4. METODYKA PROGRAMOWANIA

4.0. UWAGI WSTĘPNE

Proces przygotowania dowolnego problemu do rozwiązywania na UPMC rozpada się na kilka etapów.

1. Stwierdzenie istnienia rozwiązania problemu, jego jednoznaczności itp.

2. Wybór metody numerycznej, najkorzystniejszej dla rozwiązania danego problemu przy użyciu UPMC.

3. Skonstruowanie algorytmu realizującego metodę numeryczną wybraną w punkcie 2.

4. Wybór organizacji programu realizującego algorytm wybrany w punkcie 3, możliwie najkorzystniejszy ze względu na maszynę, którą dysponujemy.

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 zestawienie 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.

7. Sprawdzenie ułożonego programu na maszynie, przez sprawdzenie działania poszczególnych cykli i iteracji, wraz z ewentualnym próbnym wykonaniem obliczeń (np. dla tych wartości, dla których znamy wartości rozwiązania lub potrafimy dość dokładnie określić przedział, w którym powinno znajdować się rozwiązanie).

Punkty 1 i 2 nie należą do programowania, obejmujemy je wspólną nazwą analizy problemu. Punkt 2 pokrótce omówimy w niniejszym paragrafie. Punkt 3 i 4 omówimy w dalszych paragrafach rozdziału 4. Punkty 5 i 6 omówiliśmy w rozdz. 3. Punkt 7 omówimy w rozdz. 5.

Rozwiązanie zadań na małych maszynach cyfrowych wymaga bardzo starannej matematycznej analizy problemu. Mała prędkość maszyny (około 100 operacji na sekundę) i mała pojemność pamięci wewnętrznej (1024 słowa długie) bardzo ogranicza możliwość stosowania metod numerycznych.

Warunki, jakie musi spełniać metoda numeryczna dla małej maszyny cyfrowej, dość pobieżnie i nieprecyzyjnie możemy scharakteryzować w następujących punktach.

1. Metoda winna zapewniać potrzebną dokładność obliczeń. Jak wiadomo, dokładność rozwiązania zagadnienia zależy od dokładności metody i dokładności obliczeń (dokładność arytmetyczna).

2. Wybrana metoda powinna pozwalać rozwiązać zagadnienie przy możliwie najmniejszym nakładzie pracy i w możliwie najkrótszym czasie.

3. Współczynnik związania algorytmu nie może być wysoki. Przez współczynnik związania algorytmu rozumiemy ilość danych pośrednich (zapamiętywanych przez maszynę) koniecznych dla przejścia od jednego etapu obliczeń do drugiego. Na przykład przy metodzie Runeggo-Kutta rozwiązywanie równania różniczkowego $y' = f(x,y)$, z warunkami początkowymi $x = x_0, y = y_0$, współczynnik związania algorytmu równa się 2, przy stosowaniu zaś metody Adamsa dla tego samego równania współczynnik algorytmu wynosi 6.

4. Metoda musi sprowadzać rozwiązanie do elementarnych operacji (bezpośrednio lub przez podprogramy),

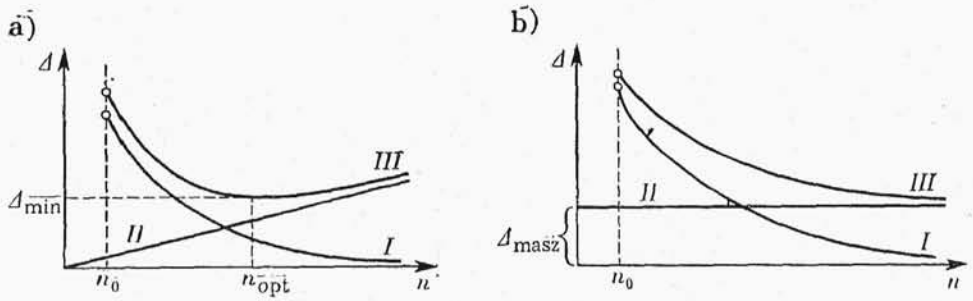
5. Metoda powinna być optymalna ze względu na czas liczenia przy ustalonej pojemności pamięci.

6. Metoda powinna zabezpieczyć prosty wybór przedziału zmienności wyników pośrednich i ostatecznych.

7. Metoda powinna dawać łatwość automatycznego wyboru przedziału w czasie rozwiązywania zagadnienia.

Dla pełniejszego omówienia punktu 1 należy zwrócić uwagę na pewną klasyfikację metod numerycznych. Przy założeniu, że opieramy się w naszych rozważaniach na teorii błędu maksymalnego (Bibliografia [12]), musimy wyróżnić dwie klasy metod numerycznych, a mianowicie:

1. Klasa metod, w których istnieje taka ilość kroków (na które dzielimy przedziały i w których rozwiązujemy nasze zadania), przy której licząc ze stałą dokładnością mamy najmniejszy błąd maksymalny wyniku. Postaramy się zilustrować to zjawisko na wykresie. Sporządźmy wykres, na którym na osi poziomej będziemy odkładali ilość kroków n, na osi pionowej zaś będziemy odkładali błąd maksymalny (rys. 4-1a). Prosta



Rys. 4-1. Zależność błędu maksymalnego wyniku od liczby kroków

a) przy wzrastającym maksymalnym błędzie rachunkowym, b) przy stałym maksymalnym błędzie rachunkowym

II na rys. 4-1a pokazuje, że maksymalny błąd arytmetyczny, przy liczeniu na słowach o ustalonej długości, jest w przybliżeniu proporcjonalny do ilości operacji, czyli do ilości kroków. Krzywa *I* na rys. 4-1a pokazuje, że błąd metody dąży asymptotycznie do zera wraz ze wzrostem ilości kroków. Krzywa *III* na rys. 4-1a jest wynikiem nakładania się maksymalnego błędu arytmetycznego i maksymalnego błędu metody.

Do powyższej klasy należą np. metody rozwiązywania równań różniczkowych zwyczajnych (z warunkami początkowymi) Eulera, Adamsa, Rungego-Kutty, metody numerycznych kwadratur jak metoda Simpsona i inne.

2. Klasa metod iteracyjnych, w których wraz ze zwiększeniem ilości kroków, licząc ze stałą dokładnością, błąd maksymalny rozwiązania (o ile postępowanie jest zbieżne) dąży do pewnej wartości asymptotycznej. Zilustrujemy to zjawisko na wykresie. Sporządźmy wykres, na którym na osi poziomej będziemy odkładali ilość kroków n (krotność iteracji), na osi pionowej zaś będziemy odkładali błąd maksymalny (rys. 4-1b). Prosta *II* na rys. 4-1b pokazuje, że przy stosowaniu formuły iteracyjnej maksymalny błąd arytmetyczny jest stały. Krzywa *I* na rys. 4-1b pokazuje, że błąd metody asymptotycznie dąży do zera wraz ze wzrostem ilości iteracji n. Krzywa *III* na rys. 4-1b jest wynikiem nakładania się maksymalnego błędu arytmetycznego i maksymalnego błędu metody.

Do powyższej klasy metod iteracyjnych należy np. metoda Gaussa-Seidla rozwiązywania układów równań algebraicznych liniowych.

Po dokonaniu wyboru metody numerycznej musimy oszacować potrzebną nam dokładność wyników pośrednich i w zależności od tego podjąć decyzję, jaką arytmetykę będziemy stosowali w naszym programie (przecinek stały czy zmienny lub przecinek programowy). Czasami może się zdarzyć, że dokładność uzyskiwana dla każdej z wymienionych arytmetyk jest za mała, wówczas możemy stosować arytmetykę podwójną (rzadkie przypadki).

Musimy pamiętać o tym, że główne trudności obliczeniowe na maszynach cyfrowych nie są związane z programowaniem, ale z doborem odpowiedniej metody numerycznej.

4.1. ORGANIZACJA PROGRAMU

4.1.0. Zanim przejdziemy do omówienia organizacji programu, wprowadźmy kilka pomocniczych pojęć.

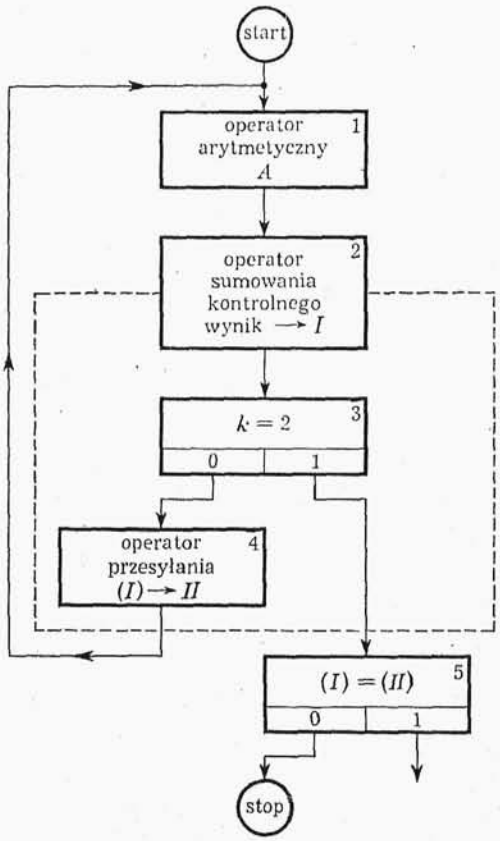
Definicja 1. Przez podprogram będziemy rozumieli zespół operatorów i predykatów, przeznaczony do wykonania zamkniętego fragmentu obliczeń (gdzie przez zamknięty fragment obliczeń rozumiemy fragment obliczeń wykonywanych według ustalonego lub nastawionego uprzednio algorytmu).

Definicja 2. Przez podprogram zamknięty będziemy rozumieli taki i tylko taki podprogram, który wywołujemy rozkazami programu głównego z zostawieniem w odpowiednim ustalonym miejscu „śladu", który umożliwia z kolei powrót do programu głównego.

Definicja 3. Przez podprogram otwarty będziemy rozumieli taki i tylko taki podprogram, którego operatory i predykaty umieszczamy bezpośrednio w sekwencji operatorów i predykatów programu głównego.

4.1.1. Kontrola wyników pośrednich. W wielu programach dla zapewnienia poprawności wyników musimy kontrolować poszczególne etapy obliczeń. Jedyną uniwersalną metodą kontroli, stosowaną obecnie, jest metoda sum kontrolnych. Przy-

kładowo metodę sumowań kontrolnych możemy zrealizować następująco: przypuśćmy, że chcemy zastosować metodę sumowań kontrolnych dla sprawdzenia prawidłowości pracy pewnego operatora arytmetycznego A, wówczas umieszczamy za operatorem arytmetycznym A operator sumowania kontrolnego, który wszystkie wyniki obliczeń



Rys. 4-2. Schemat blokowy metody sum kontrolnych

wykonywanych przez operator A i ewentualnie zmienne rozkazy operatora A kolejno sumuje ze sobą (odrzucając części całkowite, inaczej mówiąc, gubiąc wszystkie nadmiary), a następnie otrzymaną w ten sposób liczbę umieszcza pod pierwszym adresem roboczym sumowania kontrolnego. Za operatorem sumowania kontrolnego umieszczamy predykat zwany sitem [licznik samoodnawiający się, liczący do dwóch — wzór (3-13)]. Po pierwszym wykonaniu operatora sumowania kontrolnego sito przekazuje sterowanie operatorowi przesyłania zawartości pierwszego miejsca roboczego do drugiego miejsca roboczego sumowania kontrolnego, operator ten przekazuje z kolei sterowanie operatorowi A. Po drugim wykonaniu operatora sumowania kontrolnego sito przekazuje sterowanie predykatowi porównania sum kontrolnych zapisanych pod pierwszym i drugim adresem roboczym sumowania kontrolnego. W przypadku gdy obie sumy kontrolne

są zgodne, predykat porównania sum kontrolnych przekaże sterowanie kolejnemu operatorowi programu, w przypadku zaś, gdy obie sumy kontrolne nie są identyczne, powoduje zatrzymanie maszyny. Na rysunku 4-2 pokazany jest schemat blokowy metody sum kontrolnych.

W tablicy 4-1 podajemy kolejne rozkazy tej części schematu blokowego z rys. 4-2, która została otoczona przerywaną linią.

Tablica 4-1

Operator lub predykat	Kolejny adres	Kolejny rozkaz	Objaśnienia
2	$s+0000$	14 $b+4000$	ostatni rozkaz operatora sumowania kontrolnego
3	1	13 $b+0004$	sito-licznik liczący do dwóch, samoodnawiający się: początkowa zawartość adresu $b+0004$—zero β — dowolna liczba krótka liczba dodatnia
	2	11 $<\beta>$	
	3	14 $b+0004$	
	4	06 $s+0010$	
4	5	12 $b+4000$	operator przesłania zawartości pierwszego adresu roboczego operatora sumowania kontrolnego pod drugi adres roboczy
	6	14 $b+4002$	
	7	02 <wejście operatora A>	

Sumowanie kontrolne można przeprowadzić korzystając zamiast z predykatu typu „sita" z metody dwukrotnego wywołania operatora A. Podana przykładowo organizacja sumowania kontrolnego daje się łatwo stosować na wszystkich UPMC.

Jak widać z powyższego rozważania, metoda sumowania kontrolnego pochłania dużo czasu i zajmuje dużo miejsca w pamięci maszyny. Dlatego też staramy się rozwiązać kontrolę na innej drodze. Rozwiązanie takie dopasowujemy indywidualnie do rozwiązywanego zagadnienia. Na przykład jeśli rozwiązujemy układ równań różniczkowych zwyczajnych z warunkami początkowymi, to dołączamy do tego układu jeszcze jedno równanie, mianowicie równanie na długość wektora wodzącego funkcji rozwiązujących zadany układ (oczywiście, nie zawsze takie postępowanie jest możliwe). Po wykonaniu każdego kroku obliczeń, obliczamy pierwiastek z sum kwadratów rozwiązań poszczególnych równań i porównujemy otrzymaną wielkość z rozwiązaniem równania na wektor wodzący. Jeśli liczby te nie różnią się o więcej niż z góry zadaną liczbę $\varepsilon>0$, to prowadzimy obliczenia dalej, jeśli zaś różnica między nimi nie jest większa lub równa z góry zadanemu $\varepsilon>0$, to program spowoduje zatrzymanie maszyny. Przykładów takich metod można podać wiele, programista mający wprawę bez większej trudności potrafi zbudować tego typu metody kontrolne.

4.1.2. Manipulacje ręczne. Czasami zamiast zatrzymywać maszynę, wygodniej jest stosować tzw. podprogram dzwonka. To znaczy po zakończeniu pewnego etapu obliczeń lub w przypadku powstania błędu w obliczeniach zamiast zatrzymania maszyny wywołujemy pewien podprogram, który powoduje uruchomienie dzwonka dalekopisu

sygnalizującego obsłudze maszyny, że musi ona wykonywać pewne dodatkowe czynności. Dzwonek dalekopisu będzie dzwonił tak długo, dopóki obsługujący maszynę nie przedstawi odpowiedniego przełącznika na pulpicie sterowania, co spowoduje przerwanie pętli podprogramu i zatrzymanie maszyny. W tablicy 4-2 omawiamy pokrótce budowę i działanie takiego podprogramu.

Tablica 4-2

Kolejny adres	Kolejny rozkaz	Objaśnienia
$k+0000$	miejsce na ślad	
1	12 <16 0000>	nastawienie dalekopisu na drukowanie cyfr (ponieważ dzwonek znajduje się w grupie cyfr)
2	25 0000	
3	13 <02 0000>	przesłanie kodu dzwonka
4	25 0000	pętla dzwonka po włączeniu warunku Y na którąś
5	06 $k+0000$	z pierwszych czterech pozycji, pętla zostaje przerwana
6	05 $k+0010$	„stop" z pobraniem po uruchomieniu do wykonania I taktu pracy maszyna wykona rozkaz
7	01 $k+0000$	$k+0007$ i poprzez ślad powróci do podprogramu
$k+0010$	02 <program druk. liczników>	głównego, po uruchomieniu do wykonania II taktu pracy maszyna wykona rozkaz $k+0010$, a następnie przejdzie do drukowania zawartości licznika cykli

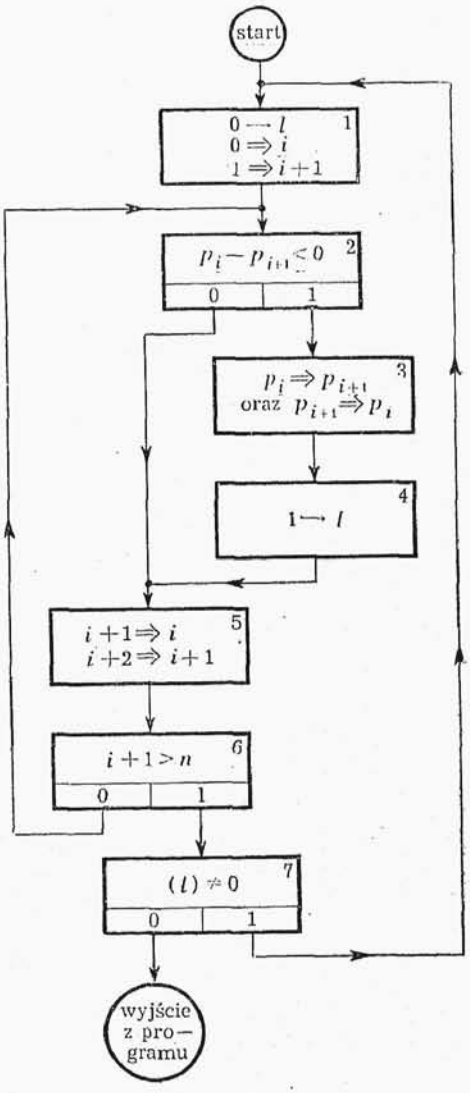
Podprogram podany w tabl. 4-2 możemy skrócić o dwa rozkazy, przyjmując następującą konwencję: stanem normalnym dalekopisu jest nastawienie go na drukowanie cyfr i znaków, po każdorazowym drukowaniu liter sprowadzamy dalekopis do stanu normalnego.

4.1.3. Drukowanie zawartości liczników cykli i licznika rozkazów. W przypadku zatrzymania się maszyny na skutek przekroczenia zakresu czy też na skutek błędu w obliczeniach, obsługujący maszynę musi mieć możliwość zorientowania się, w którym miejscu realizacji programu maszyna zatrzymała się, co było przyczyną tego zatrzymania. Stan wykonania programu charakteryzuje zawartość wszystkich liczników cykli, jakie znajdują się w programie. Dlatego też do programu dołączamy podprogram drukujący zawartość liczników cykli. Gdy maszyna zostaje ponownie uruchomiona, po zatrzymaniu, zostaje wywołany podprogram drukowania zawartości liczników cykli.

Po skonstruowaniu algorytmu realizującego wybraną metodę numeryczną, rozbijamy ten algorytm na poszczególne kroki obliczeniowe — operatory i predykaty; otrzymany w ten sposób schemat obliczeń uzupełniony operatorami i predykatami typowymi dla obliczeń automatycznych i rysujemy schemat blokowy programu. Otrzymany w ten sposób schemat blokowy przekształcamy do możliwie najprostszej postaci. Należy zwrócić tu uwagę na fakt nieistnienia jakiejś ogólnej i łatwej do formalizacji metody uproszczenia schematu blokowego. Pewne wyniki w uproszczeniu algorytmów zostały uzyskane przez Janowa, wyniki te zostały szczegółowo omówione w książce Kitowa i Krynickiego (Bibliografia [9]). Jednakże wyniki te nie dają jeszcze praktycznie opła-

calnych metod uproszczenia schematów blokowych i właściwie sprawę tę należy nadal uważać za otwartą.

4.1.4. Przykład programu. Niech będzie dany układ $n+1$ liczb stałoprzecinkowych 17 B umieszczonych pod kolejnymi adresami $a \div a+n$. Przypuśćmy, że dla pewnego celu musimy uporządkować te liczby w kolejności od większej do mniejszej. Jako algorytm postępowania wybierzemy algorytm, który porównuje dwie sąsiednie liczby i-tą oraz $(i+1)$-szą, gdzie $i = 0,1, \ldots, n-1$; w przypadku gdy i-ta liczba jest mniejsza od liczby $(i+1)$-szej zamieniamy je miejscami, w przypadku zaś gdy i-ta liczba jest większa bądź równa liczbie $(i+1)$-szej, pozostawiamy obie liczby na swoich miejscach, po czym zastępujemy i przez $i+1$, $i+1$ przez $i+2$ i postępujemy jak wyżej. Porządkowanie zakończymy wówczas, gdy w wyniku porównania wszystkich kolejnych par, w naszym ciągu $n+1$ liczb, nie dokonamy żadnego przestawienia. Na rysunku 4-3 pokazany jest schemat blokowy takiego programu.



Rys. 4-3. Schemat blokowy pierwszej metody porządkowania liczb

Kodowanie zaczniemy od predykatu nr 2; w tablicy 4-3 podajemy zakodowane operatory i predykaty nr 2, 3, 4.

Porównując z sobą operatory nr 1 i 5 możemy wydzielić z nich wspólną część, przeadresowującą, którą nazwiemy operatorem P. Zakładając, że w akumulatorze znajduje się $12<p_0>$ w przypadku operatora nr 1 albo $12<p_{i+1}>$ w przypadku operatora nr 5 przyjmiemy, że operator P ma postać podaną w tabl. 4-4.

Korzystając z operatora P możemy zakodować operator nr 1 za pomocą czterech rozkazów (tabl. 4-5).

Operator nr 5 będzie się składał z dwóch rozkazów. Możemy obecnie zakodować operator nr 5, predykat nr 6 i predykat nr 7 (tabl. 4-6).

Możemy wypisać wszystkie rozkazy programu głównego, a następnie umieścić program główny, podprogram i parametry w pamięci maszyny. Kolejne rozkazy programu głównego podajemy w tabl. 4-7.

W naszym programie korzystamy z następujących parametrów (w kodzie rozkazowym):

1) 02 0000,
2) 01 0000,
3) 00 0000,
4) 00 0001 $= <+2^{-16}>$,
5) 11 p_{n+1}.

Należy podkreślić, że przedstawiona powyżej metoda postępowania przy porządkowaniu liczb nie ma znaczenia praktycznego. W praktyce wygodniejszą metodą jest me-

Tablica 4-3

Operator lub predykat	Kolejny adres	Kolejny rozkaz		Objaśnienia
2	s+0000	12	$<p_i>$	skok przy $p_i \geqslant p_{i+1}$
	1	11	$<p_{i+1}>$	
	2	06	t+0000	
3	3	12	$<p_i>$	b+0000 adres roboczy
	4	14	b+0000	$p_i \rightarrow b$+0000
	5	12	$<p_{i+1}>$	
	6	14	$<p_i>$	$p_{i+1} \Longrightarrow p_i$
	7	12	b+0000	$(b+0000) \Longrightarrow p_{i+1}$
	s+0010	14	$<p_{i+1}>$	
4	1	12	$<2^{-16}>$	
	2	14	b+0001	b+0001 wskaźnik przestawienia

Tablica 4-4

Operator lub predykat	Kolejny adres	Kolejny rozkaz		Objaśnienia
P	p+0000	77	7777	miejsce na ślad
	1	14	s+0000	
	2	14	s+0003	
	3	10	<02 0000>	sformowanie rozkazu 14
	4	14	s+0006	
	5	10	$<2^{-16}>$	
	6	14	s+0010	
	7	11	<02 0000>	
	p+0010	14	s+0005	
	1	11	<01 0000>	
	2	14	s+0001	
	3	01	p+0000	powrót poprzez ślad do programu głównego

toda wyszukiwania maksymalnej z $n+1$ liczb i przesyłanie jej na miejsce pierwszej liczby, pierwszą liczbę zaś przesyłamy na miejsce maksymalnej, po czym powtarzamy proces wyszukując maksymalną liczbę spośród n-liczb (pierwszej liczby nie będziemy brali pod

Tablica 4-5

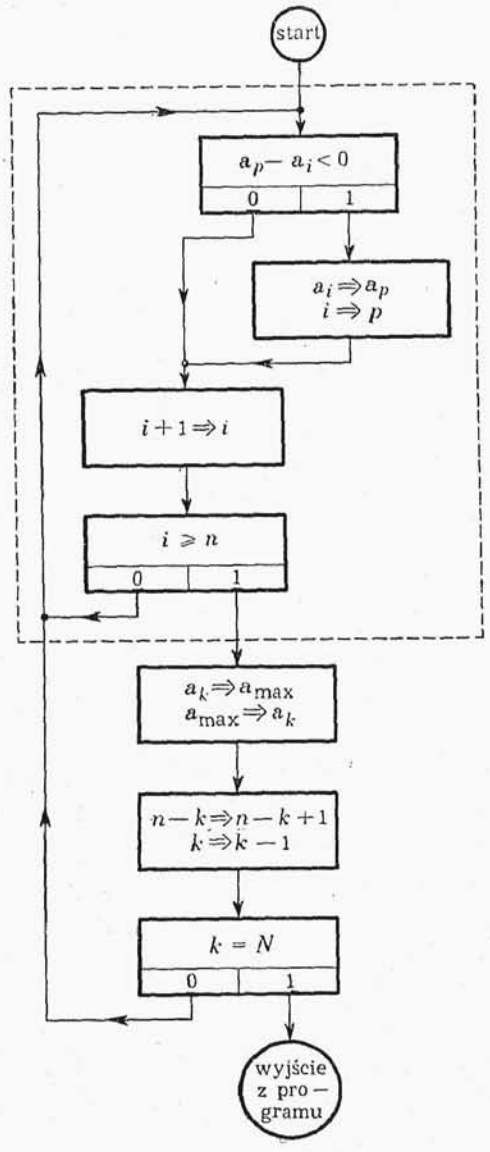
Operator lub predykat	Kolejny adres	Kolejny rozkaz	Objaśnienia
1	k+0000	12 <zero>	
	1	14 b+0001	wyzerowanie wskaźnika przestawień
	2	12 $<12<p_0>>$	
	3	04 p+0000	wywołanie podprogramu przeadresowania

Tablica 4-6

Operator lub predykat	Kolejny adres	Kolejny rozkaz	Objaśnienia
5	t+0000	12 s+0005	wywołanie podprogramu przeadresowania
	1	04 p+0000	
6	2	11 $<11<p_{n+1}>>$	sprawdzenie warunku i skok przy $i+1>n$
	3	03 s+0000	
7	4	12 b+0001	sprawdzenie zawartości wskaźnika przestawień
	5	03 k+0000	

Tablica 4-7

Operator lub predykat	Kolejny adres	Kolejny rozkaz	Objaśnienia
1	k+0000	12 <zero>	wyzerowanie licznika l
	1	14 b+0001	
	2	12 $<12<p_0>>$	wywołanie podprogramu P
	3	04 p+0000	
2	4	12 $<p_i>$	sprawdzenie warunku $p_i - p_{i+1} < 0$
	5	11 $<p_{i+1}>$	
	6	06 k+0017	
3	7	12 $<p_i>$	
	k+0010	14 b+0000	
	1	12 $<p_{i+1}>$	
	2	14 $<p_i>$	
	3	12 b+0000	
	4	14 $<p_{i+1}>$	
4	5	12 $<2^{-16}>$	przesłanie „jedynki" do licznika l
	6	14 b+0001	
5	7	12 k+0011	wywołanie podprogramu P
	k+0020	04 p+0000	
	1	11 $<11<p_{n+1}>>$	sprawdzenie, czy nie porównywaliśmy
	2	03 k+0004	już ostatniej pary
	3	12 b+0001	sprawdzenie zawartości licznika l
	4	03 k+0000	



Rys. 4-4. Schemat blokowy drugiej metody porządkowania liczb

Linią przerywaną ujęto szukanie maksymalnej liczby w ciągu $k \div n$, gdzie $k = 0, 1, \ldots, n-1$ oraz $i = k, k+1 \ldots, n$, przy czym $k \leqslant p < n$.

uwagę) itd. aż zostanie nam tylko jedna liczba, wówczas kończymy nasz proces. Na rysunku 4-4 podany jest schemat blokowy programu realizującego powyższy algorytm postępowania.

4.2. PROGRAMY OBLICZENIOWE O STAŁYM PRZECINKU

4.2.0. Omawiana przez nas UPMC, jak wiemy, wykonuje operacje arytmetyczne na liczbach z przedziału $<-1, 1-2^{-33}>$, jednak w większości problemów fizycznych i technicznych argumenty wartości funkcji przebiegają znacznie szersze przedziały, jeśli więc chcemy korzystać bezpośrednio z działań arytmetycznych maszyny, musimy wprowadzić nowe zmienne przebiegające przedział $<-1, 1-2^{-33}>$. Oczywiście, nie wystarczy tylko sprowadzanie danych początkowych do wyżej wymienionego przedziału, musimy ponadto zapewnić sobie mieszczenie się w tym przedziale wszystkich wyników pośrednich. Taką metodę prowadzenia obliczeń będziemy nazywali metodą stałego przecinka.

W czasie analizy problemu, w przypadku programowania ze stałym przecinkiem, wyróżniamy następujące etapy:

1) analiza równań i formuł zadania, w celu określenia granic przedziałów, zmienności argumentów i wartości funkcji,

2) wybór mnożników dla każdej z wielkości zmiennych, zapewniających mieszczenie się w przedziale $<-1, 1-2^{-33}>$ iloczynu mnożnika i danej wielkości zmiennej,

3) wprowadzenie nowych zmiennych, przebiegających już przedział $<-1, 1-2^{-33}>$.

Należy w tym miejscu bardzo mocno podkreślić trudności związane z określeniem granic przedziałów zmienności argumentów i wartości funkcji, nie istnieje bowiem żadna uniwersalna metoda dokładnego określenia przedziałów zmienności. W praktyce ogra-