Część trzecia

MODELOWANIE CYFROWE

6. ELEMENTY MODELOWANIA CYFROWEGO

[6.0. Uwagi wstępne, 6.1. Technika interpretacyjna, 6.2. Technika kompilacyjna, 6.3. Generowanie liczb pseudolosowych i zmiennych pseudolosowych, 6.4. Modelowanie odmiennych organizacji maszyn cyfrowych, 6.5. Modelowanie układów dynamicznych, 6.6. Maszyny cyfrowe a cybernetyka, 6.7. Uwagi końcowe]

6.0. UWAGI WSTĘPNE

Dotychczasowe rozważania ograniczyliśmy bądź do zasad działania UPMC, bądź do bezpośredniego stosowania UPMC w problemach obliczeniowych. Obok zastosowań obliczeniowych istnieje jeszcze obszerna dziedzina zastosowań UPMC do stwierdzania własności pewnych procesów, przez badanie ich modeli. W związku z tym sprecyzujemy pojęcie modelu.

Definicja 1. Model obiektu A jest to fikcyjny lub rzeczywisty twór, będący uproszczeniem obiektu A, zachowujący jednak istotne (ze względu na dane badanie) cechy przedmiotu A.

Definicja 2. Mówimy, że A jest oryginałem B wtedy i tylko wtedy, jeżeli B jest modelem A.

Dział matematyki zajmujący się budowaniem matematycznych modeli zjawisk fizycznych, technicznych, biologicznych, psychologicznych, ekonomicznych i socjologicznych oraz układaniem programów dla UPMC realizujących te modele nazywamy modelowaniem cyfrowym. Podstawowym pojęciem w tym dziale jest pojęcie układu względnie odosobnionego.

Definicja 3. Przez układ względnie odosobniony będziemy rozumieli układ o n wejściach i m wyjściach, taki, że:

1) istnieją przedziały, o długości nie równej 0, zmienności parametrów opisujących jednoznacznie świat zewnętrzny naszego układu.

2) dla wartości parametrów z powyższych przedziałów stany wszystkich wejść układu w chwili obecnej i we wszystkich chwilach minionych jednoznacznie wyznaczają stan każdego z wyjść układu w chwili obecnej.

Przykładów, spełniających powyższą definicję 3, znajdzie czytelnik wiele wśród obiektów, z którymi styka się na codzień. Za przykład układu względnie odosobnionego

może służyć odbiornik radiowy włączony do sieci. Wejściami do naszego układu będzie antena, regulatory długości fali i natężenia głosu itp. Wyjściem z układu jest głośnik. Parametrem decydującym o tym, czy stan obecny wyjścia jest jednoznacznie wyznaczony przez obecne i minione stany wejść, jest napięcie zasilające. W przypadku zmiany napięcia zasilającego poza dopuszczalny przedział wahań, nasz odbiornik przestaje działać w sposób prawidłowy, czyli inaczej mówiąc związki między wejściami a wyjściami ulegają zmianie.

Dla dalszych rozważań wprowadzimy jeszcze pojęcie modelu o skali czasowej T.

Definicja 4. Będziemy mówili, że B jest modelem A o skali czasowej T, zamiast mówić, że B jest modelem A i procesy w modelu B przebiegają w skali $1 : T$ w stosunku do procesów przebiegających w oryginale A.

Budowanie modelu procesu A polega na rozbiciu tego procesu na pewne elementarne procesy zachodzące w pewnych układach względnie odosobnionych. Poszczególne układy względnie odosobnione opisujemy za pomocą formuł matematycznych bądź logicznych, np. za pomocą równań różniczkowych czy też różnicowych.

Łącząc wyjścia poszczególnych układów z wejściami innych (budujemy tzw. schemat logiczno-czasowy) otrzymujemy model procesu A. Następnie układamy programy realizujące poszczególne układy względnie odosobnione i związki pomiędzy układami.

6.1. TECHNIKA INTERPRETACYJNA

Rozszerzeniem metody podprogramów jest technika interpretacyjna. Przy programowaniu z użyciem podprogramów, program główny składa się z rozkazów bezpośrednio realizujących działania obliczeniowe i organizacyjne oraz z rozkazów służących do wywołania podprogramów. Dla wywołania podprogramu w większości UPMC potrzeba kilku rozkazów. Technika interpretacyjna rozwiązuje to zagadnienie w inny sposób. Mianowicie program główny składa się z rozkazów, z których każdy jest interpretowany jako rozkaz wywołania podprogramów, zapisanych w pewnym kodzie (niekoniecznie w kodzie danej UPMC), przy czym wywołanie dowolnego podprogramu odbywa się przy użyciu jednego rozkazu programu głównego. W odróżnieniu od omawianej w rozdz. 4 metody podprogramów, gdzie o rodzaju wykonywanej operacji decydowała część adresowa rozkazu wywołania odpowiedniego podprogramu, w technice interpretacyjnej rodzaj podprogramu określa nie część adresową rozkazu wywołania, lecz część operacyjną tego rozkazu. „Łącznikiem" między programem głównym a podprogramami jest tzw. program interpretowania, który wywołuje odpowiednie podprogramy odpowiadające kolejnym rozkazom programu głównego. Jeżeli program główny, zwany dalej programem interpretowanym, składa się z rozkazów jednoadresowych, to program interpretujący budujemy w sposób następujący: wyróżniamy miejsca pamięci (w naszym przypadku dwa adresy długie), które będą modelowały akumulator, następnie wyróżniamy jeden adres krótki, który będzie modelował licznik rozkazów, rejestr rozkazów modelujemy w akumulatorze. Program interpretujący działa następująco: kolejne rozkazy programu interpretowanego (programu głównego) zostają pobrane do akumulatora

odpowiednio przekształcone, po czym służą do wywołania odpowiedniego podprogramu. Po wykonaniu podprogramu, zostaje przeadresowany „licznik rozkazów" programu interpretującego, po czym na podstawie wskazań przeadresowanego „licznika rozkazów" zostaje pobrany następny rozkaz programu interpretowanego do akumulatora itd. Bardziej szczegółowo prześledzimy technikę interpretacyjną na stosunkowo prostym przykładzie.

Przykład 6-1(1). Przypuśćmy, że pewne obliczenia musimy wykonywać na liczbach zespolonych, takich, że ich część rzeczywista jest dana z dokładnością 17B i ich część urojona jest dana z dokładnością 17B. Część rzeczywista i urojona tych liczb należy do przedziału $<-1, 1-2^{-16}>$. Obliczenia te możemy przeprowadzić na dwóch drogach:

1) poprzez bezpośrednie zaprogramowanie działań zespolonych,

2) poprzez technikę interpretacyjną.

Omówimy szczegółowo punkt 2. Będziemy modelowali na naszej UPMC maszynę o podobnym kodzie rozkazowym, ale działania arytmetyczne tej modelowanej maszyny będą działaniami na 34B liczbach zespolonych. Bardziej znaczące 17 bitów będzie częścią rzeczywistą liczby zespolonej, mniej zaś znaczące 17B będzie częścią urojoną liczby zespolonej. Różnice w kodzie modelowanej maszyny w porównaniu do naszej UPMC (punkt 2.3.) są następujące:

1) warunek „W" — odpowiadający generowanemu przez UPMC warunkowi W — będzie dla wszystkich rozkazów modelowanej arytmetyki równy jedności, jeśli wynik działania jest różny od zera, a jest zerem, jeśli wynik działania jest zerem;

2) operacje mnożenia i dzielenia liczb zespolonych będą dawały wyniki 17B + 17B zaokrąglone (czyli część rzeczywista i urojona wyniku zostanie zaokrąglona).

Jak wiadomo, algorytmy działań arytmetycznych dla liczb zespolonych $z_1 = x_1 + iy_1$, $z_2 = x_2 + iy_2$ mają postać:

dodawanie

$$z_1 + z_2 = (x_1 + iy_1) + (x_2 + iy_2) = (x_1 + x_2) + i(y_1 + y_2), \tag{6-1}$$

odejmowanie

$$z_1 - z_2 = (x_1 + iy_1) - (x_2 + iy_2) = (x_1 - x_2) + i(y_1 - y_2), \tag{6-2}$$

mnożenie

$$z_1 z_2 = (x_1 + iy_1)(x_2 + iy_2) = (x_1x_2 - y_1y_2) + i(x_1y_2 + x_2y_1), \tag{6-3}$$

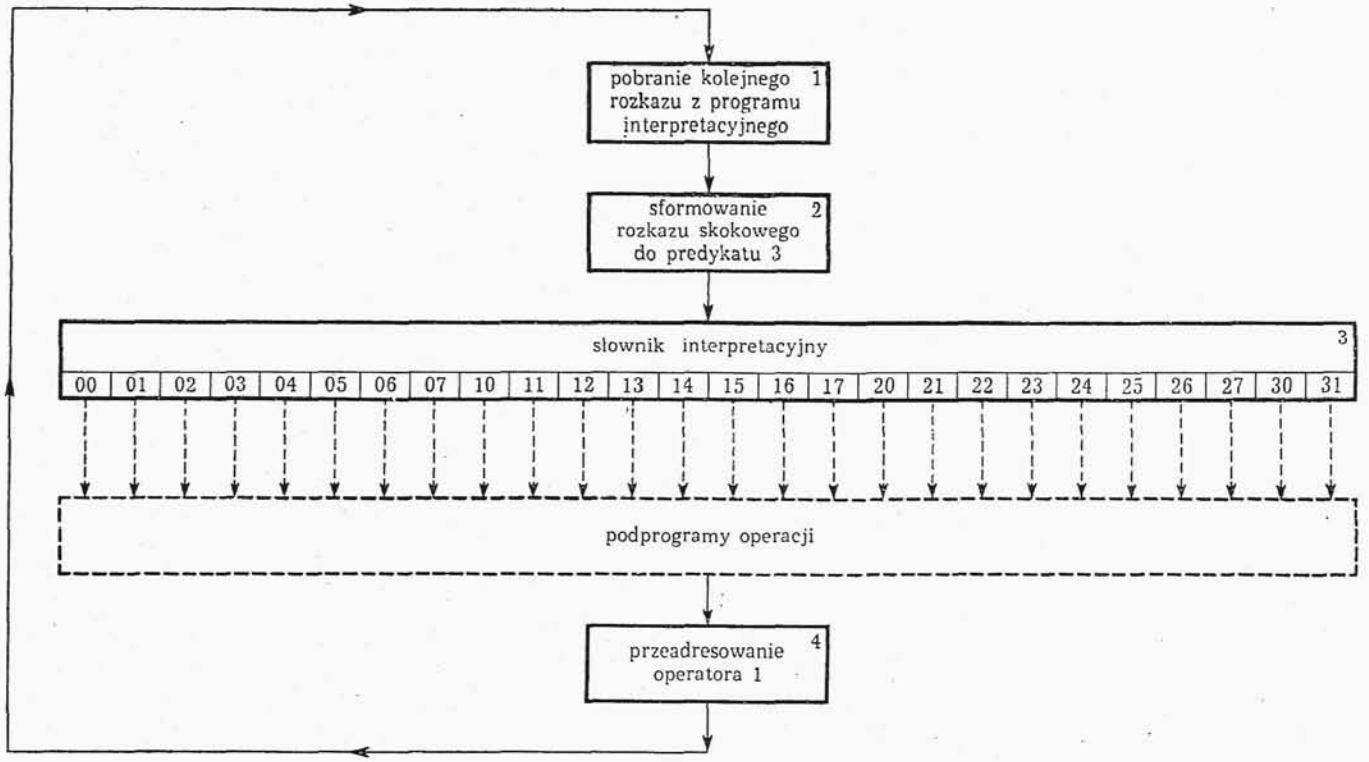
dzielenie

$$z_1 : z_2 = (x_1 + iy_1) : (x_2 + iy_2) = \frac{x_1x_2 + y_1y_2}{x_2^2 + y_2^2} + i\frac{x_2y_1 - x_1y_2}{x_2^2 + y_2^2}. \tag{6-4}$$

Omówimy obecnie program interpretujący.

Będziemy wyróżniali następujące komórki pamięci: „A" — komórka długa, tzw.

(1) Przykład opracowała Z. Jankowska.



Rys. 6-1. Schemat blokowy płaski programu interpretowania dla programu działań na liczbach zespolonych

pseudoakumulator, komórka ta składa się z dwóch komórek krótkich: „A_1" części „A" o adresie nieparzystym krótkim i „A_2" części „A" o adresie parzystym krótkim. „M" — — pseudorejestr mnożnej, „W" — pseudorejestr warunku (obie komórki pamięci „M" i „W" dzielimy podobnie jak „A" na „M_1", „M_2" i „W_1" i „W_2"). Ponadto wyróżniamy pseudolicznik rozkazów o adresie $k + 0000$. Przez adres n będziemy rozumieli adres komórki długiej składającej się z dwóch komórek krótkich n_1 i n_2, gdzie n_1 — adres komórki krótkiej nieparzystej, a n_2—adres komórki krótkiej parzystej. Na rysunku 6-1 podany jest schemat blokowy płaski programu interpretującego. Po zakodowaniu kolejne operatory i predykaty tego programu mają postać podaną w tabl. 6-1.

Tablica 6-1

Podprogram interpretowania

Operator	Kolejny adres	Kolejny rozkaz		Czynności wykonywane
1	$k+0000$	12	m	pobranie kolejnego rozkazu
	1	14	<kom. rob.>	
2	2	22	0014	sformowanie rozkazu przez słownik interpretacyjny
	3	10	<00 s>	
	4	14	$k+0005$	
3	5	00	7777	pobranie odpowiedniego rozkazu ze słownika
	6	12	$k+0000$	
4	7	10	$<2^{-16}>$	przeadresowanie „licznika rozkazów"
	$k+0010$	14	$k+0000$	
	1	02	$k+0000$	

Słownik interpretacyjny, poprzez który rozkaz $k + 0005$ programu interpretującego wywołuje odpowiedni podprogram modelujący operacje, ma postać:

$$
\begin{array}{rll}
s + 0000 & 02 & p_{00} + 0000, \\
1 & 02 & p_{01} + 0000, \\
2 & 02 & p_{02} + 0000, \\
3 & 02 & p_{03} + 0000, \\
4 & 02 & p_{04} + 0000, \\
5 & 02 & p_{05} + 0000, \\
6 & 02 & p_{06} + 0000, \\
7 & 02 & p_{07} + 0000, \\
s + 0010 & 02 & p_{10} + 0000, \\
1 & 02 & p_{11} + 0000, \\
2 & 02 & p_{12} + 0000, \\
3 & 02 & p_{13} + 0000, \\
4 & 02 & p_{14} + 0000, \\
5 & 02 & p_{15} + 0000, \\
6 & 02 & p_{16} + 0000, \\
7 & 02 & p_{17} + 0000,
\end{array}
$$

$s + 0020$ 02 $p_{20} + 0000$,
1 02 $p_{21} + 0000$,
2 02 $p_{22} + 0000$,
3 02 $p_{23} + 0000$,
4 02 $p_{24} + 0000$,
5 02 $p_{25} + 0000$,
6 02 $p_{26} + 0000$,
7 02 $p_{27} + 0000$,
$s + 0030$ 03 $p_{30} + 0000$,
1 02 $p_{31} + 0000$,
.....
7 02 $p_{37} + 0000$.

W tablicach 6-2÷6-18 omawiamy podprogramy działań modelowanych operacji (tzw. pseudooperacji).

Tablica 6-2

„00" n-pobranie i wykonanie jednego rozkazu

Kolejny adres	Kolejny rozkaz		Uwagi
$p_{00}+0000$	12	<kom. robocza>	pobranie i przetwarzanie rozkazu znajdującego się
1	10	<12 0000>	w komórce roboczej
2	14	$p_{00}+0003$	
3	12	n	wykonanie sformowanego wyżej rozkazu
4	14	<kom. rob.>	przesłanie odczytanego rozkazu do komórki roboczej
5	02	$k+0002$	skok do programu interpretacyjnego

Tablica 6-3

„01" n-skok pod adres drugiego rzędu

Kolejny adres	Kolejny rozkaz		Uwagi
$p_{01}+0000$	12	<kom. rob.>	pobranie i przetworzenie rozkazu znajdującego się
1	10	<11 0000>	w komórce roboczej
2	14	$p_{01}+0003$	
3	12	n	pobranie zawartości komórki o adresie n
4	02	$p_{02}+0001$	skok do podprogramu pseudooperacji 02

Tablica 6-4

„02" n-skokowa zmiana zawartości „licznika rozkazów"

Kolejny adres	Kolejny rozkaz		Uwagi
$p_{02}+0000$	12	<kom. rob.>	pobranie i przetworzenie rozkazu znajdującego się w
1	30	<00 7777>	komórce roboczej
2	10	<12 0000>	
3	14	$k+0000$	przesłanie sformowanego rozkazu do „licznika rozkazów"
4	02	$k+0000$	skok do początku programu interpretowania

Tablica 6-5

„03" n-skok warunkowy, gdy zawartość „W" różna od zera

Kolejny adres	Kolejny rozkaz		Uwagi
$p_{03}+0000$	12	„W"	Pobranie zawartości „W"
1	06	$k+0006$	skok do programu interpretowania, gdy („W") $\neq 0$
2	02	$p_{02}+0000$	skok do podprogramu pseudooperacji „02", gdy („W")$=0$

Tablica 6-6

„04" n-skok ze śladem pod adres n, $n+1$ do „licznika rozkazów"

Kolejny adres	Kolejny rozkaz		Uwagi
$p_{04}+0000$	12	<kom. rob.>	
1	10	<10 0000>	pobranie i przetworzenie rozkazu znajdującego się w komórce roboczej
2	14	$p_{04}+0004$	
3	12	$k+0000$	pobranie zawartości licznika rozkazów
4	14	n	wykonanie sformowanego rozkazu
5	12	<kom. rob.>	utworzenie nowej zawartości „licznika rozkazów"
6	10	<06 0001>	
7	14	$k+0000$	
10	02	$k+0000$	skok do programu interpretowania

Tablica 6-7

„05" n stop

Kolejny adres	Kolejny rozkaz		Uwagi
$p_{05}+0000$	00	<kom. rob.>	wykonanie rozkazu zapisanego w komórce roboczej
1	02	$k+0006$	po ponownym uruchomieniu maszyny skok do programu interpretowania

Tablica 6-8

„06" n-skok warunkowy, gdy zawartość „W" równa zeru

Kolejny adres	Kolejny rozkaz		Uwagi
$p_{06}+0000$	12	„W"	pobranie zawartości „W"
1	03	$k+0006$	skok do programu interpretowania, gdy („W") $\neq 0$
2	02	$p_{02}+0000$	skok do podprogramu pseudooperacji „02", gdy („W") $=0$

Tablica 6-9

„10" n-dodawanie liczb zespolonych

Kolejny adres	Kolejny rozkaz		Uwagi
$p_{10}+0000$	12	<kom. rob.>	
1	11	<00 4000>	
2	14	$p_{10}+0013$	sformowanie rozkazów dla wykonania dodawań części rzeczywistej i urojonej
3	10	$<2^{-16}>$	
4	14	$p_{10}+0006$	
5	12	„A_1"	
6	10	n_1	dodawanie części rzeczywistej ze skokiem do $d+0000$ w przypadku powstania nadmiaru
7	37	$d+0000$	
$p_{10}+0010$	14	„A_1"	
1	14	„W_1"	
2	12	„A_2"	
3	10	n_2	dodawanie części urojonej ze skokiem do $d+0000$ w przypadku powstania nadmiaru
4	37	$d+0000$	
5	14	„A_2"	
6	14	„W_2"	
7	02	$k+0006$	

Tablica 6-10

„12" n-przesłanie zawartości adresu n do pseudoakumulatora

Kolejny adres	Kolejny rozkaz		Uwagi
$p_{12}+0000$	00	<kom. rob.>	
1	14	„A"	
2	14	„W"	
3	02	$k+0006$	

Tablica 6-11

„13" n-przesłanie do pseudoakumulatora liczby sprzężonej do liczby zapisanej pod adresem n

Kolejny adres	Kolejny rozkaz		Uwagi
$p_{13}+0000$	12	<kom. rob.>	
1	11	<00 4001>	sformowanie rozkazów dla
2	14	$p_{13}+0010$	obliczenia liczby sprzężonej
3	10	<17 0001>	
4	14	$p_{13}+0005$	
5	12	n_1	
6	14	„A_1"	przesłanie części rzeczywistej
7	14	„W_1"	
$p_{13}+0010$	13	n_2	
1	37	$d+0000$	skok w przypadku powstania
2	14	„A_2"	nadmiaru
3	14	„W_2"	zmiana znaku części urojo-
4	02	$k+0006$	nej

Tablica 6-12

„14" n-przesłanie zawartości pseudoakumulatora do pamięci

Kolejny adres	Kolejny rozkaz		Uwagi
$p_{14}+0000$	12	<kom. rob.>	
1	14	$p_{14}+0003$	
2	12	„A"	
3	14	n	
4	14	„W"	
5	02	$k+0006$	

Tablica 6-13

„16" n-pomnożenie zawartości „M" (pseudorejestru mnożnej) przez zawartość komórki pamięci o adresie n z umieszczeniem zaokrąglonego wyniku w pseudoakumulatorze „A". Podobnie jak w naszej UPMC, jeśli chcemy pomnożyć przez siebie dwie liczby, musimy jedną z nich umieścić w „M", a następnie wykonać pseudooperację „16" n. Podprogram pseudooperacji „16" n korzysta z dwóch długich komórek roboczych (kom. rob. 1 oraz kom. rob. 2).

Kolejny adres	Kolejny rozkaz		Uwagi
$p_{16}+0000$	12	<kom. rob.>	
1	10	<10 4000>	
2	14	$p_{16}+0005$	n_2
3	10	$<2^{-16}>$	
4	14	$p_{16}+0014$	n_1

Tablica 6-13 (c.d.)

Kolejny adres	Kolejny rozkaz		Uwagi
5	27	n_2	
6	16	„M_2“	y_2y_1
7	37	$d+0000$	skok w przypadku
$p_{16}+0010$	14	<kom. rob. 1>	powstania nadmiaru
1	16	„M_1“	y_2x_1
2	37	$d+0000$	skok w przypadku
3	14	<kom. rob. 2>	powstania nadmiaru
4	27	n_1	
5	16	„M_1“	x_2x_1
6	37	$d+0000$	skok w przypadku powstania nadmiaru
7	11	<kom. rob. 1>	$x_2x_1-y_2y_1=x$
$p_{16}+0020$	37	$d+0000$	skok w przypadku
1	21	0021	powstania nadmiaru
2	15	0000	
3	20	0021	
4	37	$d+0000$	
5	14	„A_1“	
6	14	„W_1“	
7	16	„M_2“	x_2y_2
$p_{16}+0030$	37	$d+0000$	skok w przypadku powstania nadmiaru
1	10	<kom. rob. 2>	$x_2y_1+x_1y_2=y$
2	21	0021	
3	15	0000	
4	20	0021	
5	37	$d+0000$	skok w przypadku
6	14	„A_2“	powstania nadmiaru
7	14	„W_2“	
$p_{16}+0040$	02	$k+0006$	

Tablica 6-14

„17“ n-podzielenie zawartości pseudoakumulatora „A“ przez zawartość komórki pamięci o adresie n, z umieszczeniem wyniku w pseudoakumulatorze „A“. Podprogram pseudooperacji „17“ n korzysta z trzech długich komórek roboczych (kom. rob. 1, kom. rob. 2, i kom. rob. 3)

Kolejny adres	Kolejny rozkaz		Uwagi
$p_{17}+0000$	12	<kom. rob.>	
1	11	<01 4000>	
2	14	$p_{17}+0012$	
3	10	$<2^{-16}>$	
4	14	$p_{17}+0024$	
5	10	<11 0000>	
6	14	$p_{17}+0023$	
7	11	$<2^{-16}>$	

Tablica 6-14 (d.c.)

Kolejny adres	Kolejny rozkaz		Uwagi
p_{17}+0010	14	p_{17}+0011	
1	27	n_2	y_2
2	16	n_2	y_2
3	37	d+0000	skok w przypadku powstania nadmiaru
4	14	<kom. rob. 1>	
5	16	„A_2“	y_1y_2
6	37	d+0000	skok w przypadku powstania nadmiaru
7	14	<kom. rob. 2>	
p_{17}+0020	16	„A_1“	x_1y_2
1	37	d+0000	skok w przypadku powstania nadmiaru
2	14	<kom. rob. 3>	
3	27	n_1	x_2
4	16	n_1	x_2
5	37	d+0000	skok w przypadku powstania nadmiaru
6	10	<kom. rob. 1>	$x_2^2+y_2^2$
7	37	d+0000	skok w przypadku powstania nadmiaru
p_{17}+0030	14	<kom. rob. 1>	
1	16	„A_1“	x_1x_2
2	37	d+0000	skok w przypadku powstania nadmiaru
3	10	<kom. rob. 2>	$x_1x_2+y_1y_2$
4	37	d+0000	skok w przypadku powstania nadmiaru
5	14	<kom. rob. 2>	
6	16	„A_2“	y_1x_2
7	37	d+0000	skok w przypadku powstania nadmiaru
p_{17}+0040	11	<kom. rob. 3>	$y_1x_2-x_1y_2$
1	37	d+0000	skok w przypadku powstania nadmiaru
2	17	<kom. rob. 1>	y
3	37	d+0000	skok w przypadku powstania nadmiaru
4	21	0021	
5	15	0000	
6	20	0021	
7	14	„A_2“	
p_{17}+0050	14	„W_2“	
1	12	<kom. rob. 2>	
2	17	<kom. rob. 1>	x
3	37	d+0000	skok w przypadku powstania nadmiaru
4	21	0021	
5	15	0000	
6	20	0021	
7	37	d+0000	skok w przypadku powstania nadmiaru
p_{17}+0060	14	„A_1“	
1	14	„W_1“	
2	02	k+0006	

Tablica 6-15

„20" n-pomnożenie liczby zespolonej przez skalar 2^n

Kolejny adres	Kolejny rozkaz		Uwagi
$p_{20}+0000$	12	<kom. rob.>	
1	14	$p_{20}+0004$	
2	14	$p_{20}+0011$	
3	12	„A_1"	
4	20	n	
5	37	$d+0000$	skok przy
6	14	„A_1"	nadmiarze
7	14	„W_1"	
$p_{20}+0010$	12	„A_2"	
1	20	n	
2	37	$d+0000$	skok przy
3	14	„A_2"	nadmiarze
4	14	„W_2"	
5	02	$k+0006$	

Tablica 6-16

„21" n-pomnożenie liczby zespolonej z przez skalar 2^{-n}

Kolejny adres	Kolejny rozkaz		Uwagi
$p_{21}+0000$	12	<kom. rob.>	
1	14	$p_{21}+0004$	
2	14	$p_{21}+0010$	
3	12	„A_1"	
4	21	n	
5	14	„A_1"	
6	14	„W_1"	
7	12	„A_2"	
$p_{21}+0010$	21	n	
1	14	„A_2"	
2	14	„W_2"	
3	02	$k+0006$	

Tablica 6-17

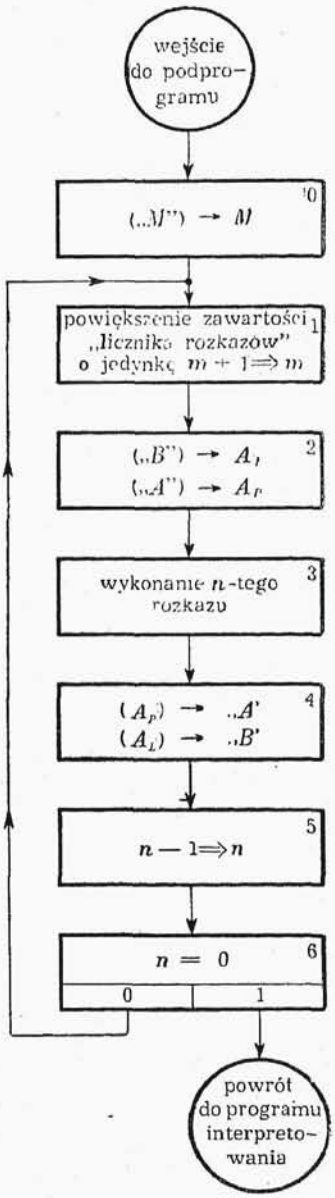
„26" N — obliczenie modułu liczby zespolonej z, znajdującej się w pseudoakumulatorze. Podprogram korzysta z jednej komórki roboczej: kom. rob. 1 oraz z podprogramu obliczenia pierwiastka kwadratowego

Kolejny adres	Kolejny rozkaz		Uwagi
$p_{26}+0000$	27	„A_1"	obliczenie x^2
1	16	„A_1"	
2	14	<kom. rob. 1>	
3	27	„A_2"	obliczenie y^2
4	16	„A_2"	
5	10	<kom. rob. 1>	x^2+y^2
6	37	$d+0000$	skok przy nadmiarze
7	04	<podprogr. $\sqrt{\ }$>	wywołanie podprogramu pierwiastka kwadratowego
$p_{26}+0010$	14	„A"	
1	14	„W"	
2	02	$k+0006$	

Tablica 6-18

„27" n-przesłanie zawartości komórki o adresie n, do pseudorejestru mnożnej „M"

Kolejny adres	Kolejny rozkaz		Uwagi
$p_{27}+0000$	12	<kom. rob.>	
1	11	<15 0000>	
2	14	$p_{27}+0003$	
3	12	n	
4	14	„M"	
5	14	„W"	
6	02	$k+0006$	



Rys. 6-2. Schemat blokowy pseudooperacji „07"

W przykładowej UPMC rozkaz w kodzie 07n nie istnieje. Natomiast w naszym programie wykorzystamy kod 07n na pseudooperację: wykonaj n kolejnych pseudorozkazów, występujących za pseudorozkazem 07n jako rozkazy maszyny. Inaczej mówiąc, nie interpretujemy n pseudooperacji. Na rysunku 6-2 podany jest schemat blo-

Tablica 6-19

Operator lub predykat	Kolejny adres	Kolejny rozkaz		Uwagi
0	$p_{07}+0000$	27	<„M“>	ustawienie rejestru M
1	1	12	$k+0000$	przeadresowanie „licznika rozkazów“
	2	10	$<2^{-16}>$	
	3	14	$k+0000$	
	4	11	<12 0000>	sformowanie operatora 3
	5	14	$p_{07}+0011$	
2	6	12	<„B“>	ustawienie akumulatora
	7	22	0042	
	$p_{07}+0010$	10	<„A“>	
3	1	00	$m+1$	wykonanie kolejnego nieinterpretowanego rozkazu
4	2	14	<„A“>	zapamiętanie akumulatora
	3	23	0042	
	4	14	<„B“>	
5	5	12	<kom. rob.>	$n-1 \Longrightarrow n$
	6	11	$<2^{-16}>$	
	7	14	<kom. rob.>	
6	$p_{07}+0020$	11	<07 0000>	
	1	06	$p_{07}+0001$	
	2	02	$k+0000$	

kowy podprogramu realizującego pseudooperację 07n. W tablicy 6-19 podane są kolejne rozkazy programu realizującego pseudooperacje 07n.

Uwaga: „11“ n-odejmowanie liczb zespolonych wykonujemy za pomocą tego samego programu co pseudooperację „10“n. Jak łatwo zauważyć, gdy w komórce roboczej znajduje się rozkaz „11“m, wówczas podprogram dodawania staje się podprogramem odejmowania.

Pseudooperacje o częściach operacyjnych: „15“, „22“, „23“, „24“, „25“, „30“, są po prostu rozkazami naszej UPMC i realizowane są wspólnie poprzez podprogram będący fragmentem podprogramu pseudooperacji 07n. W przypadku powstania nadmiaru przy wykonywaniu któregoś z działań na liczbach zespolonych jest wykonywany podprogram drukujący zawartość pseudolicznika rozkazów. Podprogram ten ma następującą postać:

$d+0000$		
1	12	$k+0000$
2	04	$d+0010$
3	04	$d+0010$
4	04	$d+0010$
5	04	$d+0010$
6	05	$d+0006$
7	01	$d+0000$

$d + 0010$		
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.