12\,N >$,
5	10	$< 20\,0000 >$,
6	14	M,
7	01	$r + 0000$.

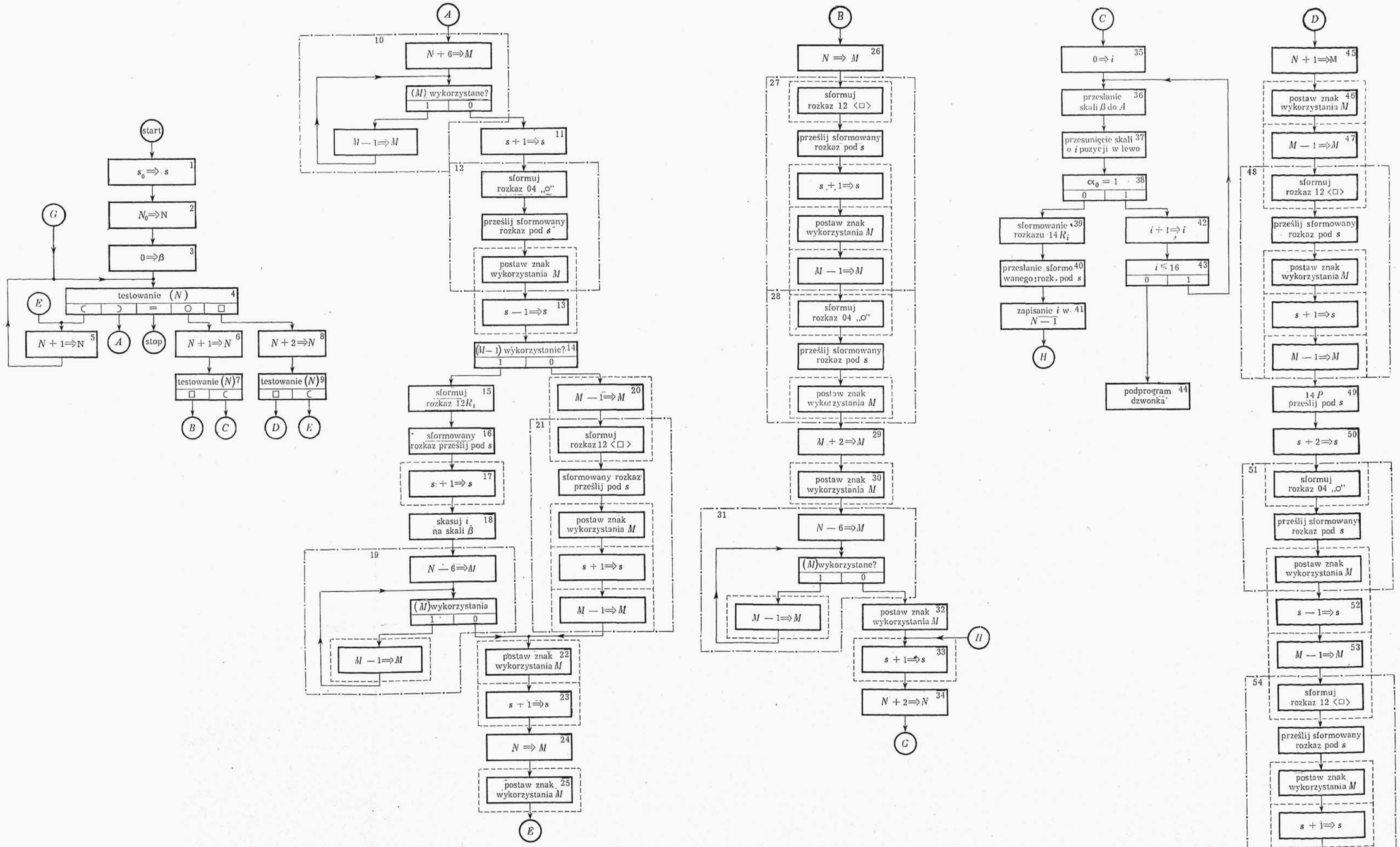
4. Szukanie w kierunku malejącym adresów pierwszego symbolu, w którym nie ma znaku wykorzystania, począwszy od N-6

$k + 0000$		
1	12	$< 12\,N >$,
2	11	$< 00\,0006 >$,
3	14	$k + 0004$,
4	12	M,
5	20	0000,
6	03	$k + 0011$,
7	04	$l + 0000$,
$k + 0010$	02	$k + 0004$,
1	01	$k + 0000$.

5. Formowanie rozkazu 12 $< \square >$, gdzie symbol $\square$ zapisany jest w komórce o adresie M, sformowany rozkaz zostaje przesłany pod adres s, przy symbolu zapisanym pod adresem M postawiony zostaje symbol wykorzystania, zawartość $< 12\,M >$ zostaje zmniejszona o jeden.

$v + 0000$		
1	00	$< 12\,M >$,
2	30	$< 00\,7600 >$,
3	22	0006,
4	10	$< 12 <a >>$,
5	00	$< 14\,s >$,
6	04	$r + 0000$,
7	04	$h + 0000$,
$v + 0010$	04	$l + 0000$,
1	01	$v + 0000$.

6. Formowanie rozkazu 04 „○“, gdzie symbol „○“ zapisany jest w komórce



Rys. 6-4. Schemat blokowy programu kodującego uproszczone formuły arytmetyczne

o adresie M, sformowany rozkaz zostaje przesłany pod adres s, przy symbolu zapisanym pod adresem M podstawiony zostaje symbol wykorzystania,

$t + 0000$		
1	00	$< 12\ M >$,
2	30	$< 00\ 4000 >$,
3	22	0013,
4	10	$< 12 < 04$ „+“ $>>$,
5	14	$t + 0006$
6	12	04 „○“
7	00	$< 14\ s >$,
$t + 0010$	04	$r + 0000$,
1	01	$t + 0000$.

Powyższy podprogram został ułożony, przy założeniu, że rozkazy 04 „+“ i 04 „·“ są zapamiętane w kolejnych krótkich komórkach pamięci.

W tablicy 6-24 przedstawione są kolejne rozkazy programu głównego, działającego w myśl przyjętych założeń.

6.3. GENEROWANIE LICZB PSEUDOLOSOWYCH I ZMIENNYCH PSEUDOLOSOWYCH

Omówione w punkcie 6.5 metody modelowania układów dynamicznych wymagają niejednokrotnie statystycznego badania tych ostatnich, ze względu na losowe zmiany stanów pewnych (nie sterowanych) wejść tych układów. Niniejszy punkt ma charakter pomocniczy.

6.3.0. Liczby losowe a zmienne losowe. Przez ciągi k-cyfrowych liczb losowych rozumiemy takie i tylko takie ciągi, dla których, jeśli weźmiemy dostatecznie dużo wyrazów, wystąpienie każdej k-cyfrowej liczby jest równie prawdopodobne. Ponadto w ciągu liczb losowych nie mogą wystąpić żadne inne regularności. Przez ciągi k-cyfrowych zmiennych losowych rozumiemy takie i tylko takie ciągi, dla których, jeśli weźmiemy dostatecznie dużo wyrazów, prawdopodobieństwo wystąpienia każdej k-cyfrowej liczby losowej jest określone przez całkę z funkcji, dla której wyżej wspomniany ciąg jest zbiorem argumentów. Funkcję tę będziemy nazywali gęstością prawdopodobieństwa. Granica całkowania przebiega od minimum zbioru argumentów do badanej liczby. Liczby losowe są to zmienne losowe, dla których funkcja rozkładu prawdopodobieństwa ma wartość stałą.

6.3.1. Liczby pseudolosowe. Przez ciągi k-cyfrowych liczb pseudolosowych będziemy rozumieli ciągi o stałej funkcji rozkładu, dla których jednak istnieje stała naturalna T, będąca okresem ciągu liczb k-cyfrowych.

W praktyce obliczeniowej zastępujemy ciąg liczb losowych ciągami pseudolosowymi. Metody generowania liczb pseudolosowych rozwinęły się w związku z powstaniem tzw. metody Monte Carlo (metody modeli stochastycznych). Zastosowanie tej ostatniej do rozwiązania zagadnień na UPMC wymagało opracowania metod generowania liczb

pseudolosowych przy użyciu bądź formuł arytmetycznych, bądź kombinatorycznych. Omówimy obecnie kilka metod generowania liczb pseudolosowych.

6.3.2. Metody addytywne. Wspólnym dla wszystkich metod addytywnych jest generowanie ciągów liczb pseudolosowych przez dodawanie pewnych wielkości. Jedną z metod addytywnych jest metoda zredukowanego ciągu Fibonacciego, określoną następującą formułą rekurencyjną:

$$a_0 = 0, \quad a_1 = 1; \quad a_{n+2} \equiv (a_{n+1} + a_n) \pmod{m}. \tag{6-7}$$

Utworzony według formuły (6-7) ciąg jest ciągiem pseudolosowym, którego okres zależy od modułu m. Na przykład dla $m = 2^{44}$ okres $T = 3 \cdot 2^{43} \approx 2{,}5 \cdot 10^{13}$. (Badania powyższe przeprowadzono na maszynie SEAC). Jak widać z przytoczonego wyżej przykładu, ciągi otrzymywane przy użyciu formuły (6-7) nie muszą spełniać podanej przez nas definicji ciągu liczb pseudolosowych, ponieważ rozkład w tak otrzymanym ciągu nie jest równomierny.

Znacznie lepsze metody daje inna metoda addytywna. Danych jest k-ciągów $\{a_n^i\}$ (gdzie $i = 1,2, \ldots, k$), z których każdy ma okres T_i. Jeśli liczby $T_1, T_2, \ldots, T_k$ są względem siebie pierwsze, to sumując między sobą odpowiednie wyrazy ciągów a_n^i otrzymamy ciąg o okresie $T = \prod_{i=1}^{k} T_i$. Oczywiście, że na to aby wypadkowy ciąg był ciągiem liczb pseudolosowych, należy odpowiednio dobrać ciągi składowe. Metoda ta w dość prosty sposób daje się zaprogramować na UPMC. Jedyną jej wadą jest fakt, że dla otrzymania ciągu o stosunkowo dużym okresie trzeba zająć dość dużo miejsca w pamięci.

6.3.3. Metody multiplikatywne. Spośród metod multiplikatywnych omówimy jedną, tzw. metodę Lehmmera. Metoda ta opiera się na następującej własności. Jeśli liczba a jest pierwiastkiem pierwotnym liczby pierwszej p (tzn. że dla każdego dodatniego $x < p-1$ nie zachodzi kongruencja $a^x = 1 \pmod{p}$), to ciąg $\{u_n\}$, określony rekurencyjnym wzorem

$$\left.\begin{aligned} u_n &\equiv a u_{n-1} \pmod{p}, \\ u_0 &< p, \end{aligned}\right\} \tag{6-8}$$

jest ciągiem okresowym o okresie $T = p-1$ zawierającym wszystkie liczby $1+p-1$, tzw. układ zupełny reszt potęgowych mod p.

Dobre wyniki można uzyskać stosując metodę Lehmmera w połączeniu z drugą z przedstawionych metod addytywnych. Mianowicie ciągi $\{a_n^i\}$ generujemy stosując niezależnie metodę Lehmmera dla różnych wartości p, po czym sumując otrzymane w ten sposób liczby mod m. Należy podkreślić, że dla otrzymania ciągu liczb pseudolosowych (w myśl przyjętej przez nas definicji) trzeba starannie dobrać ciągi $\{a_n^i\}$, ponieważ w zależności od doboru tych ciągów otrzymamy różne rozkłady liczb w ciągu $\{a_n\}$.

6.3.4. Liczby tasowane. H. Steinhaus opracował metodę otrzymywania ciągów liczb pseudolosowych poprzez tasowanie kolejnych n liczb. Otrzymane przez H. Steinhausa liczby tasowane są liczbami pseudolosowymi w myśl przyjętej przez nas definicji. Metoda ta jednak jest nieprzydatna dla stosowania na UPMC, ze względu na konieczność dysponowania wszystkimi kolejnymi liczbami, które występują w generowanym ciągu.

Uogólnieniem metody zaproponowanej przez H. Steinhausa jest metoda generowania liczb pseudolosowych poprzez tasowanie cyfr. Założenie tej metody jest następujące: daną jest macierz o n wierszach i k kolumnach, której elementami są cyfry $0,1 \ldots, g-1$; gdzie g jest podstawą rozwinięcia liczb. Wiersze tej macierzy traktujemy jako k-cyfrowe liczby w rozwinięciu przy podstawie g. Nasza macierz jest więc układem n k-cyfrowych liczb. Przyjmiemy, że n jest całkowitą wielokrotnością podstawy rozwinięcia naszych liczb g oraz że każda cyfra $0,1, \ldots, g-1$ wystąpi w naszej macierzy $\frac{n}{g}$ k razy. Oznaczmy elementy naszej macierzy symbolami a_{pq}, gdzie $p = 0,1, \ldots, n-1$; $q = 0,1, \ldots, k-1$. Macierz a_{pq} ma $(nk-1)$ — przekątnych, w dalszym ciągu będziemy zakładali, że $k<n$. Niech j będzie indeksem przekątnej ($j = 0,1 \ldots, (nk-2)$). Element a_{pq} leży na j-tej przekątnej wtedy i tylko wtedy, gdy indeksy p i q spełniają następujący związek :

$$p+q=j, \tag{6-9}$$

$$\min(j, n-1) \geqslant p \geqslant \max(0, j-k+1). \tag{6-10}$$

Ilość elementów j-tej przekątnej określamy za pomocą wzoru

$$l(j) = \min(j, k-1, n+k-j-2)+1. \tag{6-11}$$

Podstawą metody jest przekształcenie macierzy $||a_{pq}||$ w macierz $||a'_{pq}||$, polegające na przepisaniu począwszy od s-tej przekątnej macierzy $||a_{pq}||$ w wiersze macierzy $||a'_{pq}||$, po czym po przekątnej $(nk-2)$ brana jest przekątna 0; jako ostatnia zostaje przepisana przekątna $(s-1)$. Przy kolejnych przekształceniach ciąg indeksów przekątnych, od których zaczynamy transformacje, generujemy na niezależnej drodze, na przykład stosując zredukowany ciąg Fibonacciego, dla $m = (nk-2)$. W zależności od tego, jak dobierzemy ciąg indeksów przekątnych, otrzymamy dłuższe lub krótsze ciągi k-cyfrowych liczb. Istnieje hipoteza, że dla n kilkakrotnie większych od k tak otrzymany ciąg liczb k-cyfrowych ma rozkład zbliżony do równomiernego. Hipoteza ta została udowodniona dla $g = k = 2$ (jako indeksów przekątnych użyto zredukowanego ciągu Fibonacciego). Powyższa metoda jest jeszcze mało znana, wydaje się jednak, że może ona dać bardzo dobre rezultaty. Dla $k = 5$ i $n = 10$ mamy $\frac{10!}{(5!)^{10}}$ różnych macierzy, gdyby więc udało się stworzyć ciąg, w którym każda macierz wystąpiłaby co najmniej jeden raz, otrzymalibyśmy ciąg pięciocyfrowych liczb o okresie rzędu 10^{45}. Oczywiście w tak otrzymanym ciągu należałoby starannie zbadać sprawę równomierności rozkładu.

6.3.5. Generowanie zmiennych losowych. Zakładając, że dysponujemy możliwością generowania liczb pseudolosowych, możemy korzystając z podanego dalej twierdzenia Wołkowa generować ciągi dowolnych zmiennych pseudolosowych o zadanej z góry gęstości prawdopodobieństwa. Metoda ta była z powodzeniem stosowana w Centrum Obliczeniowym A. N. ZSRR w Moskwie, przy użyciu maszyny BESM.

Twierdzenie Wołkowa. Założenia. Niech będą dane dwa ciągi liczb losowych $\{x_n\}$ i $\{y_n\}$, takie, że $a \leqslant x_n \leqslant b$ oraz $c \leqslant y_n \leqslant d$, oraz niech będzie dana funkcja $f(x)$ ciągła dla $a \leqslant x \leqslant b$, o wartościach należących do przedziału $<c, d>$. Utworzymy nowy

ciąg $\{z_m\}$, będący podciągiem ciągu $\{x_n\}$, zaliczając do ciągu $\{z_n\}$ te wyrazy ciągu $\{x_n\}$, dla których spełniony jest związek:

$$z_n = x_n \quad \text{wtedy i tylko wtedy, gdy } f(x_n) - y_n > 0. \tag{6-12}$$

Teza. Funkcja

$$\frac{1}{\int_a^b f(t)\,dt} f(x)$$

jest funkcją gęstości prawdopodobieństwa dla ciągu $\{z_n\}$.

Dowód powyższego twierdzenia jest natychmiastowy.

6.4. MODELOWANIE ODMIENNYCH ORGANIZACJI MASZYN CYFROWYCH

Omówiona w punkcie 6.1 technika interpretacyjna jest przydatna do modelowania na UPMC dowolnych kodów rozkazowych. Przypuśćmy, że projektując nową maszynę cyfrową chcemy stwierdzić użyteczność kodu rozkazowego tej maszyny. Dokonujemy tego na dwóch drogach:

1) przez ułożenie programów i podprogramów w kodzie budowanej maszyny,

2) poprzez sprawdzenie na posiadanej UPMC ułożonych programów i podprogramów.

Mogłyby się wydawać, że punkt 1) jest wystarczającą kontrolą. Jak wskazuje jednak doświadczenie, samo ułożenie programu nie wystarcza, dopiero po sprawdzeniu programu na maszynie można powiedzieć, że działa on rzeczywiście poprawnie.

6.5. MODELOWANIE UKŁADÓW DYNAMICZNYCH

Zdefiniujemy najpierw pojęcie układu dynamicznego.

Definicja 1. Przez układ dynamiczny będziemy rozumieli taki i tylko taki układ materialny, dla którego istnieje skończony układ funkcji względem czasu, opisujących jednoznacznie stan układu w danej chwili.

Należy podkreślić, że funkcje opisujące układ nie muszą być danę w jawnej postaci. Mogą to być np. rozwiązania pewnych równań różniczkowych względem czasu, z danymi warunkami początkowymi. Z układami dynamicznymi mamy do czynienia zarówno w automatyce, biologii, socjologii, jak i ekonomii. W dalszych rozważaniach ograniczymy się do układów ekonomicznych, ze względu na możliwość modelowania tych ostatnich na małych UPMC.

W modelach ekonomicznych będziemy rozróżniali następujące układy względnie odosobnione:

1) układy produkcyjne,
2) układy rozdzielcze,
3) układy opóźniające,
4) układy sterujące (planujące),