Marek Greniewski

WSTĘP DO PROGRAMOWANIA i MODELOWANIA CYFROWEGO

„. . . Maszyny cyfrowe stały się dla matematyków i inżynierów instrumentem obliczeniowym, działającym analogicznie do biur rachmistrzów. Podobny pogląd reprezentuje w swojej książce poświęconej cybernetyce Pierre de Latil (*Wstęp do cybernetyki*), kwalifikując maszynę cyfrową jako urządzenie o trzecim stopniu automatyzacji. Jednak samochód na przykład został uznany przez de Latila za urządzenie o wyższym stopniu automatyzacji od maszyn cyfrowych, a to dlatego, że dla de Latila cyfrowa maszyna była tylko instrumentem obliczeniowym. Traktowanie jednak maszyny cyfrowej jako wyłącznie instrumentu obliczeniowego jest już przestarzałe. Można dziś zaryzykować twierdzenie, że maszyny cyfrowe są automatami zdolnymi do wykonania najbardziej złożonych czynności, jakie można sobie wyobrazić. Przy użyciu maszyn cyfrowych można modelować wszelkiego rodzaju procesy, systemy sterowania, różne urządzenia analizujące i zwierzęta syntetyczne. . . . "

„. . . Cybernetyka, nauka o sterowaniu i łączności w maszynach, organizmach żywych i społeczeństwach, w znacznej swej części zajmuje się budowaniem modeli różnorakich zjawisk zachodzących bądź w organizmach żywych, bądź w społeczeństwach. Modele częściowe lub całościowe organizmów żywych pozwalają nieraz poznać hipotetyczny mechanizm zjawiska. Ponadto badanie modeli pozwala niejednokrotnie zaplanować doświadczenie, które to z kolei pozwoli rozstrzygnąć, jak dane reakcje zrealizowała przyroda. . ."

„. . . Pozostaje jeszcze do omówienia cel, w jakim budujemy modele układów dynamicznych. Cel ten może być trojaki:

1. Badania zachowania się stabilności układu w zależności od charakteru sterowania.

2. Badanie wpływu poszczególnych wyjść informacyjnych na działania sterowania i ewentualna eliminacja tych wyjść informacyjnych, których wpływ na działanie sterowania można uznać za mały.

3. Badanie wpływu centralnego układu sterowania na działanie różnych grup układów względnie odosobnionych naszego układu dynamicznego i ewentualne opracowanie lokalnego sterowania dla pewnych autonomicznych grup układów względnie odosobnionych.

O ile badania omówione w punkcie 1 są stosunkowo proste i w zasadzie dysponujemy odpowiednimi kryteriami dla tych badań, o tyle dla badań określonych w punktach 2 i 3 nie ma dotychczas jakichś ogólnych kryteriów i wymagają one za każdym razem indywidualnego podejścia. . ."

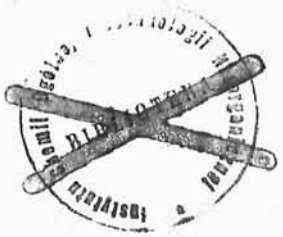
WSTĘP DO PROGRAMOWANIA I MODELOWANIA CYFROWEGO

MAREK GRENIEWSKI

WSTĘP DO PROGRAMOWANIA I MODELOWANIA CYFROWEGO

WARSZAWA 1961
PAŃSTWOWE WYDAWNICTWO NAUKOWE

Obwolutę projektował
Henryk Białoskórski



644

5644

Printed in Poland

ERRATA

Strona	Wiersz od góry	Wiersz od dołu	Jest	Powinno być
72		6	x	$< x >$
120	Rys. 5-4		— 12	= 12
181	kol. 1 15		22-32	24-32
181	kol. 2 10		43	34

SPIS RZECZY

Część trzecia

MODELOWANIE CYFROWE

PRZEDMOWA

Programowane maszyny cyfrowe spowodowały zasadniczy zwrot w zakresie stosowania matematyki przez naukę i technikę. Około piętnastu lat temu problemy obliczeniowe, wymagające wykonania około miliona operacji arytmetycznych, stanowiły granicę ludzkich możliwości, dziś — przy użyciu nowoczesnych programowanych maszyn cyfrowych — rozwiązywane są problemy wymagające miliarda operacji arytmetycznych. Ten olbrzymi rozwój techniki obliczeniowej dał w wyniku postęp wielu dziedzin nauki, techniki i administracji.

Maszyny cyfrowe stały się dla matematyków i inżynierów instrumentem obliczeniowym, działającym analogicznie do biur rachmistrzów. Podobny pogląd reprezentuje w swej książce poświęconej cybernetyce Pierre de Latil(1), kwalifikując maszynę cyfrową jako urządzenie o trzecim stopniu automatyzacji. Jednak samochód na przykład został uznany przez Latila za urządzenie o wyższym stopniu automatyzacji od maszyn cyfrowych, a to dlatego, że dla de Latila cyfrowa maszyna była tylko instrumentem obliczeniowym. Traktowanie jednak maszyny cyfrowej jako wyłącznie instrumentu obliczeniowego jest już przestarzałe. Można dziś zaryzykować twierdzenie, że maszyny cyfrowe są automatami zdolnymi do wykonania najbardziej złożonych czynności, jakie można sobie wyobrazić. Przy użyciu maszyn cyfrowych można modelować wszelkiego rodzaju procesy, systemy sterowania, różne urządzenia analizujące i zwierzęta syntetyczne. Historycznie pierwszym tego rodzaju zastosowaniem maszyn cyfrowych jest wykorzystanie przy automatycznych obliczeniach metod Monte Carlo. Przy użyciu tych metod, maszyna cyfrowa pracuje jako model stochastyczny procesu imitującego dane zjawisko. Drugim tego rodzaju zastosowaniem maszyn cyfrowych jest stosowanie tzw. programów interpretacyjnych bądź kompilacyjnych. Cały ten młody dział nauki został nazwany techniką modelowania cyfrowego i odgrywa coraz większą rolę.

Analizy projektowanych układów sterowania, do niedawna wykonywane na urządzeniach analogowych, coraz częściej dokonywane są na maszynach cyfrowych. Badania cybernetyczne nad zwierzętami syntetycznymi są dzisiaj już wykonywane (za pomocą programów modelujących) na maszynach cyfrowych, odpowiedni program przeorganizowuje maszynę cyfrową na zwierzę syntetyczne. Nie wszyscy zdają sobie sprawę z tego, że homeostat Ashby'ego daje się modelować na maszynie cyfrowej w bardzo prosty sposób.

---

(1) Pierre de Latil, *Sztuczne myślenie. Wstęp do Cybernetyki*, Warszawa 1958, PWT. (tłumaczenie z francuskiego).

Dlatego też programowanie (umiejętność układania algorytmów pracy maszyny cyfrowej) staje się częścią wykształcenia matematyka pracującego nad zastosowaniami matematyki i inżyniera rozwiązującego problemy techniczne.

W książce niniejszej autor postawił sobie za zadanie wprowadzenie przyszłego użytkownika maszyn cyfrowych, matematyka lub inżyniera, w problematykę programowania. Poza systematycznym wykładem elementów programowania, autor przedstawia zarys bardziej skomplikowanych zastosowań programowanych maszyn cyfrowych. Dla zrozumienia niniejszego wykładu wystarcza znajomość matematyki w zakresie wykładu politechnicznego oraz znajomość elementarnych wiadomości o maszynach cyfrowych. Czytelnik, który nie zetknął się dotychczas z maszynami cyfrowymi, powinien przed czytaniem niniejszej książki zapoznać się z którąś z popularnych książek poświęconych maszynom matematycznym, np. książką A. B. Empachera *Maszyny liczą same?*(1).

Wykład oparty jest o przykładową maszynę cyfrową, która tylko w drobnych szczegółach różni się od maszyny EMAL-2, opracowanej przez doc. Romualda Marczyńskiego. Algorytmy w działach arytmetyki binarnej podaję w postaci opracowanej również przez doc. R. Marczyńskiego.

Maszyna cyfrowa EMAL-2 została zbudowana w okresie 1958—1960 przez zespół pracowników Pracowni Maszyn Cyfrowych Zakładu Matematyki Stosowanej Instytutu Badań Jądrowych PAN. W skład zespołu wchodzili: doc. R. Marczyński (kierownik zespołu), mgr inż. K. Bałakier, mgr inż. L. Niemczycki i mgr inż. A. Harland. Autor również starał się być pomocny w tej pracy.

Przedstawione w niniejszej książce metody programowania były opracowywane w Zakładzie Matematyki Stosowanej Instytutu Badań Jądrowych PAN począwszy od kwietnia 1958 r. przez zespół kierowany przez autora. W skład zespołu wchodzili: mgr Wanda Ciechomska-Sawicka, mgr Jerzy Davison, mgr Adam B. Empacher, mgr Zofia Jankowska, mgr Anna Nähr, dr Stefan Paszkowski, mgr Jadwiga Rogińska-Empacherowa, oraz mgr Andrzej Wakulicz, aspirant Instytutu Matematycznego PAN.

Przedstawione dalej metody programowania są ilustrowane programami dla maszyny przykładowej; metody te cechuje ogólność, dzięki której, mutatis mutandis, mogą one być stosowane dla dowolnej maszyny cyfrowej.

Przykład ilustrujący technikę interpretacyjną został rozbudowany bardziej niż tego wymagały potrzeby obliczeniowe, a to dla pełniejszego pokazania budowy programu interpretacyjnego.

Na treść poniższego wykładu wywarli niewątpliwie wpływ prof. dr n. Mieczysław Warmus, doc. Romuald Marczyński oraz członkowie wymienionego wyżej zespołu matematycznego. Niech mi będzie wolno złożyć wszystkim wyżej wymienionym wyrazy wdzięczności.

AUTOR

Warszawa, wrzesień 1960 r.

(1) Warszawa 1959 r., Wydawnictwo Wiedza Powszechna.

Część pierwsza

MASZYNY CYFROWE

1. PROGRAMOWANE MASZYNY CYFROWE

[1.1. Wiadomości podstawowe, 1.2. Przedstawianie liczb, 1.3. Struktura rozkazów]

1.1. WIADOMOŚCI PODSTAWOWE

1.1.1. Maszyny cyfrowe a maszyny analogowe. W dalszym ciągu terminów *urządzenie* i *maszyna* będziemy używali w potocznym znaczeniu, przyjmując, że maszyna jest czymś bardziej skomplikowanym od urządzenia. Od wielu lat w technice obliczeniowej występowały dwa rodzaje urządzeń: urządzenia cyfrowe i urządzenia analogowe. W urządzeniach cyfrowych liczby są przedstawiane za pomocą układów cyfr, skąd dokładność bezwzględna urządzeń cyfrowych jest stała. W urządzeniach analogowych liczby przedstawiane są jako wielkości fizyczne. Najprostszym przykładem urządzenia cyfrowego są liczydła, na których cyfry reprezentowane są za pomocą zespołów krążków. Prostym, powszechnie znanym przykładem urządzenia analogowego jest suwak logarytmiczny; na suwaku logarytmicznym liczby przedstawione są jako odcinki.

Historia techniki obliczeniowej to zarówno historia urządzeń cyfrowych (od arytmometrów Leibniza i Pascala do współczesnych elektronowych maszyn cyfrowych), jak też historia urządzeń analogowych (od suwaka logarytmicznego poprzez planimetry, integratory itp. do współczesnych analogowych analizatorów). Powstanie nowoczesnych elektronowych maszyn cyfrowych pociągnęło za sobą gwałtowny rozwój matematyki obliczeniowej. Powstały nowe metody numeryczne dostosowane do rachunków automatycznych; metody te charakteryzują się wysokim stopniem formalizacji, automatyzacji procesu obliczeniowego odpowiada bowiem formalizacja rachunku.

Rywalizacja bezpośrednio po drugiej wojnie światowej między elektronowymi maszynami analogowymi a elektronowymi maszynami cyfrowymi zakończyła się przewagą cyfrówek.

Wypracowanie odpowiednich metod obliczeniowych jak i rozwój techniki zdecydowały o dużo szerszym zasięgu stosowalności maszyn cyfrowych. Nie wynika stąd, że elektronowe analogi są zbędne. Przeciwnie, analogi przy rozwiązywaniu wielu codziennych problemów techniki odgrywają poważną rolę. Jednakże w wielkich problemach techniki i to nie tylko w problemach czysto obliczeniowych, w potocznym znaczeniu

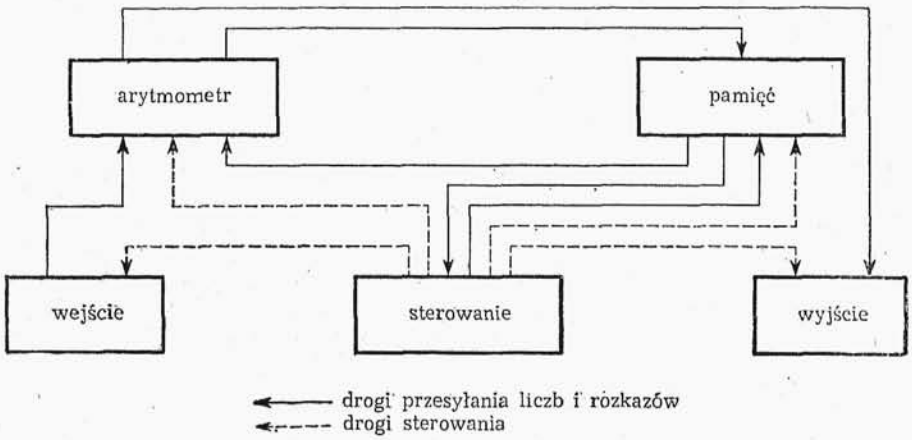
tego słowa, ale i w innych, np. w problemach analizowania i projektowania układów dynamicznych, programowane maszyny cyfrowe zdobyły sobie palmę pierwszeństwa.

W dalszych rozważaniach ograniczymy się wyłącznie do maszyn cyfrowych.

1.1.2. Zasadnicze części programowanej maszyny cyfrowej oraz ich działanie. Uniwersalna programowana maszyna cyfrowa (w skrócie UPMC) składa się z następujących pięciu zespołów:

1) pamięci,
2) układu sterowania,
3) arytmometru,
4) wejścia,
5) wyjścia.

Na rysunku 1-1 przedstawiono strukturę zasadniczych połączeń pomiędzy wyżej wymienionymi zespołami.



Rys. 1-1. Schemat blokowy UPMC

1.1.2.1. Pamięć. *Pamięć* jest to urządzenie służące do przechowywania *słów* (o określonej ilości cyfr), pamięć jest podzielona na komórki, z których każda służy do przechowywania jednego słowa. Wszystkie komórki pamięci są ponumerowane: numery przyporządkowane tym komórkom będziemy nazywali *adresami komórek pamięci* lub krótko *adresami*.

1.1.2.2. Sterowanie. *Sterowanie* kieruje pracą maszyny; działa ono w następujący sposób: rozkazy dla maszyny (czyli instrukcje postępowania maszyny) są zakodowane w postaci liczbowej i zapisane w pamięci maszyny; każdy rozkaz składa się z dwóch części: z części operacyjnej podającej rodzaj czynności, jakie ma wykonać maszyna, i części adresowej mówiącej, na jakich liczbach zapisanych w pamięci ma być wykonana operacja lub gdzie ma być (np. pod jakim adresem) zapisany wynik. O kolejności pobierania rozkazów z pamięci do wykonania decyduje zwykle specjalny rejestr, tzw. licznik rozkazów. W maszynie występują dwa rodzaje rozkazów:

1) rozkazy, które nie zmieniają kolejności wykonywania operacji (np. dodawanie, mnożenie, przesyłanie z komórki do komórki itp.),

2) rozkazy, które służą do zmiany skokowej zawartości licznika rozkazów.

Wykonanie rozkazów z grupy 1 powoduje tylko powiększenie zawartości licznika rozkazów o jeden, wykonanie zaś rozkazów z grupy 2 może spowodować dowolnie założoną zmianę zawartości licznika rozkazów. Wśród rozkazów grupy 2 zasadniczą rolę grają rozkazy, których wykonanie lub niewykonanie jest uzależnione od zachodzenia pewnej ustalonej relacji (np. od tego czy zawartość komórki pamięci o adresie n jest większa od zawartości komórki pamięci o adresie m).

1.1.2.3. Arytmometr. *Arytmometr* służy do wykonywania operacji arytmetycznych i logicznych.

1.1.2.4. Wejście. *Wejście* służy do wprowadzenia liczb i rozkazów do maszyny; jest uruchamiane specjalnym rozkazem.

1.1.2.5. Wyjście. *Wyjście* służy do wyprowadzenia na zewnątrz z maszyny liczb i innych informacji. Jest ono uruchamiane specjalnym rozkazem.

Poza wymienionymi składowymi UPMC należy jeszcze wspomnieć o *pulpicie sterowania*. Jest to urządzenie połączone ze sterowaniem umożliwiające ręczną ingerencję obsługującego maszynę przy wykonywaniu zadań przez maszynę. Przełączniki i przyciski znajdujące się na pulpicie sterowania oddziałują na odpowiednie elementy maszyny umożliwiając ręczne wykonanie szeregu operacji maszyny. Do sprawy tej wrócimy w rozdz. 2 omawiając organizację UPMC wybranej jako maszyny przykładowej dla dalszych rozważań.

W dalszym ciągu będziemy używali wspólnego terminu dla liczb i rozkazów kodowanych cyfrowo. Liczby i rozkazy będziemy nazywali *słowami* (patrz pkt. 1.1.2.1)

Obecnie na świecie zainstalowanych jest kilka tysięcy UPMC. Maszyny te charakteryzują się różnymi parametrami i są stosowane do różnorodnych problemów. Obecnie już jest trudno sklasyfikować w jakiś dokładniejszy sposób istniejące czy też projektowane maszyny cyfrowe ze względu na możliwości obliczeniowe.

Tablica 1-1

Maszyny	Ilość operacji na sekundę
małe	do 1 000
średnie	1 000 ÷ 15 000
duże	15 000 ÷ 100 000
wielkie	powyżej 100 000

Szybki rozwój techniki urządzeń cyfrowych, doskonalenie i miniaturyzacja elementów, pozwala budować coraz szybsze i sprawniejsze maszyny cyfrowe. Klasyfikacja maszyn sprzed kilku lat stały się obecnie zupełnie nieaktualne. Maszyny, które uchodziły jeszcze niedawno za duże, są dziś maszynami średnimi.

Na użytek czytelnika wprowadzimy pewną prowizoryczną klasyfikację maszyn

uniwersalnych przeznaczonych do obliczeń naukowych i technicznych przedstawioną w tabl. 1-1 opartą o ilość operacji wykonywanych przez maszynę w ciągu sekundy.

Klasyfikacja powyższa nie jest doskonała, jednakże oparcie klasyfikacji o więcej czynników, jak pojemność pamięci, prędkość wejść i prędkość wyjść, na ogół nie prowadzi do dokładniejszych rozróżnień. Ponadto powierzchnie zajmowane przez maszyny jak też ilość i rodzaj elementów nie dają się wykorzystać przy klasyfikacji.

1.1.3. Maszyny binarne a maszyny dziesiętne. Wśród współcześnie budowanych i eksploatowanych UPMC rozróżniamy maszyny liczące na liczbach przedstawionych w rozwinięciu dwójkowym oraz maszyny liczące na liczbach przedstawionych w rozwinięciu dziesiętnym.

Maszyny pierwszego typu nazywamy *maszynami binarnymi* w odróżnieniu od maszyn drugiego typu zwanych *maszynami dziesiętnymi*.

Warto podkreślić, że ostatnio (1959 r.) została uruchomiona w Centrum Obliczeniowym Uniwersytetu Moskiewskiego pierwsza na świecie maszyna licząca w systemie trójkowym. Maszynę tę nazwano Sietuń; nie mieści się ona w wyżej przyjętym podziale na maszyny binarne i dziesiętne. Maszynę tę musimy zaliczyć do odrębnej grupy maszyn trójkowych.

1.1.4. Maszyny stałoprzecinkowe a maszyny zmiennoprzecinkowe. *Maszyną stałoprzecinkową* nazywamy UPMC, w której liczby, na których maszyna wykonuje operacje, przybliżają wszystkie liczby z danego przedziału ze stałym maksymalnym błędem bezwzględnym. Natomiast przez *maszyny zmiennoprzecinkowe* będziemy rozumieli wszystkie UPMC nie będące maszynami stałoprzecinkowymi.

1.1.5. Maszyny szeregowe i równoległe. UPMC, w których wszystkie cyfry liczb, na których wykonywane są operacje, przedstawione są za pomocą niezależnych układów, będziemy nazywali *maszynami równoległymi*. UPMC, w których przesyłanie liczb odbywa się za pomocą układów, w których w kolejnych chwilach czasu znajdują się kolejne cyfry dwójkowe liczby przesyłanej, będziemy nazywali *maszynami szeregowymi*.

1.1.6. Najważniejsze typy pamięci stosowanych w UPMC.

1.1.6.1. Pamięć rtęciowa (zwana również pamięcią ultrasoniczną). Jest to historycznie pierwszy masowo stosowany typ pamięci. Pamięć rtęciowa składa się ze sztywnej rury napełnionej rtęcią, zamkniętej na końcach płytkami kwarcu. Przyłożenie do jednej z płytek impulsu elektrycznego wywoła „skurcz" (odkształcenie objętościowe) płytki, który z kolei zainicjuje powstanie podłużnej fali ultrasonicznej w rtęci. Fala ta przenosi się poprzez rtęć i wywołuje nacisk rtęci na płytkę kwarcową, znajdującą się na drugim końcu rury. Nacisk ten powoduje chwilowe odkształcenie płytki kwarcowej, która z kolei daje, pod wpływem tego odkształcenia, impuls elektryczny. Impuls ten jest wzmocniony i przekazany na wejściową płytkę kwarcową itp. Długość rury jest tak dobrana, aby w omówionym wyżej układzie mogły być zapamiętane całe serie impulsów.

Impulsy zapamiętywane w wyżej opisanym układzie pamięciowym można pobierać tylko wtedy, gdy znajdują się one na wyjściu z rury. Dlatego też przy pobraniu z pamięci rtęciowej słowa trzeba odczekać pewien czas, aż słowo to pojawi się na wyjściowej płytce kwarcowej. Czas ten będziemy nazywali *czasem oczekiwania*. Ze względów eksploata-

cyjnych interesować nas będą dwie wielkości: średni czas oczekiwania, równy połowie czasu opóźnienia uzyskiwanego w rurze rtęciowej i tzw. długość słowa w jednostkach czasu.

Pamięć rtęciowa ma szereg wad, które zdecydowały o zaniechaniu jej stosowania. Są to: stosunkowo powolne działanie, duże wymiary, znaczny wpływ temperatury na prawidłowość działania pamięci, duża czułość na zakłócenia zewnętrzne. Ponadto pamięć rtęciowa jest niesłychanie czuła na wstrząsy, które łatwo mogą spowodować uszkodzenie płytek kwarcowych.

1.1.6.2. Pamięć magnetostrykcyjna. Pamięć ta podobnie jak omawiana w punkcie 1.1.6.1. pamięć rtęciowa wykorzystuje zjawisko rozchodzenia się fali akustycznej w metalu, dokładniej w drucie niklowym. W odróżnieniu jednak od pamięci rtęciowej, wykorzystany tu został efekt rozchodzenia się fali poprzecznej. Nazwa pamięci magnetostrykcyjnej pochodzi od zjawiska magnetostrykcji, wykorzystanego dla wywołania poprzecznej fali w pręcie niklu. Wejściem do pamięci jest cewka nawinięta na jednym końcu drutu niklowego, wewnątrz której powstaje pole magnetyczne wywołane przez przyłożony impuls elektryczny. Pod wpływem pola magnetycznego w części pręta, znajdującego się wewnątrz cewki, powstaje „skurcz" (fala poprzeczna). Fala ta przesuwa się wzdłuż pręta, ruchowi fali towarzyszy ruch pola magnetycznego. W umieszczonej na drugim końcu pręta cewce, pod wpływem pola magnetycznego jest indukowany prąd, który z kolei poprzez wzmacniacz jest przyłożony na cewkę wejściową. Ponieważ jednak tłumienie akustycznej fali poprzecznej w niklu jest duże, opracowanie więc tej pamięci nie było proste. Obecnie pamięć niklowa jest używana przez angielską firmę Ferranti w produkowanych przez nią maszynach (Pegasus I i II, Syrius itp.).

1.1.6.3. Pamięć ferrytowa. Oparta jest na zjawisku histerezy w materiałach ferromagnetycznych. Pamięć taka zbudowana jest z ferrytowych lub permalloiowych toroidalnych rdzeni, z których każdy jest wykorzystany do zapamiętania jednej cyfry binarnej (dwójkowej). Istnieją rozwiązania, w których dla zapamiętania jednej cyfry binarnej używane są dwa rdzenie ferrytowe. Rozwiązania takie mają lepsze parametry pracy, ze względu na równe obciążenia. Rozróżniamy dwa rodzaje pamięci ferrytowych: pamięci równoległe, używane w dużych i wielkich maszynach cyfrowych, jak radzieckie maszyny BESM-II i M-20, amerykańskie maszyny IBM 704, IBM 709, Larc, Lincoln TX-2 itp. oraz pamięci ferrytowe szeregowe, używane w małych maszynach cyfrowych, jak zachodnioniemiecka maszyna Zuse Z-22.

Pokrótce omówimy zasady działania równoległej pamięci ferrytowej. Jak już wspominaliśmy, pamięć taka jest zbudowana z toroidalnych rdzeni magnetycznych o prawie prostokątnej pętli histerezy. Rdzeń taki ma dwa stany nasycenia magnetycznego, różniące się względem siebie znakiem. Jeśli przez uzwojenie nawinięte na rdzeniu przepuścimy dostatecznie duży prąd elektryczny, to pole magnetyczne w rdzeniu przyjmie jeden z dwóch stanów, w zależności od kierunku przepływu prądu. Jednemu z tych stanów przyporządkowujemy cyfrę binarną „0", a drugiemu — cyfrę binarną „1". Pamięć równoległa składa się z matryc złożonych z rdzeni; ilość matryc odpowiada ilości cyfr binarnych w liczbach, na których dana UMPC wykonuje działania. Ilość rdzeni w każdej z matryc odpowiada pojemności pamięci ferrytowej. Na każdym rdze-

niu w matrycy znajdują się trzy uzwojenia. Jedno z tych uzwojeń połączone jest szeregowo z analogicznymi uzwojeniami na wszystkich rdzeniach w danym wierszu. Drugie uzwojenie połączone jest szeregowo z analogicznymi uzwojeniami na wszystkich rdzeniach w danej kolumnie. Trzecie uwzojenie połączone jest szeregowo z analogicznymi uzwojeniami na wszystkich rdzeniach danej matrycy.

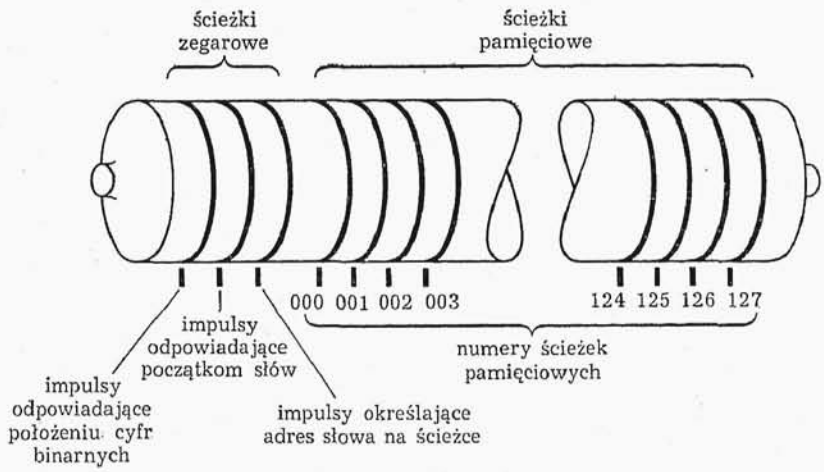
Dla zapisania „1" na rdzeniu będącym na przecięciu wybranego wiersza i kolumny (w danej matrycy) wystarczy przyłożyć do przewodów łączących uzwojenia rdzeni, znajdujących się w danym wierszu, impuls wytwarzający pole magnetyczne o natężeniu H_0, gdzie H_0 jest tak dobranym natężeniem, że nie powoduje zmiany kierunku pola magnetycznego w rdzeniu, natomiast natężenie $2H_0$ wystarczy już do zmiany kierunku pola magnetycznego w rdzeniu. Podobnie do przewodu łączącego uzwojenia rdzeni w danej kolumnie przykładamy impuls wytwarzający pole magnetyczne o natężeniu H_0. Jeśli impulsy te przyłożyliśmy jednocześnie, to w rdzeniu będącym na przecięciu wybranego wiersza i wybranej kolumny zostaje zapisana „1" (oczywiście, o ile oba uzwojenia mają tę samą kierunkowość).

Dla odczytania cyfry binarnej zapisanej na przecięciu wybranego wiersza i wybranej kolumny, podobnie jak dla zapisania, przykładamy impulsy do przewodów łączących odpowiednie uzwojenia, z tym że impulsy te muszą mieć napięcia przeciwne do napięcia impulsu zapisującego „1", a to w celu wytworzenia pola o natężeniu $2H_0$ ale przeciwnie skierowanego. W przypadku gdy na wybranym rdzeniu była zapisana jedynka, w trzecim uzwojeniu (przechodzącym przez wszystkie rdzenie matrycy) powstaje impuls.

Pamięć ferrytowa w odróżnieniu od omawianych poprzednio pamięci rtęciowej i magnetostrykcyjnej jest pamięcią statyczną i czas dostępu do każdego słowa zapisanego w tej pamięci jest stały. Równa się on czasowi przełączenia układów wybierających z pamięci. Czas dostępu do pamięci ferrytowej waha się od 0,4 μs do kilku μs w zależności od techniki układów elektronowych, własności materiałów magnetycznych i rozmiaru geometrycznego rdzeni ferrytowych. Ponadto pamięć ferrytowa pamięta w sposób ciągły, tzn. jeśli wyłączymy maszynę z sieci i ponownie ją włączymy, to zawartość pamięci nie ulegnie wymazaniu, jak w przypadku pamięci rtęciowej czy też magnetostrykcyjnej. Pamięć ferrytowa jest obecnie jedynym typem szybkiej pamięci, masowo stosowanym w maszynach cyfrowych.

1.1.6.4. Pamięć bębnowa. Pamięć ta wykorzystuje podobnie jak pamięć ferrytowa zjawisko histerezy w materiałach ferromagnetycznych. Pamięć bębnowa składa się z bębna pokrytego warstwą materiału ferromagnetycznego, wirującego ze stałą prędkością, i głowic czytająco-piszących (w niektórych rozwiązaniach bywają używane pary głowic, z których jedna jest głowicą czytającą, a druga głowicą piszącą). Zapisywanie zer lub jedynek polega na przemagnesowaniu małego wycinka powierzchni bębna w jednym z dwóch kierunków. Czytanie polega na badaniu, w którą stronę wycinek powierzchni bębna został namagnesowany. Każda z głowic czytająco-piszących jest umieszczona na stałe na pewnym pasie powierzchni magnetycznej bębna. Pasy takie będziemy dalej nazywali ścieżkami. Każda ścieżka podzielona jest na części odpowiadające pojedynczym słowom. Współrzędne tych słów i poszczególnych cyfr binarnych,

z których złożone są słowa, mogą być określone za pomocą tzw. ścieżek zegarowych. Na ścieżkach zegarowych zapisane są na stałe impulsy, odpowiadające bądź położeniu poszczególnych cyfr binarnych, bądź początkom słów, bądź wreszcie zakodowane są adresy poszczególnych słów na ścieżce. Na rysunku 1-2 pokazany jest schematyczny rysunek bębna magnetycznego. Podobnie jak w przypadku pamięci rtęciowej i magnetostrykcyjnej, pamięć bębnowa ma czas oczekiwania. Średni czas oczekiwania równa



Rys. 1-2. Schemat bębna magnetycznego

się czasowi połowy obrotu bębna. Dla bębna wirującego z prędkością 100 obr/s. średni czas oczekiwania wynosi 5 ms. W dużych maszynach pamięć bębnowa jest pamięcią pomocniczą. W małych maszynach bądź jest jedyną pamięcią, bądź jest używana równocześnie z małą pamięcią szybką. Pamięć bębnowa w różnych odmianach jest stosowana w większości UPMC. Obok szybkiej pamięci ferrytowej jest to powszechnie stosowany typ pamięci.

1.1.7. Urządzenia zewnętrzne UPMC. Wejścia, wyjścia oraz pomocnicze urządzenia pamięciowe, jak urządzenia z taśmami magnetycznymi, obejmujemy wspólną nazwą *urządzeń zewnętrznych*. Urządzenia zewnętrzne w małych i średnich maszynach oparte są w zasadzie bądź o urządzenia działające na taśmie dziurkowanej (np. pięciokanałowej taśmie dalekopisowej) i wykorzystujące typowe urządzenia dalekopisowe, łącznie ze specjalnymi szybkimi czytnikami i reperforatorami taśmy dalekopisowej, bądź o urządzenia na karty dziurkowane. Wyjątek w tym zakresie stanowią radzieckie maszyny typu Urał (Urał I, Urał II, Kristał i Pogoda), które korzystają ze specjalnych taśm celuloidowych 11-kanałowych oraz specjalnych drukarek równoległych (drukujących jednocześnie 16 cyfr dziesiętnych). W dużych i wielkich maszynach cyfrowych, wejścia i wyjścia są wąskim gardłem. Olbrzymiej prędkości wewnętrznej pracy maszyny odpowiada wielokrotnie mniejsza sprawność wejść i jeszcze niższa sprawność wyjść. Najlepsze rozwiązanie mechaniczne drukarek — wyjść dla dużych i wielkich UPMC (tzw. drukarki drukujące w biegu) — nie uzyskują większych prędkości niż

dwadzieścia kilka wierszy na sekundę. W praktyce nawet na drukarkach drukujących w wierszu 100 znaków, można wykorzystać od dwudziestu do trzydziestu znaków w wierszu. Stąd wynika, że na najlepszych mechanicznych drukarkach szybkich można praktycznie drukować nie więcej niż 600 znaków/s. Prędkość taka w porównaniu z prędkością maszyny jest mała i bardzo zmniejsza praktyczną prędkość pracy UPMC. Dlatego też stosuje się systemy kombinowane wyjść i wejść, gdzie wprowadzanie i wyprowadzanie odbywa się poprzez dodatkowe urządzenia z taśmami magnetycznymi.

Budowanie szybszych urządzeń mechanicznych wydaje się niemożliwe, natomiast w ostatnich latach opracowano pierwsze drukarki szybkie, wykorzystując zjawiska kserografii i elektrografii. Urządzenia takie dają daleko większe prędkości drukowania (rzędu 300 wierszy/s.). Jak dotychczas, koszt tego typu urządzeń jest duży.

1.1.8. Zasadnicze typy UPMC, ze względu na zastosowanie. Uniwersalne programowane maszyny cyfrowe możemy obecnie podzielić na trzy typy:

1) maszyny do przetwarzania danych,
2) maszyny uniwersalne do obliczeń naukowych i technicznych,
3) maszyny dla celów programowanego sterowania.

Postaramy się w możliwie krótki sposób o scharakteryzowanie każdego z tych typów:

1. Maszyny do przetwarzania danych stosowane są do zagadnień, w których stosunek ilości danych do ilości wykonywanych operacji jest rzędu 1/100. Maszyna taka jest przystosowana do wykonywania małej ilości prostych działań arytmetyczno-logicznych i przesyłania dużej ilości informacji między poszczególnymi miejscami pamięci. Dlatego też maszyny do przetwarzania danych są często maszynami dziesiętnymi. Pamięć w takiej maszynie jest bardzo rozbudowana, może mieć ona jednakże duże czasy oczekiwania. Ponadto maszyna taka musi mieć możliwość masowego wyprowadzania wyników na bieżąco zarówno dla dalszego przekształcenia, jak również na bezpośredni użytek zewnętrzny (np. taśma magnetyczna i drukarka).

2. Maszyna uniwersalna do obliczeń naukowych i technicznych charakteryzuje się dużym stosunkiem ilości operacji do danych wejściowych. Są to na ogół maszyny, które wykonują wielkie ilości złożonych operacji; pamięć wewnętrzna w takich maszynach ma czas oczekiwania stosunkowo mały.

3. Maszyny do celów programowanego sterowania rozwiązują problemy na bieżąco zwykle w czasie rzeczywistym przebiegania danego zjawiska. Dane wejściowe dostarczane są przeważnie w sposób ciągły (1). Są to maszyny bardzo szybkie o dużej pojemności pamięci. Maszyny te zwykle pracują dla potrzeb sterowania, oczywiście mają one specjalne systemy wyjścia i wejścia.

Należy podkreślić, że UPMC typu 2 i 3 są z zasady maszynami binarnymi.

Maszyny do przetwarzania danych odgrywają wielką rolę w życiu gospodarczym wielu krajów. Na Zachodzie maszyn do przetwarzania danych jest mniej więcej dziesięciokrotnie więcej niż maszyn do obliczeń naukowych i technicznych. Istnieją systemy orga-

(1) W tym przypadku w skład urządzeń zewnętrznych wchodzą specjalne urządzenia zwane konwerterami, służące do zamiany wielkości funkcyjnych (ciągłych) na wielkości dyskretne (cyfrowe).

nizacji bankowej, handlowej, przemysłowej całkowicie oparte na elektronowej technice cyfrowej.

Pod względem zastosowania można dokonać podziału maszyny typu 1, na trzy grupy:

1) maszyny współpracujące z istniejącą organizacją biurową przedsiębiorstw (np. z kartami dziurkowanymi, z księgowością ręczną itp.),

2) maszyny realizujące pełną automatyzację elektronową biura,

3) maszyny wykorzystywane w celach analitycznych — kalkulacyjnych (planowanie gospodarcze, planowanie zaopatrzenia i in.).

Użycie maszyn cyfrowych w biurach handlowych nawet przy dużym ich koszcie jest ekonomicznie uzasadnione w granicach 5÷7 lat amortyzacji, dając ponadto pewną rezerwę czasową na potrzeby dodatkowych prac np. analitycznych.

W ramach współpracy z istniejącą organizacją w biurach (punkt 1) maszyny sporządzają listy płacy, prowadzą księgowość finansową, materiałową, przeprowadzają inwentaryzacje, kalkulację kosztów, analizę zapotrzebowania i zbytu. Koszty inwestycyjne takich zespołów wahają się od 500 000 do 2 000 000 dolarów USA. Koszt maszyny cyfrowej w takim zestawie stanowi 20÷40% kosztów całkowitych.

W dalszym etapie automatyzacji (punkt 2) w bankach, instytucjach ubezpieczeniowych oraz centralach handlowych i przemysłowych, wszelkie operacje, włącznie z drukowaniem wyciągów, polis i zapotrzebowań, wykonywane są całkowicie przez zespół urządzeń pracujących z niezwykłą niezawodnością. Koszt inwestycji zautomatyzowanego biura wynosi od 2 do kilkunastu milionów dolarów. Na pozycje te składa się przede wszystkim koszt urządzeń wyjściowych drukujących, taśmy magnetycznej dla wprowadzania i przechowywania danych; koszt samej maszyny cyfrowej pochłania około 10% całej sumy.

1.2. PRZEDSTAWIANIE LICZB

1.2.0. Jak już wspominaliśmy w punkcie 1.1.3, ze względu na przedstawienia liczb wyróżniamy dwa rodzaje UPMC, a mianowicie maszyny binarne i maszyny dziesiętne. Podział taki jest jeszcze bardzo powierzchowny, ponieważ w praktyce dysponujemy niejedną arytmetyką binarną, ale co najmniej czterema takimi arytmetykami. Podobnie wygląda sprawa z maszynami dziesiętnymi. W dziesiętnych maszynach cyfrowych cyfry dziesiętne są kodowane za pomocą kilku cyfr binarnych (co najmniej czterech). Kodów takich można budować wiele, w każdym z nich inaczej wykonywane są działania arytmetyczne. Obecnie omówimy kolejno sposoby przedstawiania liczb w najważniejszych arytmetykach stosowanych w maszynach cyfrowych, ograniczając się do arytmetyk stałoprzecinkowych przedziału [-1,1].

1.2.1. Arytmetyka binarna prosta: Liczby dodatnie x są postaci

$$x = \sum_{i=1}^{n} \alpha_i \cdot 2^{-i}, \qquad \text{gdzie} \qquad \alpha_i \, e \, \{0,1\}, \tag{1-1}$$

reprezentowane są w maszynie jako

$$0\alpha_1\alpha_2 \ldots \alpha \tag{1-2}$$

natomiast liczby ujemne $-x$ mają postać

$$1 + \sum_{i=1}^{n} \alpha_i \cdot 2^{-i}; \tag{1-3}$$

reprezentowane są w maszynie jako

$$1\alpha_1\alpha_2 \ldots \alpha_n. \tag{1-4}$$

Arytmetyka binarna prosta jest dostosowana do maszyn równoległych; stosowanie jej w maszynach szeregowych prowadzi bądź do zmniejszania prędkości pracy arytmometru, bądź do jego rozbudowy. Niewygodne jest ponadto występowanie dwu symboli zera: plus zero i minus zero.

1.2.2. Arytmetyka binarna negacyjna. Podobnie jak w punkcie 1.2.1, liczby dodatnie mają postać (1-1) i są reprezentowane w maszynie przez wyrażenie typu (1-2). Liczby ujemne $-x$ mają postać

$$1 + \sum_{i=1}^{n} (1-\alpha_i)\, 2^{-i}, \quad \text{gdzie } \alpha_i \in \{0,1\}, \tag{1-5}$$

reprezentowane są zaś w maszynie jako

$$1\alpha'_1\alpha'_2 \ldots \alpha'_n, \quad \text{gdzie} \quad \alpha'_i = (1 - \alpha_i). \tag{1-6}$$

Arytmetyka binarna negacyjna, podobnie jak arytmetyka binarna prosta, ma dwa wyrażenia na zero: „plus zero" — wyrażenie złożone z samych zer — oraz „minus zero" — wyrażenie złożone z samych jedynek.

Obie omawiane wyżej arytmetyki są w wielu maszynach stosowane wspólnie, a mianowicie w pewnych częściach maszyny, np. w rejestrze, zwanym akumulatorem, liczby są przedstawione w arytmetyce negacyjnej, w pamięci zaś liczby są przedstawione w arytmetyce prostej. Zmiana przedstawienia następuje automatycznie przy przesyłaniu liczb z jednej części maszyny do drugiej. Rozwiązanie takie jest spowodowane uproszczeniem pewnych algorytmów działań przy takim podwójnym przedstawieniu.

1.2.3. Arytmetyka binarna uzupełnieniowa. Podobnie jak w punktach 1.2.1 i 1.2.2, liczby dodatnie mają postać (1-1) i są reprezentowane w maszynie przez wyrażenie typu (1-2). Liczby ujemne $-x$ przedstawiamy jako uzupełnienie do 2 liczby x, skąd zresztą nazwa arytmetyki,

$$-x \text{ odpowiada } 2-x. \tag{1-7}$$

Szczegółowo omówimy binarną arytmetykę uzupełnieniową w punkcie 2.1. Należy tu podkreślić, że arytmetyka ta ma pewną niesymetrię przedziału w odróżnieniu od arytmetyk omawianych w punktach 1.2.1 i 1.2.2. Mianowicie w binarnej arytmetyce uzupełnieniowej liczby przebiegają przedziały $<-1, 1-2^{-n}>$. W praktyce obliczeniowej pozbywamy się tej niesymetrii przez niekorzystanie z lewego końca przedziału „-1". Binarna arytmetyka uzupełnieniowa jest stosowana w większości maszyn szeregowych, gdzie pozwala ona na nierozróżnianie pozycji uzupełnienia od pozostałych pozycji rozwinięcia przy dodawaniu pary liczb.

1.2.4. Arytmetyka minus binarna. W odróżnieniu od omawianych dotychczas przedstawień liczb, liczby dodatnie w tej arytmetyce nie mają tak regularnego przedstawienia. Liczby dodatnie i ujemne x mają wspólną postać

$$x = -\sum_{i=1}^{n+1} \alpha_i (-2)^{-i}, \qquad \alpha_i \in \{0,1\}. \tag{1-8}$$

To pozorne uproszczenie (wspólne przedstawienie liczb dodatnich i ujemnych i prostota algorytmów dodawania i mnożenia) ma istotną wadę. Algorytm badania znaku liczby jest stosunkowo trudny w porównaniu z odpowiednim algorytmem w trzech poprzednio omawianych przedstawieniach, gdzie wystarczyło sprawdzić, jaka cyfra binarna znajduje się na najbardziej znaczącej pozycji. Ponadto w arytmetyce tej występuje bardzo poważna niesymetria przedziału (patrz Pawlak, Wakulicz [15]).

1.2.5. Przedstawianie liczb dziesiętnych za pomocą czterocyfrowych kodów binarnych. Wszystkie czterocyfrowe kody binarne cyfr dziesiętnych N mają postać:

$$N = \alpha_1 g_1 + \alpha_2 g_2 + \alpha_3 g_3 + \alpha_4 g_4 + a, \tag{1-9}$$

gdzie $\alpha_i \in \{0,1\}$ dla $i = 1,2,3,4$, zaś g_1, g_2, g_3, g_4 jest to tzw. baza kodu, a — tzw. akces kodu.

W tablicy 1-2(1) podane są wartości g_1, g_2, g_3, g_4 dla bardziej znanych kodów o $a = 0$.

Tablica 1-2

g_1	g_2	g_3	g_4	g_1	g_2	g_3	g_4	g_1	g_2	g_3	g_4	g_1	g_2	g_3	g_4	g_1	g_2	g_3	g_4
5	2	1	1	6	4	2	1	7	4	2	−1	8	4	3	−2	6	5	2	−4
4	3	1	1	7	4	2	1	8	4	2	−1	6	2	1	−3	6	5	3	−4
5	3	1	1	8	4	2	1	6	2	1	−2	7	2	1	−3	6	4	3	−5
6	3	1	1	5	3	1	−1	5	3	1	−2	7	5	1	−3	7	5	3	−6
4	2	2	1	6	3	1	−1	6	3	1	−2	5	4	2	−3	6	3	−1	−1
5	2	2	1	5	2	2	−1	7	3	1	−2	6	4	2	−3	6	3	−2	−1
6	2	2	1	6	2	2	−1	4	4	1	−2	8	4	2	−3	5	4	−2	−1
3	3	2	1	4	3	2	−1	5	4	1	−2	6	2	1	−4	6	4	−2	−1
4	3	2	1	5	3	2	−1	6	4	1	−2	7	2	1	−4	7	4	−2	−1
5	3	2	1	6	3	2	−1	8	4	1	−2	8	2	1	−4	8	4	−2	−1
6	3	2	1	7	3	2	−1	6	3	2	−2	7	5	1	−4	7	2	−3	−1
7	3	2	1	4	4	2	−1	4	4	3	−2	8	6	1	−4	7	2	−4	−1
4	4	2	1	5	4	2	−1	5	4	3	−2	6	3	2	−4	8	4	−3	−2
5	4	2	1	6	4	2	−1	6	4	3	−2	8	3	2	−4	8	7	−4	−2

W tablicy 1-3(1) podana jest postać kolejnych cyfr dziesiętnych dla czterech kodów.

Spośród kodów $a \neq 0$ zasługuje na uwagę kod tzw. akces trzy, dla którego $g_1 = 8$, $g_2 = 4$, $g_3 = 2$, $g_4 = 1$, zaś $a = 3$. W tablicy 1-4 podana jest postać kolejnych cyfr dziesiętnych w kodzie akces trzy.

(1) Tablice 1-2 i 1-3 zostały zaczerpnięte z książki R. K. Richardsa (Bibliografia [17]).

Tablica 1-3

	8 4 2 1	2 4 2 1	5 4 2 1	7 5 3 −6
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1	0 0 0 1	1 0 0 1
2	0 0 1 0	0 0 1 0	0 0 1 0	0 1 1 1
3	0 0 1 1	0 0 1 1	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 0 0	0 1 0 0	1 0 1 1
5	0 1 0 1	1 0 1 1	1 0 0 0	0 1 0 0
6	0 1 1 0	1 1 0 0	1 0 0 1	1 1 0 1
7	0 1 1 1	1 1 0 1	1 0 1 0	1 0 0 0
8	1 0 0 0	1 1 1 0	1 0 1 1	0 1 1 0
9	1 0 0 1	1 1 1 1	1 1 0 0	1 1 1 1

Tablica 1-4

Cyfry dziesiętne	Kod cyfr dziesiętnych	Cyfry dziesiętne	Kod cyfr dziesiętnych
0	0 0 1 1	5	1 0 0 0
1	0 1 0 0	6	1 0 0 1
2	0 1 0 1	7	1 0 1 0
3	0 1 1 0	8	1 0 1 1
4	0 1 1 1	9	1 1 0 0

1.3. STRUKTURA ROZKAZÓW

W UPMC rozkazy składają się z dwu części: z części operacyjnej określającej rodzaj czynności wykonywanych przez dany rozkaz i z części adresowej podającej bądź adresy (lub adres) komórek pamięci, na których (lub na której) zawartości ma być wykonywana dana operacja, bądź pewne pomocnicze parametry operacji, bądź jedno i drugie.

Przez liczbę adresów będziemy rozumieli maksymalną liczbę adresów, jaka występuje w rozkazach arytmetycznych danej maszyny. Wśród współczesnych UPMC około 50% to maszyny o liczbie adresów jeden, tzw. maszyny jednoadresowe, na drugim miejscu znajdują się maszyny o liczbie adresów trzy, tzw. maszyny trójadresowe, na trzecim miejscu znajdują się maszyny o liczbie adresów dwa, tzw. maszyny dwuadresowe, na czwartym miejscu znajdują się pozostałe maszyny o liczbie adresów cztery (tzw. maszyny czteroadresowe) albo więcej.

W naszych rozważaniach ograniczymy się jedynie do UPMC z licznikiem rozkazów o liczbie adresów jeden, dwa i trzy(1).

(1) Istnieją UPMC o liczbie adresów nie mniejszej od dwóch, nie mające licznika rozkazów. W maszynach tych jeden z adresów rozkazu wskazuje adres następnego rozkazu, który ma być wykonany, po wykonaniu bieżącego rozkazu.

Wprowadzimy następujące oznaczenia: A — rejestr podstawowy arytmometru, a, b, c — adresy komórek pamięci. Liczby zapisane w rejestrze podstawowym lub w komórkach pamięci będziemy dalej nazywali zawartością rejestru podstawowego czy też zawartością pamięci. Zawartość komórki pamięci o adresie a będziemy oznaczali (a), podobnie zawartość rejestru A, będziemy oznaczali (A).

Na ogół jednoadresowe rozkazy arytmetyczne mają postać:

$$\left.\begin{aligned} (A) \bigcirc (a) &\to A, \\ g(a) &\to A, \\ g(A) &\to a, \\ g(A) &\to A, \end{aligned}\right\} \tag{1-10}$$

gdzie przez „→" oznaczyliśmy przesyłanie zawartości pod adres, a przez „○" i „g" rodzaje operacji wykonywane na zawartościach rejestru podstawowego arytmometru lub zawartości dowolnej komórki pamięci.

Operacja mnożenia w większości jednoadresowych maszyn ma nieco inną strukturę

$$(M) \cdot (a) \to A, \tag{1-11}$$

gdzie M oznacza pomocniczy rejestr arytmometru, tzw. rejestr mnożnej.

W tym przypadku wraz z operacją mnożenia związana jest pomocnicza operacja przesyłania do rejestru M

$$(a) \to M. \tag{1-12}$$

Na przykład jednoadresowe są następujące maszyny radzieckie: Urał I, Urał II, Urał IV, M-180; amerykańskie: IBM 701, IBM 704, IBM 709.

Typowymi arytmetycznymi rozkazami dwuadresowej maszyny są rozkazy

$$\left.\begin{aligned} (a) \bigcirc (b) &\to A, \\ (a) \bigcirc (A) &\to b, \\ (a) \to A &\to b. \end{aligned}\right\} \tag{1-13}$$

Każdy z tych rozkazów można przedstawić za pomocą pary rozkazów jednoadresowych

$$\left.\begin{aligned} (a) &\to A \\ (A) \bigcirc (b) &\to A \end{aligned}\right\} \quad \left.\begin{aligned} (A) \bigcirc (a) &\to A \\ (A) &\to b \end{aligned}\right\} \quad \left.\begin{aligned} (a) &\to A \\ (A) &\to b \end{aligned}\right\},$$

Przykładami maszyn dwuadresowych są: radziecka maszyna M-3 i amerykańska maszyna ERA-1103.

W maszynach trójadresowych operacje arytmetyczne mają postać

$$(a) \bigcirc (b) \to c; \tag{1-14}$$

co równoważne jest trzem rozkazom jednoadresowym

$$\left.\begin{aligned} (a) &\to A, \\ (A) \bigcirc (b) &\to A, \\ (A) &\to c. \end{aligned}\right\} \tag{1-15}$$

Przykładami maszyn trójadresowych są: radzieckie maszyny BESM-II, M-20 i amerykańskie Bizmac, Datematic-1000, Ramac (IBM-305).

Przedstawione wyżej struktury rozkazów maszyn jedno-, dwu- i trójadresowych należy uznać za typowe. Struktury odmienne, np. maszyny trójadresowej o arytmometrze z rejestrem i rozkazach arytmetycznych postaci

$$\begin{gathered}(A) \bigcirc (a) \bigcirc (b) \rightarrow c, \\ (A) \bigcirc (a) \bigcirc (b) \bigcirc (c) \rightarrow A, \\ (a) \bigcirc (b) \bigcirc (c) \rightarrow A\end{gathered} \tag{1-16}$$

należy uznać za nietypowe. Rozkazy powyższego typu nie dają, mimo pozorów, ułatwień obliczeniowych, a stwarzają natomiast wiele trudności przy programowaniu. Czytelnik po zapoznaniu się z rozdz. 6 i 7 zrozumie, że proces automatyzacji kodowania, lub programowania jest prosty, jeśli rozkazy maszyny mają prostą, możliwie jednolitą strukturę.

Z punktu widzenia ekonomicznego wykorzystania pamięci i właściwej dokładności obliczeń w UPMC o słowach stałej długości ważny jest właściwy dobór długości rozkazów i liczb. Jak wykazało doświadczenie, ze względu na praktyczną prędkość liczenia i wykorzystywaną objętość pamięci, istnieje optimum tylko dla maszyn jedno- i trójadresowych. W maszynach dwuadresowych sprawy tej nie można rozwiązać; przy dopasowaniu długości słowa do rozkazu otrzymamy słowa 30-bitowe (1) (6 bitów — kod operacji, 12 bitów — pierwszy adres, 12 bitów — drugi adres), dla celów obliczeniowych 30 bitów jest dokładnością zbyt małą. W obecnie budowanych małych i średnich maszynach długość liczb waha się od 33 do 42 bitów.

Przy rozważaniu maszyn jednoadresowych ograniczymy się, przykładowo, do długości liczby około 40 bitów. W maszynie jednoadresowej możemy tę długość ekonomicznie wykorzystać, przyjmując, że rozkazy są o połowę krótsze od liczb. Będziemy więc mieli do czynienia z dwoma rodzajami słów, długimi 40 bitów i krótkimi 20 bitów. W większości maszyn jednoadresowych o rozkazach mających długość o połowę mniejszą od liczb pamięć jest podzielona na komórki odpowiadające słowom krótkim, których pary tworzą z kolei komórki długie (w większości maszyn jednoadresowych odczytywać i zapisywać można w pamięci zarówno słowa krótkie, jak i długie). W wyżej omawianych maszynach jednoadresowych rozkaz długości 20 bitów ma na ogół budowę następującą: 6 bitów — kod operacji, 13 bitów — adres, 1 bit — znak mówiący o tym, czy adres jest adresem słowa krótkiego, czy długiego.

W maszynach trójadresowych przy tej samej pojemności pamięci 2^{12} słów, ekonomicznie możemy wykorzystać 42 bitowe słowa (6 bitów kod operacji, 12 bitów pierwszy adres, 12 bitów drugi adres i 12 bitów trzeci adres).

W większości produkowanych obecnie maszyn cyfrowych znajdują się specjalne rejestry, zwane B-rejestrami bądź modyfikatorami, służące do automatycznej zmiany adresu wykonywanego rozkazu. W UPMC z B-rejestrami rozkaz ma poza tym specjalną część określającą adres B-rejestru. Na przykład w maszynie jednoadresowej z mody-

(1) Bit — cyfra binarna, inaczej zwana cyfrą dwójkową, skrót od słów „binary digit".

fikatorami rozkaz składa się z trzech części: części operacyjnej K, części adresowej a i części p podającej numer modyfikatora. Sterowanie wypełnia taki rozkaz w następujący sposób: część operacyjna K jest realizowana zgodnie z czynnością, którą reprezentuje, korzystając z adresu równego $a + (p)$. We współczesnych UPMC ilość B-rejestrów waha się od jednego do siedmiu (w maszynach binarnych), do dziewięciu (w maszynach dziesiętnych).

W. S. Linski (Bibliografia [10]) przeprowadził, opierając się na materiale statystycznym, analizę stopnia wykorzystania pamięci przez program w maszynach jedno-, dwu- i trójadresowych oraz analizę czasu potrzebnego na rozwiązanie tego samego zagadnienia na maszynie jedno-, dwu- i trójadresowej. Analiza powyższa została przeprowadzona przy założeniu, że czasy wykonania przez arytmometr operacji oraz czasy pobierania i zapisywania w pamięci w rozważanych jedno-dwu- i trójadresowych maszynach są równe.

Oznaczając przez M_1 ilość rozkazów w programie dla rozwiązania danego zagadnienia na maszynie jednoadresowej, przez M_2 ilość rozkazów w programie dla maszyny dwuadresowej, przez M_3 ilość rozkazów w programie dla maszyny trójadresowej, Linski ustalił następujące nierówności:

$$M_3 \leqslant M_2 \leqslant M_1, \tag{1-17}$$

$$1{,}3 \leqslant \frac{M_1}{M_3} \leqslant 1{,}85, \tag{1-18}$$

przy czym wartość średnia M_1/M_3 wynosi 1,52,

$$1{,}05 \leqslant \frac{M_1}{M_2} \leqslant 1{,}53, \tag{1-19}$$

przy czym wartość średnia M_1/M_2 wynosi 1,28.

Przy założeniu, że jeden rozkaz trójadresowy jest dwukrotnie dłuższy od rozkazu jednoadresowego można przyjąć, że ilość wykorzystanych komórek długich w pamięci maszyny jednoadresowej wynosi około 0,76 ilości komórek pamięci w maszynie trójadresowej zajętej przez program dla rozwiązania danego zagadnienia.

Podobnie dla maszyn jedno- i dwuadresowych można przyjąć, że programy dla maszyny jednoadresowej zajmują około 0,64 ilości komórek potrzebnych dla zapamiętania programów maszyny dwuadresowej.

Oznaczając przez T_1 czas potrzebny dla rozwiązania danego zagadnienia na maszynie jednoadresowej, przez T_2 — czas potrzebny dla rozwiązania danego zagadnienia na maszynie dwuadresowej, przez T_3 — czas potrzebny na rozwiązanie danego zagadnienia na maszynie trójadresowej, Linski pokazał, że zachodzi następująca nierówność:

$$T_1 < T_2 < T_3. \tag{1-20}$$

Trudno jednak powiedzieć, dla jakiej klasy problemów materiał statystyczny, na którym oparł swe rozważania Linski, był reprezentatywny.

2. ZARYS ORGANIZACJI MASZYNY TYPOWEJ

[2.1. Arytmetyka uzupełnieniowa, 2.2. Krótki opis maszyny typowej, 2.3. Kod rozkazowy]

2.1. ARYTMETYKA UZUPEŁNIENIOWA

2.1.0. Zajmiemy się obecnie omówieniem podstawowych zasad binarnej arytmetyki uzupełnieniowej, w której $(n+1)$-bitowe liczby dwójkowowymierne należą do przedziału $<-1, 1-2^{-n}>$ (punkt 1.2). Liczba dodatnia x, jak wiemy, jest przedstawiona w tej arytmetyce następująco:

$$0.\alpha_1\alpha_2\alpha_3 \ldots \alpha_{n-1}\alpha_n,$$

gdzie α_i są to kolejne cyfry dwójkowe (binarne) rozwinięcia liczby x.

Natomiast gdy x jest liczbą ujemną, wówczas w arytmetyce uzupełnieniowej zastępujemy ją liczbą $2+x$, która z kolei ma postać

$$1.\alpha'_1\alpha'_2\alpha'_3 \ldots \alpha'_{n-1}\alpha'_n,$$

gdzie α' $(i = 1, \ldots, n)$ są to cyfry rozwinięcia liczby $1+x$.

Rozpatrzmy kilka przykładów przedstawiania liczb ujemnych:

liczba $\frac{11}{32}$ ma postać

$$0.010110 \ldots 0;$$

natomiast liczba $-\frac{11}{32}$ ma postać

$$2-\frac{11}{32}=\frac{53}{32}, \quad 1.101010 \ldots 0,$$

liczba $\frac{115}{128}$ ma postać:

$$0.11100110 \ldots 0,$$

natomiast liczba $-\frac{115}{128}$ ma postać

$$2-\frac{115}{128}=\frac{141}{128}, \quad 1.00011010 \ldots 0.$$

2.1.1. Algorytm uzupełnienia. Jeżeli mamy liczbę dodatnią x, to dla otrzymania w arytmetyce uzupełnieniowej liczby $-x$ wykonujemy operacje $2-x$. Podobnie jeśli mamy liczbę $-x$, reprezentowaną jako $2-x$, a chcemy otrzymać liczbę x, to musimy wykonać operację $2-(2-x)=x$. Dla praktycznego obliczania uzupełnienia liczb podamy obecnie algorytm.

Niech liczba x będzie postaci

$$\alpha_0 \cdot \alpha_1\alpha_2 \ldots \alpha_{n-1}\alpha_n;$$

wprowadzimy pomocniczą liczbę binarną

$$\omega_0 \cdot \omega_1 \omega_2 \ldots \omega_{n-1} \omega_n,$$

której cyfry są określone przez następujące związki:

$$\omega_i = 0$$

wtedy i tylko wtedy, gdy $\alpha_i = \alpha_{i+1} = \ldots = \alpha_n = 0$ (gdzie $i = n, \ldots 1,0$),

$$\omega_i = 1$$

wtedy i tylko wtedy, gdy istnieje wskaźnik całkowity nieujemny j, spełniający warunek $i \leqslant j \leqslant n$, dla którego $\alpha_j = 1$.

Kolejne cyfry uzupełnienia liczby x wyznaczamy korzystając ze związku

$$\alpha'_i = \omega_i - \alpha_i \omega_{i+1} \text{ (gdzie } i = 0,1, \ldots, n).$$

Dla udowodnienia powyższego algorytmu wystarczy zauważyć, że

$$\alpha_i + \alpha'_i = \begin{cases} 0, & \text{gdy} \quad \omega_i = \omega_{i+1} = 0, \\ 1, & \text{gdy} \quad \omega_i = \omega_{i+1} = 1, \\ 2, & \text{gdy} \quad \omega_i = 1, \omega_{i+1} = 0, \end{cases}$$

oraz skorzystać ze związku

$$2 = \sum_{i=0}^{k} 2^{-i} + 2^{-k}.$$

Przekroczenie zakresu przy uzupełnianiu następuje wtedy i tylko wtedy, gdy $x = -1$.

Przykład 2-1. Obliczamy uzupełnienie liczby

$$\frac{25}{32} = 0.1100100;$$

korzystając z przedstawionego wyżej algorytmu otrzymamy

$$1.0011100;$$

łatwo sprawdzić, że otrzymaliśmy liczbę $\frac{39}{32}$, która jest uzupełnieniem liczby $\frac{25}{32}$.

2.1.2. Algorytm sumowania szeregowego. Niech będą dane dwie dodatnie liczby binarne (dwójkowe), o skończonych rozwinięciach. Załóżmy, że obie te liczby są tej samej długości

$$x_1 = \sum_{i=0}^{n} \alpha_i(x_1) 2^{-i},$$

$$x_2 = \sum_{i=0}^{n} \alpha_i(x_2) 2^{-i}.$$

Dodawanie szeregowe polega na dodawaniu do siebie kolejnych współczynników, począwszy od współczynnika przy najniższych potęgach dwójki.

Jeśli oznaczymy cyfry wyniku przez $\alpha_i(y)$, a przeniesienia z dodawania przez p_i, to algorytm dodawania możemy opisać za pomocą tabl. 2-1, realizującej wzór

$$\alpha_i(x_1) + \alpha_i(x_2) + p_{i+1} = \alpha_i(y) + p_i,$$

gdzie $\alpha_i, p_i \in \{0,1\}$ dla $i = n, n-1, \ldots, 1,0$.

Tablica 2-1

$\alpha_i(x_1)$	$\alpha_i(x_2)$	p_{i+1}	$\alpha_i(y)$	p_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Przy dodaniu do siebie współczynników przy 2^{-n} kładziemy $p_{n+1} = 0$. Niech będą dane dwie liczby x_1, x_2 z przedziału $<-1, 1-2^{-n}>$. Oznaczmy dalej przez y_1 przedstawienie liczby x_1 w arytmetyce uzupełnieniowej, przez y_2 zaś przedstawienie liczby x_2 w arytmetyce uzupełnieniowej. Musimy rozpatrzyć cztery przypadki:

1) $x_1 \geqslant 0$, $x_2 \geqslant 0$, wówczas $y_1 = x_1$, $y_2 = x_2$,
2) $x_1 \geqslant 0$, $x_2 < 0$, wówczas $y_1 = x_1$, $y_2 = 2 + x_2$,
3) $x_1 < 0$, $x_2 \geqslant 0$, wówczas $y_1 = 2 + x_1$, $y_2 = x_2$,
4) $x_1 < 0$, $x_2 < 0$, wówczas $y_1 = 2 + x_1$, $y_2 = 2 + x_2$.

Rozpatrzmy pierwszy przypadek

$$y_1 + y_2 = x_1 + x_2,$$

czyli dodawanie w tym przypadku dwóch liczb w arytmetyce uzupełnieniowej sprowadza się do zwykłego dodawania liczb w rozwinięciu binarnym.

Przypadek drugi i trzeci rozpatrzmy łącznie

$$y_1 + y_2 = 2 + x_1 + x_2 = 2 + (x_1 + x_2).$$

Z kolei musimy rozważyć dwa podprzypadki:

1) gdy $x_1 + x_2 < 0$,
2) gdy $x_1 + x_2 \geqslant 0$.

W pierwszym z wymienionych podprzypadków wynik dodawania daje poprawne

przedstawienie liczby w arytmetyce uzupełnieniowej, w drugim należy zmniejszyć o 2; jest to równoważne nieuwzględnianiu pozycji α_{-1} wyniku.

Rozpatrzmy czwarty przypadek:

$$y_1 + y_2 = 4 + x_1 + x_2 = 2 + 2 + x_1 + x_2,$$

ponieważ $x_1 + x_2 < 0$, aby otrzymać prawidłową postać, wynik należy zmniejszyć o 2, jest to równoważne nieuwzględnianiu pozycji α_{-1} wyniku.

Jak wynika z powyższych rozważań, dodawanie liczb z przedziału $< -1, 1-2^{-n}>$ w arytmetyce uzupełnieniowej sprowadza się do szeregowego dodawania liczb binarnych i zaniedbywania pozycji α_{-1}, odpowiadającej pierwszej potędze dwójki 2^1.

Przekroczenie zakresu. Przy dodawaniu (odejmowaniu) liczb z przedziału $< -1, 1-2^{-n}>$ może nastąpić przekroczenie zakresu. Przy dodawaniu przekroczenie zakresu może nastąpić w dwóch przypadkach:

1) gdy $x_1 > 0$, $x_2 > 0$ oraz $y = x_1 + x_2 \geqslant 1$,
2) gdy $x_1 < 0$, $x_2 < 0$ oraz $y = x_1 + x_2 < -1$.

Oznaczymy zerowy bit liczby x przez $\alpha_0(x)$; bit przekroczenie zakresu oznaczymy przez ϱ (jeśli nastąpiło przekroczenie zakresu, to $\varrho = 1$, jeśli zaś nie nastąpiło, to $\varrho = 0$). Korzystając z powyższych oznaczeń w tabl. 2-2 przedstawiliśmy przekroczenie zakresu w zależności od zerowych bitów argumentów i wyniku dodawania.

Tablica 2-2

$\alpha_0(x_1)$	$\alpha_0(x_2)$	$\alpha_0(y)$	ϱ
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

W przyjętym przez nas systemie przedstawiania liczb ujemnych operację odejmowania sprowadzamy do operacji dodawania odjemnej z uzupełnieniem odjemnika.

2.1.3. Algorytm mnożenia przez $\frac{1}{2}$. Jeśli mamy liczbę dodatnią

$$0.\alpha_1\alpha_2 \ldots \alpha_n,$$

to mnożąc ją przez $\frac{1}{2}$ otrzymujemy w wyniku liczbę

$$0.0\alpha_1\alpha_2 \ldots \alpha_n.$$

Podobnie jeśli mamy liczbę ujemną

$$1.\alpha_1\alpha_2 \ldots \alpha_n,$$

to w wyniku otrzymamy liczbę

$$1.1\alpha_1\alpha_2 \ldots \alpha_n.$$

Ogólnie algorytm mnożenia przez $\frac{1}{2}$ dla liczb binarnych w arytmetyce uzupełnieniowej ma postać:

$$\tfrac{1}{2}(\alpha_0 \cdot \alpha_1\alpha_2 \ldots \alpha_n) = \alpha_0 \cdot \alpha_0\alpha_1\alpha_2 \ldots \alpha_n.$$

2.1.4. Algorytm mnożenia przez 2. Mnożenie dowolnej liczby binarnej w arytmetyce uzupełnieniowej

$$\alpha_0 \cdot \alpha_1\alpha_2 \ldots \alpha_n,$$

przez 2 jest wykonalne (bez przekroczenia zakresu) tylko w przypadku, gdy

$$\alpha_0 = \alpha_1,$$

wówczas

$$2(\alpha_0 \cdot \alpha_1\alpha_2 \ldots \alpha_n) = \alpha_1 \cdot \alpha_2\alpha_3 \ldots \alpha_n,$$

w przypadku zaś, gdy

$$\alpha_0 \neq \alpha_1$$

mnożenie wyprowadza poza przedział $< -1, 1-2^{-n+1} >$, czyli następuje przekroczenie zakresu.

2.1.5. Algorytm mnożenia. Niech będzie dana para liczb zwana dalej mnożną i mnożnikiem. Dalej będziemy oznaczali mnożną literą M, mnożnik zaś literą F

$$M = m_0 \cdot m_1 m_2 \ldots m_{n-1},^{(1)}$$
$$F = f_0 \cdot f_1 f_2 \ldots f_{n-1}.$$

Dopisujemy do F na n-tej pozycji 0, otrzymamy

$$F = f_0 \cdot f_1 f_2 \ldots f_{n-1}\, 0.$$

Rozważmy dalej ciąg par

$$0, f_{n-1}; \quad f_{n-2}, f_{n-1}; \quad \ldots; \quad f_{k-1}, f_k; \quad \ldots; \quad f_0, f_1.$$

Będziemy tworzyli ciąg iloczynów cząstkowych

$$A_0, A_1, \ldots, A_n,$$

gdzie pierwszy wyraz ciągu przyjmujemy równy 0:

$$A_0 = 0;$$

dalsze zaś wyrazy tego ciągu tworzymy korzystając z reguł podanych w tabl. 2-3, roz-

(1) Przy czym zakładamy, że $M \neq -1$, w przypadku gdy $M = -1$, zamiast omawianego dalej algorytmu należy zastosować algorytm uzupełnienia, omawiany w punkcie 2.1.1, do liczby F, aby otrzymać poprawny wynik.

ważając kolejne pary f_k, f_{k-1}. Iloczyn A jest równy podwojonemu n-temu iloczynowi cząstkowemu A_n:

$$A = 2A_n.$$

Dowód powyższego algorytmu, podanego przez Booth'a, nie nastręcza trudności.

Tablica 2-3

f_{k-1}	f_k	A_{n-k+1}	(gdzie $k = n, n-1, \ldots, 1$)
0	0	$\frac{1}{2} A_{n-k}$	
0	1	$\frac{1}{2}(A_{n-k} + M)$	
1	0	$\frac{1}{2}(A_{n-k} - M)$	
1	1	$\frac{1}{2} A_{n-k}$	

Wystarczy zauważyć, że w przypadku, gdy F ma na k-tej pozycji jedynkę, pozostałe zaś bity są zerami, to dwa kroki algorytmu mające istotny wpływ na wynik, odpowiadające dwóm parom f_{k+1}, f_k i f_k, f_{k-1} są równoważne pomnożeniu przez 2^{-k},

$$-M \cdot 2^{-k} + M \cdot 2^{-k+1} = M \cdot (2^{-k+1} - 2^{-k}) = M \cdot 2^{-k}$$

Dalej dowód przebiega indukcyjnie.

Uwaga: W przypadku gdy M i F są liczbami n-bitowymi, wynik mnożenia jest liczbą $2n$-1 bitową.

Przykład 2-2. Niech $M = \frac{3}{4}$, zaś $F = \frac{5}{8}$. W arytmetyce uzupełnieniowej mają postać:

$$M = 0.110, \quad F = 0.101.$$

Rozważmy kolejne pary i skonstruujemy kolejne iloczyny cząstkowe:

$$\begin{aligned} f_3 = 1, \quad f_4 = 0, \quad A_1 &= \tfrac{1}{2}(-M) \quad = 1.1010, \\ f_2 = 0, \quad f_3 = 1, \quad A_2 &= \tfrac{1}{2}(A_1 + M) = 0.00110, \\ f_1 = 1, \quad f_2 = 0, \quad A_3 &= \tfrac{1}{2}(A_2 - M) = 1.101110, \\ f_0 = 0, \quad f_1 = 1, \quad A_4 &= \tfrac{1}{2}(A_3 + M) = 0.0011110, \end{aligned}$$

$$A = 2A_4 = 0.011110 = \frac{15}{32}.$$

Przykład 2-3. Niech $M = -\frac{11}{32}$, zaś $F = \frac{11}{32}$. W arytmetyce uzupełnieniowej mają one postać:

$$M = 1.10101, \quad F = 0.01011.$$

Rozważmy kolejne pary i skonstruujemy kolejne iloczyny cząstkowe:

$$\begin{array}{llll} f_5 = 1, & f_6 = 0, & A_1 = \tfrac{1}{2}(-M) & = 0.001011, \\ f_4 = 1, & f_5 = 1, & A_2 = \tfrac{1}{2}A_1 & = 0.0001011, \\ f_3 = 0, & f_4 = 1, & A_3 = \tfrac{1}{2}(A_2 + M) & = 0.1011111, \\ f_2 = 1, & f_3 = 0, & A_4 = \tfrac{1}{2}(A_3 - M) & = 0.000110111, \\ f_1 = 0, & f_2 = 1, & A_5 = \tfrac{1}{2}(A_4 + M) & = 1.1110000111, \\ f_0 = 0, & f_1 = 0, & A_6 = \tfrac{1}{2}A_5 & = 1.11110000111, \end{array}$$

$$A = 2A_6 = 1.1110000111 = -\frac{121}{1024}.$$

2.1.6. Algorytm zaokrąglenia. Niech będzie dana liczba binarna w arytmetyce uzupełnieniowej o $n + s$ bitach. Przypuśćmy, że potrzebne nam jest zaokrąglenie tej liczby do n bitów, w tym celu musimy sprawdzić znak liczby; jeśli liczba jest dodatnia, to musimy dodać do niej jedynkę na $n + 1$ pozycji, czyli dodać 2^{-n+1};

$$0.\alpha_1\alpha_2 \ldots \alpha_{n-1}\alpha_n \ldots \alpha_{n+s} + \underbrace{0.00 \ldots 010}_{n+1 \text{ bitów}} \ldots 0.$$

W przypadku zaś, gdy liczba jest ujemna, musimy dodać do niej ciąg jedynek począwszy od pierwszej pozycji, a skończywszy na $n + 1$ pozycji, czyli odjąć 2^{-n+1}; uprzednio kładąc $\alpha_{n+1} = 1$;

$$1.\alpha_1\alpha_2 \ldots \alpha_{n-1}\alpha_n \ldots \alpha_{n+s} + \underbrace{1.11 \ldots 110}_{n+1 \text{ bitów}} \ldots 0$$

wzór algebraiczny opisujący algorytm zaokrąglenia liczby x ma postać:

$$z(x) = x + 2^{-n+1}[2 - 2\alpha_0 + \alpha_{n+1}(\alpha_0 - 1)].$$

Przekroczenie zakresu może nastąpić tylko w dwu przypadkach: w pierwszym, jeśli liczba jest dodatnia (czyli $\alpha_0 = 0$) i bity $\alpha_1, \alpha_2, \ldots, \alpha_n$ są jedynkami, w drugim przypadku, jeśli liczba jest ujemna (czyli $\alpha_0 = 1$) i bity $\alpha_1, \alpha_2, \ldots, \alpha_n$ są zerami.

2.1.7. Algorytm dzielenia. Niech będzie dana para liczb: dzielna i dzielnik. Oznaczmy dzielnik literą M, dzielną zaś literą A,

$$A = a_0 \cdot a_1 a_2 \ldots a_{n-1},$$

$$M = m_0 \cdot m_1 m_2 \ldots m_{n-1}.$$

Nasz algorytm musi zapewniać wykonanie dzielenia, tylko w przypadku gdy

$$|A| < |M|, \tag{2-1}$$

ponieważ w przeciwnym razie nastąpiłoby przekroczenie zakresu. Algorytm nasz będzie konstruował nam dwa ciągi: ciąg pomocniczy i ciąg ilorazów częściowych.

Sprawdzenie warunku (2-1). Będziemy rozróżniali dwa przypadki:

$$\begin{array}{ll} \text{jeśli } a_0 = m_0, & \text{to } A_1 = A - M, \\ \text{jeśli } a_0 \neq m_0, & \text{to } A_1 = A + M. \end{array}$$

Tablica 2-4

a_0	m_0	$\|A\| < \|M\|$ wtedy i tylko wtedy, gdy
0	0	$a_0^1 = 1$
0	1	$a_0^1 = 1$
1	0	$a_0^1 = 0$
1	1	$a_0^1 = 0$

Oznaczając bit uzupełnienia A_1 (czyli bit o wskaźniku zero) przez a_0^1 możemy scharakteryzować za pomocą tabl. 2-4 wszystkie przypadki spełniania nierówności (2-1).

W przypadku gdy nie jest spełniona nierówność (2-1); należy:

$$\text{jeśli } a_0 = m_0, \quad \text{to } A_1 + M = A - M + M = A,$$
$$\text{jeśli } a_0 \neq m_0, \quad \text{to } A_1 - M = A + M - M = A,$$

po czym musi być generowany sygnał nadmiaru.

W przypadku spełnienia nierówności (2-1) wykonujemy kolejne kroki algorytmu.

Pierwszy krok algorytmu:

$$\text{jeśli } a_0^1 = m_0, \quad \text{to } A_2 = 2A_1 - M \quad \text{oraz } I_1 = 1,$$
$$\text{jeśli } a_0^1 \neq m_0, \quad \text{to } A_2 = 2A_1 + M \quad \text{oraz } I_1 = 0.$$

k-ty krok algorytmu (gdzie $k = 2,3,\ldots,n$):

$$\text{jeśli } a_0^k = m_0, \quad \text{to } A_{k+1} = 2A_k - M \quad \text{oraz } I_k = I_{k-1} + 2^{-k+1},$$
$$\text{jeśli } a_0^k \neq m_0, \quad \text{to } A_{k+1} = 2A_k + M \quad \text{oraz } I_k = I_{k-1}.$$

Przez a_0^k oznaczyliśmy bit uzupełnienia (czyli bit o wskaźniku zero) liczby A_k. Iloraz $I = I_n$, zaś reszta z dzielenia R wyraża się wzorem:

$$\text{jeśli } a_0^{n+1} = m_0, \quad \text{to } R = 2^{-n}\,(A_{n+1} - M),$$
$$\text{jeśli } a_0^{n+1} \neq m_0, \quad \text{to } R = 2^{-n}\,(A_{n+1} + M).$$

Przez a_0^{n+1} oznaczyliśmy bit uzupełnienia (czyli bit o wskaźniku zero) liczby A_{n+1}.

Przykład 2-4.

$$A = -\frac{3}{8} = 1.101, \quad M = \frac{1}{2} = 0.100$$

Sprawdzenie nierówności (2-1):

$$1 \neq 0, \quad A_1 = 0.001,$$

jak widać z tabl. 2-4, nierówność jest spełniona, wobec czego możemy przystąpić do dzielenia.

pierwszy krok: $0 = 0$, $A_2 = 1.110$, $I_1 = 1.000$,
drugi krok: $1 \neq 0$, $A_3 = 0.000$, $I_2 = 1.000$,
trzeci krok: $0 = 0$, $A_4 = 1.100$, $I_3 = 1.000$,
czwarty krok: $1 \neq 0$, $A_5 = 1.100$, $I_4 = 1.010$,

$$I = I_4 = 1.010 = -\frac{3}{4},$$

$$R = 2^{-4}\ (1.100 + 0.100) = 0.0000.$$

Przykład 2-5.

$$A = -\frac{9}{64} = 1.110111, \quad M = \frac{3}{8} = 0.011.$$

Sprawdzenie nierówności (2-1):

$$1 \neq 0, \quad A_1 = 0{,}001111,$$

jak widać z tabl. 2-4, nierówność jest spełniona, wobec czego możemy przystąpić do dzielenia.

pierwszy krok:	$0 = 0$,	$A_2 = 0.00011$,	$I_1 = 1.00000$,
drugi krok:	$0 = 0$,	$A_3 = 1.1101$,	$I_2 = 1.10000$,
trzeci krok:	$1 \neq 0$,	$A_4 = 0{,}000$,	$I_3 = 1.10000$,
czwarty krok:	$0 = 0$,	$A_5 = 1.101$,	$I_4 = 1.10100$,
piąty krok:	$1 \neq 0$,	$A_6 = 1.101$,	$I_5 = 1.10100$,
szósty krok:	$1 \neq 0$,	$A_7 = 1.101$,	$I_6 = 1.10100$,

$$I = I_6 = 1.10100 = -\frac{3}{8},$$

$$R = 2^{-6}\ (1.101 + 0.011) = 0.00000.$$

2.2. KRÓTKI OPIS MASZYNY TYPOWEJ

2.2.0. Obecnie omówimy zasady organizacji małej UPMC. Maszyna taka mimo niskich parametrów i prostoty organizacji, wystarczy do wyłożenia zasadniczych elementów programowania w całej rozciągłości. Przedstawiona niżej typowa mała UPMC będzie maszyną szeregową stałoprzecinkową, pracującą w binarnej arytmetyce uzupełnieniowej. Maszyna wykonuje operacje na liczbach dwójkowo wymiernych z przedziału $<-1,\ 1-2^{-33}>$.

W maszynie wyróżniamy dwa rodzaje słów:

1) słowa krótkie o długości 17 pozycji binarnych — w skrócie 17 B[1],

2) słowa długie o długości 34 pozycji binarnych — w skrócie 34 B.

Rozkazy (instrukcje) dla maszyny są kodowane binarnie i przedstawione za pomocą słów krótkich.

Maszyna może wykonywać wszystkie operacje arytmetyczne, logiczne zarówno na słowach krótkich, jak i na słowach długich. Zasadniczo słowa 17 bitowe służą do przedstawiania rozkazów, a słowa 34 bitowe — liczb. Gdy używamy słowa 17 bitowe jako

[1] Literę B będziemy czytali bit, 17B czytamy 17 bitów.

$$I = I_4 = 1.010 = -\frac{3}{4},$$

$$R = 2^{-4}\ (1.100 + 0.100) = 0.0000.$$

Przykład 2-5.

$$A = -\frac{9}{64} = 1.110111, \quad M = \frac{3}{8} = 0.011.$$

Sprawdzenie nierówności (2-1):

$$1 \neq 0, \quad A_1 = 0{,}001111,$$

jak widać z tabl. 2-4, nierówność jest spełniona, wobec czego możemy przystąpić do dzielenia.

pierwszy krok:	$0 = 0$,	$A_2 = 0.00011$,	$I_1 = 1.00000$,
drugi krok:	$0 = 0$,	$A_3 = 1.1101$,	$I_2 = 1.10000$,
trzeci krok:	$1 \neq 0$,	$A_4 = 0{,}000$,	$I_3 = 1.10000$,
czwarty krok:	$0 = 0$,	$A_5 = 1.101$,	$I_4 = 1.10100$,
piąty krok:	$1 \neq 0$,	$A_6 = 1.101$,	$I_5 = 1.10100$,
szósty krok:	$1 \neq 0$,	$A_7 = 1.101$,	$I_6 = 1.10100$,

$$I = I_6 = 1.10100 = -\frac{3}{8},$$

$$R = 2^{-6}\ (1.101 + 0.011) = 0.00000.$$

2.2. KRÓTKI OPIS MASZYNY TYPOWEJ

2.2.0. Obecnie omówimy zasady organizacji małej UPMC. Maszyna taka mimo niskich parametrów i prostoty organizacji, wystarczy do wyłożenia zasadniczych elementów programowania w całej rozciągłości. Przedstawiona niżej typowa mała UPMC będzie maszyną szeregową stałoprzecinkową, pracującą w binarnej arytmetyce uzupełnieniowej. Maszyna wykonuje operacje na liczbach dwójkowo wymiernych z przedziału $<-1,\ 1-2^{-33}>$.

W maszynie wyróżniamy dwa rodzaje słów:

1) słowa krótkie o długości 17 pozycji binarnych — w skrócie 17 B[1],

2) słowa długie o długości 34 pozycji binarnych — w skrócie 34 B.

Rozkazy (instrukcje) dla maszyny są kodowane binarnie i przedstawione za pomocą słów krótkich.

Maszyna może wykonywać wszystkie operacje arytmetyczne, logiczne zarówno na słowach krótkich, jak i na słowach długich. Zasadniczo słowa 17 bitowe służą do przedstawiania rozkazów, a słowa 34 bitowe — liczb. Gdy używamy słowa 17 bitowe jako

[1] Literę B będziemy czytali bit, 17B czytamy 17 bitów.

liczby, wtedy maszyna automatycznie dopełnia tę liczbę w rejestrach arytmometru do 34 bitów zerami (od 17 do 34 bitu), tzn. na siedemnastu miejscach najmniej znaczących liczby

$$\alpha_0 \cdot \alpha_1 \ldots \alpha_7 \ldots \alpha_{16}\, \alpha_{17}\, \alpha_{18} \ldots \alpha_{33}$$

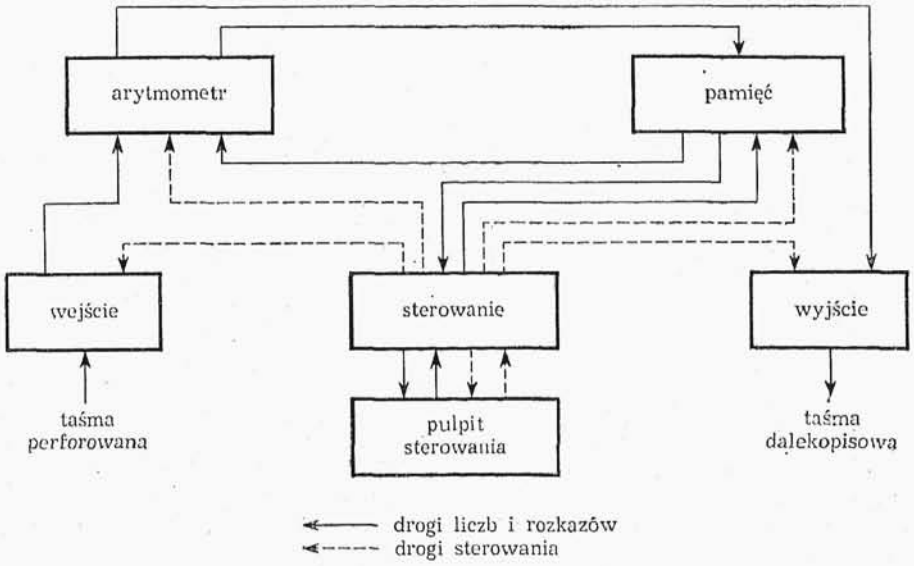
cyfry od α_{17} do α_{33} są zerami.

Prędkość maszyny wynosi około 100 operacji/s.

Maszyna typowa składa się z siedmiu zasadniczych części:

1) wejścia,
2) wyjścia,
3) pamięci,
4) sterowania,
5) arytmometru,
6) pulpitu sterowania,
7) zasilacza.

Schemat ideowy powiązania między sobą pierwszych sześciu podzespołów pokazano na rys. 2-1.



Rys. 2-1. Schemat przykładowej UPMC

W dalszym ciągu zajmiemy się krótkim omówieniem pierwszych sześciu części maszyny. Zasilaczem nie będziemy się zajmowali ze względu na charakter naszych rozważań.

2.2.1. Wejście. Wejście do maszyny stanowi fotoczytnik standartowej taśmy dalekopisowej czytający skokowo, o prędkości czytania ~50 rządków na sekundę. Czytnik jest uruchomiony specjalnym rozkazem maszyny, a odczytany wynik zostaje umieszczony w tzw. miejscu charakterystycznym akumulatora (punkt 2.2.5); poza tym czytnik

samoczynnie przesuwa taśmę, o ile na niej wydziurkowany jest symbol odstępu wejścia i zatrzymuje się na pierwszym rządku, na którym wydziurkowany jest symbol różny od odstępu wejścia. Czytnik czytający taśmę perforowaną neguje pierwszą pozycję rządka taśmy, ponadto każda z pozycji odczytanych przez wejście może być blokowana

Tablica 2-5

Kod dziurkarki i wejścia

Lp.	Symbole na klawiaturze dziurkarki	Kod na taśmie perforowanej	Kod odczytywany przez fotoczytnik
0	„odstęp"	0 0 0 0 0	czytnik nie czyta
1	X	0 0 0 1 1	1 0 0 1 1
2	N	0 0 1 0 1	1 0 1 0 1
3	H	0 0 1 1 0	1 0 1 1 0
4	L	0 0 1 1 1	1 0 1 1 1
5	M	0 1 0 0 0	1 1 0 0 0
6	S	0 1 0 0 1	1 1 0 0 1
7	G	0 1 0 1 0	1 1 0 1 0
8	B	0 1 0 1 1	1 1 0 1 1
9	D	0 1 1 0 0	1 1 1 0 0
10	F	0 1 1 0 1	1 1 1 0 1
11	J	0 1 1 1 0	1 1 1 1 0
12	K	0 1 1 1 1	1 1 1 1 1
13	0 (zero)	1 0 0 0 0	0 0 0 0 0
14	1	1 0 0 0 1	0 0 0 0 1
15	2	1 0 0 1 0	0 0 0 1 0
16	3	1 0 0 1 1	0 0 0 1 1
17	4	1 0 1 0 0	0 0 1 0 0
18	5	1 0 1 0 1	0 0 1 0 1
19	6	1 0 1 1 0	0 0 1 1 0
20	7	1 0 1 1 1	0 0 1 1 1
21	8	1 1 0 0 0	0 1 0 0 0
22	9	1 1 0 0 1	0 1 0 0 1
23	+	1 1 0 1 0	0 1 0 1 0
24	—	1 1 0 1 1	0 1 0 1 1
25	:	1 1 1 0 0	0 1 1 0 0
26	=	1 1 1 0 1	0 1 1 0 1

(tzn. niezależnie od wyniku czytania wejście stawia na zablokowaną pozycję zero) za pomocą specjalnego układu przełączników (punkt 2.2.6). Taśmę dziurkujemy na dziurkarce dalekopisowej o odpowiednio dobranym kodzie. Kod dziurkarki jest przedstawiony w tabl. 2-5.

2.2.2. Wyjście. Wyjściem z maszyny jest odbiornik dalekopisowy (arkuszowy) odpowiednio przystosowany, prędkość drukowania do 10 znaków na sekundę. Wyjście jest uruchomione za pomocą specjalnego rozkazu maszyny, na który zostaje wydruko-

wana zawartość tzw. miejsca charakterystycznego akumulatora (punkt 2.2.5), po zanegowaniu pierwszej pozycji. Kod wyjścia jest podany w tabl. 2-6.

Tablica 2-6

Kod wyjścia

Lp.	Kod wewnętrzny maszyny	Kod zewnętrzny dalekopisu	Symbole drukowane
0	1 0 0 0 0	0 0 0 0 0	niewykorzystane
1	1 0 0 0 1	0 0 0 0 1	powrót karetki
2	1 0 0 1 0	0 0 0 1 0	nowy wiersz
3	1 0 0 1 1	0 0 0 1 1	X /
4	1 0 1 0 0	0 0 1 0 0	odstęp
5	1 0 1 0 1	0 0 1 0 1	N ,
6	1 0 1 1 0	0 0 1 1 0	H niewykorzystane
7	1 0 1 1 1	0 0 1 1 1	L)
8	1 1 0 0 0	0 1 0 0 0	M .
9	1 1 0 0 1	0 1 0 0 1	S ' (apostrof)
10	1 1 0 1 0	0 1 0 1 0	G niewykorzystane
11	1 1 0 1 1	0 1 0 1 1	B ?
12	1 1 1 0 0	0 1 1 0 0	D niewykorzystane
13	1 1 1 0 1	0 1 1 0 1	F niewykorzystane
14	1 1 1 1 0	0 1 1 1 0	J dzwonek
15	1 1 1 1 1	0 1 1 1 1	K (
16	0 0 0 0 0	1 0 0 0 0	P 0
17	0 0 0 0 1	1 0 0 0 1	Q 1
18	0 0 0 1 0	1 0 0 1 0	W 2
19	0 0 0 1 1	1 0 0 1 1	E 3
20	0 0 1 0 0	1 0 1 0 0	R 4
21	0 0 1 0 1	1 0 1 0 1	T 5
22	0 0 1 1 0	1 0 1 1 0	Y 6
23	0 0 1 1 1	1 0 1 1 1	U 7
24	0 1 0 0 0	1 1 0 0 0	I 8
25	0 1 0 0 1	1 1 0 0 1	O 9
26	0 1 0 1 0	1 1 0 1 0	Z +
27	0 1 0 1 1	1 1 0 1 1	A —
28	0 1 1 0 0	1 1 1 0 0	C :
29	0 1 1 0 1	1 1 1 0 1	V =
30	0 1 1 1 0	1 1 1 1 0	przestawia na drukowanie cyfr i znaków
31	0 1 1 1 1	1 1 1 1 1	przestawia na drukowanie liter

2.2.3. Pamięć. Pamięcią wewnętrzną w maszynie typowej jest bęben magnetyczny, wirujący z prędkością 100 obrotów na sekundę. Na bębnie znajdują się 32 ścieżki robocze, z których każda podzielona jest na 32 komórki po 34B (słowa długie), które z kolei dzielą się na dwie komórki po 17B (słowa krótkie); traktując sprawę formalnie, na bębnie mamy nie 32 ścieżki, lecz aż 128 ścieżek; wynika to z faktu, że na każdej

ścieżce słowa długie nagrywamy w czterech fazach(1), wobec czego ze względu na czas oczekiwania (punkt 1.1.6) musimy rozważyć 128 ścieżek zamiast 32. Ponadto na bębnie znajdują się jeszcze trzy ścieżki zegarowe, którymi tu nie będziemy się zajmowali. Jak widać z powyższego, pamięć jest podzielona na 1024 komórki po 34B (lub, jak kto woli, na 2048 komórek po 17B). Każdej komórce przyporządkowany pewien numer (który pozwala ją odnaleźć w pamięci) nazywamy dalej adresem komórki pamięci lub krócej adresem.

Obecnie omówimy system numeracji komórek pamięci. Każdej komórce 17 B przyporządkowano kolejną liczbę naturalną w systemie ósemkowym(2) począwszy od 0000′ do 3777′(3). Komórki 34 B, jak już wspominaliśmy, składają się z dwóch komórek 17B; komórki 17B wchodzące w skład 34B mają adresy: pierwszy parzysty, drugi nieparzysty. Komórce 34 B przyporządkowujemy adres parzystej komórki 17B, wchodzącej w skład komórki 34B powiększony o liczbę ósemkową 4000′. Tak więc adresy komórek 34B przebiegają liczby ósemkowe od 4000′ do 7776′ co 0002′.

Tak na przykład komórka długa o adresie 5760 składa się z dwóch komórek krótkich o adresach 1760 i 1761. Podobnie komórka długa o adresie 7326 składa się z dwóch komórek krótkich o adresach 3326 i 3327.

Należy w tym miejscu wspomnieć o tym, że ze względu na szeregową budowę maszyny bardziej znacząca część słowa długiego znajduje się w komórce 17B o adresie nieparzystym, mniej znacząca zaś część słowa długiego znajduje się w komórce 17B o adresie parzystym.

2.2.4. Sterowanie. Sterowanie składa się z elementów, jak rejestry, pętle operacji itd. Zasadniczymi elementami, które musimy omówić, aby zrozumieć kod rozkazowy maszyny, są: rejester L zwany inaczej *licznikiem rozkazów* (punkt 1.1.2.1), oraz rejestr D zwany inaczej *rejestrem dyspozytora*. Zanim zajmiemy się omówieniem pracy tych rejestrów poświęcimy trochę miejsca na zapoznanie się z budową rozkazów.

Rozkazy w naszej maszynie są kodowane w słowach krótkich (17B), pięć najbardziej znaczących pozycji binarnych przeznaczono na kod operacji (rodzaj operacji), 12 pozostałych pozycji zostało wykorzystane w następujący sposób: najbardziej znacząca pozycja binarna została wykorzystana jako znak adresu długiego (w systemie ósemko-

(1) Inaczej mówiąc, za ostatnim bitem pierwszego słowa długiego nagrywamy kolejno: ostatni bit ósmego słowa długiego, ostatni bit szesnastego słowa długiego, ostatni bit dwudziestego czwartego słowa długiego, następnie nagrywamy przedostatni bit pierwszego słowa długiego itd.

(2) Wprowadzenie liczb ósemkowych ma na celu skrócenie zapisu, zamiast pisać trzy cyfry binarne — piszemy jedną cyfrę ósemkową. Przejście od systemu binarnego do ósemkowego jest natychmiastowe, wartość cyfry ósemkowej e wyznaczamy z wzoru

$$e = 4\alpha_i + 2\alpha_{i-1} + \alpha_{i-2}$$

gdzie α_i, α_{i-1}, α_{i-2} są to kolejne trzy cyfry binarne, które zastępujemy jedną cyfrą ósemkową.

(3) W dalszym ciągu dla rozróżniania liczb binarnych ósemkowych i dziesiętnych przyjmujemy następującą konwencję: część ułamkową od całkowitej oddzielamy w liczbach binarnych kropką, w liczbach ósemkowych — apostrofem, a w liczbach dziesiętnych — przecinkiem. Dla liczb całkowitych w przypadkach wątpliwych na końcu liczby binarnej stawiamy kropkę, na końcu zaś liczby ósemkowej — apostrof.

wym odpowiada mu liczba 4000') pozostałych 11 przeznaczono na zapis adresów słów krótkich (za pomocą 11 pozycji binarnych można zapisać w systemie ósemkowym liczby od 0000' do 3777'). Widzimy, że rozkaz ma następującą budowę:

$$5\,\text{B} + 1\,\text{B} + 11\,\text{B} = 17\,\text{B}.$$

Zajmiemy się obecnie opisem rejestru dyspozytora — jest to rejestr na 17 B, poszczególne pozycje oddziałują na poszczególne zespoły sterowania zgodnie z podziałem funkcyjnym rozkazu, a więc pierwszych pięć pozycji rejestru uruchamia poprzez tzw. matrycę operacji pętlę poszczególnych operacji, szósta pozycja służy do określania, czy wybieramy z pamięci adres słowa krótkiego, czy też długiego, pozostałych jedenaście służy do określania adresu w pamięci albo parametrów operacji (szczegóły w punkcie 2.3).

Omówimy bliżej licznik rozkazów, jest to rejestr na 11 B; licznik rozkazów służy do zapamiętania adresów, pod którymi znajdują się kolejno wykonywane rozkazy. Rejestr L jest powiązany z półsumatorem, który służy do powiększania zawartości rejestru L o jedynkę, ponadto istnieją drogi przesyłania zawartości rejestru L do części adresowej rejestru D i odwrotnie do przesyłania części adresowej rejestru D do rejestru L oraz drogi przesyłania zawartości rejestru L do pamięci i odwrotnie.

Postaramy się obecnie wyjaśnić zasady pracy maszyny. Maszyna pracuje dwutaktowo.

Pierwszy takt pracy. Zawartość licznika rozkazów zostaje przesłana do części adresowej rejestru dyspozytora, a następnie zostaje powiększona o jedynkę. Rozkaz o części operacyjnej równej zeru i o adresie słowa krótkiego (jest to tzw. rozkaz pobrania z pamięci, patrz punkt 2.3.4) powoduje pobranie słowa zapisanego pod wskazanym adresem i umieszczenie go w rejestrze dyspozytora.

Drugi takt pracy. Zostaje wykonany rozkaz znajdujący się w rejestrze dyspozytora.

Po drugim takcie powtarza się pierwszy itd.

2.2.5. Arytmometr. W skład arytmometru wchodzi kilka rejestrów i układów. Ze względu na opis, potrzebny nam dla zrozumienia działania poszczególnych rozkazów, wymienimy następujące elementy arytmometru:

1) rejestr podstawowy zwany akumulatorem (w skrócie oznaczany A) o długości 68 B, pozycje akumulatora będziemy numerowali zgodnie z numeracją pozycji liczb binarnych w arytmetyce uzupełnieniowej (punkt 2.1.1);

2) rejestr mnożnej (w skrócie oznaczany M) o długości 34 B;

3) rejestr sygnału W (w skrócie oznaczany W) o długości 1 B;

4) rejestr sygnału N (w skrócie oznaczany N) o długości 1 B;

5) układ realizujący uzupełnienie liczby (w skrócie zwany uzupełnieniem);

6) układ realizujący sumę dwóch liczb (w skrócie zwany sumatorem).

Obecnie pokrótce omówimy rejestry A i M.

Akumulator. Słowa długie z pamięci możemy przesyłać do 34 bardziej znaczących pozycji akumulatora i, odwrotnie, zawartość bardziej znaczących 34 pozycji możemy przesyłać do pamięci (oczywiście pod adres słowa długiego), podobnie słowa krótkie z pamięci maszyny możemy przesyłać na 17 najbardziej znaczących pozycji akumulatora i, odwrotnie, zawartość 17 najbardziej znaczących pozycji akumulatora możemy prze-

syłać do pamięci (oczywiście pod adres słowa krótkiego). Możemy również przesuwać arytmetycznie zawartość akumulatora w lewo albo w prawo o ilość pozycji mniejszą lub równą 63, podobnie możemy przesuwać zawartość akumulatora cyklicznie[1]) w lewo albo w prawo o ilość pozycji mniejszą lub równą 63. Ponadto możemy przesyłać zawartość akumulatora przez sumator i wynik umieszczać z powrotem w akumulatorze (oczywiście w sumatorze dokonujemy sumowania zawartości akumulatora z zawartością dowolnego adresu pamięci n), możliwe jest również przesyłanie z pamięci przez układ uzupełniania do akumulatora. W przypadku przekroczenia zakresu bit przekroczenia zakresu nie ginie, lecz zostaje umieszczony na specjalnej dodatkowej pozycji akumulatora; po przesunięciu zawartości akumulatora w kierunku pozycji najmniej znaczących, zawartość pozycji przekroczenia zakresu zostaje umieszczona na pozycji zerowej akumulatora, ponadto bit ten bierze udział w operacji dodawania i odejmowania, w pozostałych operacjach bit ten nie bierze udziału. Pięć najbardziej znaczących pozycji akumulatora będziemy nazywali miejscem charakterystycznym akumulatora, na to miejsce wprowadzimy rządek taśmy perforowanej odczytany przez wejście oraz zawartość tych pięciu pozycji przesyłamy na wyjście.

Rejestr mnożnej. Możliwe jest tylko przesyłanie z pamięci do rejestru mnożnej; zawartość komórek krótkich jest przesyłana z pamięci na 17 bardziej znaczących pozycji rejestru M.

2.2.6. Pulpit sterowania. Służy do manipulacji ręcznych oraz obserwacji pracy maszyny, przyczyn zatrzymania itp. Na pulpicie sterowania znajdują się neonówki i przełączniki. W Załączniku 2 zajmiemy się omówieniem roli poszczególnych neonówek i przełączników (Załącznik 2 umieszczony jest na końcu książki).

2.3. KOD ROZKAZOWY

2.3.0. Dla potrzeb dalszych rozważań wprowadzimy kilka definicji:

Definicja 1. Będziemy mówili, że n jest adresem krótkim, zamiast mówić, że n jest adresem słowa krótkiego.

Definicja 2. Będziemy mówili, że n jest adresem długim, zamiast mówić, że n jest adresem słowa długiego.

Definicja 3. Będziemy mówili krótko — zawartość adresu n, zamiast mówić słowo zapisane pod adresem n.

Definicja 4. Będziemy mówili krótko — przesłanie słowa (z pamięci do któregoś z rejestrów albo odwrotnie), zamiast mówić przesłanie słowa (z pamięci do któregoś z rejestrów albo odwrotnie) i zapisanie go (w którymś z rejestrów albo w którejś z komórek pamięci) na miejscu poprzedniej zawartości, przy czym zawartość miejsca, z którego przesyłamy (komórki pamięci albo któregoś z rejestrów), nie ulega zmianie.

Definicja 5. Będziemy mówili krótko — sekwencja rozkazów, zamiast mówić skończony ciąg rozkazów umieszczony pod kolejnymi adresami w pamięci, takich,

---

[1]) Przez przesuwanie cykliczne rozumiemy takie przesuwanie, przy którym zakładamy, że najbardziej znaczący bit α_0 i najmniej znaczący bit α_{67} są sąsiednimi bitami.

że maszyna musi je wykonywać jeden po drugim w kolejności wzrastających adresów (pod którymi zapisane są rozkazy) począwszy od pierwszego wyrazu ciągu, niezależnie od zawartości komórek pamięci i rejestrów wykorzystanych w tym ciągu obliczeń.

Definicja 6. Zawartość adresu n będziemy oznaczali krótko (n), podobnie zawartość akumulatora A będziemy oznaczali (A), zawartość rejestru M będziemy oznaczali (M), zawartość rejestru L będziemy oznaczali (L) i zawartość rejestru D będziemy oznaczali (D).

2.3.1. Operacje przesyłania. Operacje przesyłania służą do przesyłania liczb (rozkazów) z pamięci do rejestrów A i M oraz zawartości rejestru A do pamięci. Obecnie omówimy kolejno te operacje i podamy proste przykłady ich zastosowania.

Plus przesyłanie: 1) gdy n jest adresem długim, prześlij zawartość adresu n do 34 bardziej znaczących pozycji akumulatora, 2) gdy n jest adresem krótkim, prześlij zawartość adresu n do 17 najbardziej znaczących pozycji akumulatora. Jeśli liczba przesłana do akumulatora jest różna od zera, generuj sygnał $W = 1$, jeśli zaś jest zerem, generuj sygnał $W=0$. Generuj sygnał $N=0$.

Kod operacji: 12 n.

Zapis symboliczny: $(n) \rightarrow A$.

Minus przesyłanie: 1) gdy n jest adresem długim, prześlij uzupełnienie zawartości adresu n do 34 bardziej znaczących pozycji akumulatora, 2) gdy n jest adresem krótkim, prześlij uzupełnienie zawartości adresu n do 17 najbardziej znaczących pozycji akumulatora. Jeśli liczba przesyłana do akumulatora jest różna od zera, generuj sygnał $W = 1$, jeśli zaś jest zerem, generuj sygnał $W = 0$. Jeśli nastąpiło przekroczenie zakresu, generuj sygnał $N = 1$.

Kod operacji: 13 n.

Zapis symboliczny: $-(n) \rightarrow A$.

Przesyłanie do pamięci: 1) gdy n jest adresem długim, prześlij zawartość 34 bardziej znaczących pozycji akumulatora do komórki pamięci o adresie n, 2) gdy n jest adresem krótkim, prześlij zawartość 17 najbardziej znaczących pozycji akumulatora do komórki pamięci o adresie n. Jeśli zawartość akumulatora jest ujemna, generuj sygnał $W = 1$, jeśli zawartość akumulatora jest nieujemna, generuj sygnał $W = 0$. Generuj sygnał $N = 0$.

Uwaga: W wyniku operacji przesyłanie do pamięci zawartości akumulatora nie ulega zmianie.

Kod operacji: 14 n.

Zapis symboliczny: $(A) \rightarrow n$.

Przesyłanie do M: 1) gdy n jest adresem długim, prześlij zawartość adresu n do rejestru mnożnej M; 2) gdy n jest adresem krótkim, prześlij zawartość adresu n do 17 bardziej znaczących pozycji rejestru mnożnej M. Jeśli liczba przesłana do rejestru mnożnej M jest ujemna, generuj sygnał $W = 1$, jeśli jest nieujemna, generuj sygnał $W = 0$. Generuj sygnał $N=0$.

Kod operacji: 27 n.

Zapis symboliczny: $(n) \rightarrow M$.

Rozpatrzmy obecnie przykłady zastosowania powyższych operacji.

Przypuśćmy, że chcemy przesłać zawartość komórki pamięci o adresie n do komórki pamięci o adresie m. Wykonujemy to w sposób następujący:

1) $(n) \rightarrow A$,
2) $(A) \rightarrow m$.

Załóżmy, że w komórce pamięci o adresie n mamy jakąś liczbę l (co do modułu mniejszą od 1), przypuśćmy, że dla pewnych celów potrzebna nam jest nie liczba l, ale liczba $-l$, która musi być umieszczona w komórce pamięci o adresie m. Wykonujemy wtedy operacje:

1) $-(n) \rightarrow A$,
2) $(A) \rightarrow m$.

2.3.2. Operacje arytmetyczne (dokończenie). *Dodawanie*: 1) gdy n jest adresem długim, do 34 bardziej znaczących pozycji akumulatora dodaj zawartość adresu n; 2) gdy n jest adresem krótkim, do 17 najbardziej znaczących pozycji akumulatora dodaj zawartość adresu n. Jeśli wynik jest liczbą ujemną, generuj sygnał $W = 1$, jeśli zaś jest liczbą nieujemną, generuj sygnał $W = 0$. Jeśli zaś nastąpiło przekroczenie zakresu, generuj sygnał $N = 1$.

Kod operacji: 10 n.

Zapis symboliczny: $(A) + (n) \rightarrow A$.

Odejmowanie: 1) gdy n jest adresem długim od 34 bardziej znaczących pozycji akumulatora odejmij zawartość adresu n; 2) gdy n jest adresem krótkim, od 17 najbardziej znaczących pozycji akumulatora odejmij zawartość adresu n. Jeśli wynik jest liczbą ujemną, generuj sygnał $W = 1$, jeśli zaś liczbą nieujemną, generuj sygnał $W = 0$. Jeśli nastąpiło przekroczenie zakresu, generuj sygnał $N = 1$.

Kod operacji: 11 n.

Zapis symboliczny: $(A) - (n) \rightarrow A$.

Mnożenie: pomnóż zawartość adresu n przez zawartość rejestru mnożnej M; wynik umieść w akumulatorze. Jeśli wynik jest liczbą ujemną, generuj sygnał $W = 1$, jeśli zaś jest liczbą nieujemną, generuj sygnał $W = 0$. Jeśli nastąpiło przekroczenie zakresu, generuj sygnał $N = 1$[1]).

Uwaga: wykonanie operacji mnożenia nie zmienia zawartości rejestru M.

Kod operacji: 16 n.

Zapis symboliczny: $(M) \cdot (n) \rightarrow A$.

Zaokrąglenie: wykonaj na zawartości akumulatora operacje zaokrąglenia, według algorytmu podanego w punkcie 2.1.6. Jeśli zawartość akumulatora jest ujemna, generuj sygnał $W = 1$, jeśli zawartość akumulatora jest nieujemna, generuj sygnał $W = 0$. Jeśli zaś w wyniku zaokrąglenia nastąpiło przekroczenie zakresu, generuj sygnał $N = 1$.

Kod operacji: 15 n.

Zapis symboliczny: $z(A) \rightarrow A$.

---

[1]) Przy mnożeniu przekroczenie zakresu może nastąpić tylko w przypadku pomnożenia -1 przez -1, wówczas wynik mnożenia równa się $+1$ (punkt 2.1.5).

Dzielenie: Zawartość akumulatora podziel przez zawartość adresu n, umieszczając iloraz na 34 bardziej znaczących pozycjach akumulatora, resztę zaś umieszczając na 34 mniej znaczących pozycjach akumulatora (iloraz i reszta są traktowane jako dwa niezależne słowa długie). Jeśli wynik jest liczbą ujemną, generuj sygnał $W = 1$, jeśli zaś wynik jest liczbą nieujemną, generuj sygnał $W = 0$. Jeśli $|(A)| \geqslant |(n)|$, generuj sygnał $N = 1$, przy czym dzielenie nie zostaje wykonane.

Uwaga: W przypadku gdy maszyna nie wykonuje dzielenia, przechodzi ona do następnego taktu pracy.

Kod operacji: 17 n.

Zapis symboliczny: $(A) : (n) \rightarrow A$.

Tworzenie wartości bezwzględnej: zastąp zawartość akumulatora bezwzględną wartością zawartości akumulatora. Jeśli liczba zawarta w akumulatorze jest różna od zera, generuj sygnał $W = 1$, jeśli zaś jest równa zeru, generuj sygnał $W = 0$. Jeśli nastąpiło przekroczenie zakresu(1) generuj sygnał $N = 1$.

Kod operacji: 26 n.

Zapis symboliczny: $|(A)| \rightarrow A$.

Rozpatrzmy obecnie przykłady zastosowania powyższych operacji. We wszystkich rozpatrywanych przykładach zakładamy, że w wyniku wykonywanej operacji nie nastąpiło przekroczenie zakresu. Przypuśćmy, że chcemy dodać zawartość adresu n_1 do zawartości adresu n_2 i wynik umieścić pod adresem n_3 — wykonujemy to w następujący sposób:

1) $(n_1) \rightarrow A$,
2) $(A) + (n_2) \rightarrow A$,
3) $(A) \rightarrow n_3$.

Przypuśćmy z kolei, że chcemy pomnożyć zawartość adresu n_1 przez zawartość adresu n_2 wynik mnożenia zaokrąglić i przesłać pod adres n_3; wykonujemy to w następujący sposób:

1) $(n_1) \rightarrow M$,
2) $(M) \cdot (n_2) \rightarrow A$,
3) $z(A) \rightarrow A$,
4) $(A) \rightarrow n_3$.

Dzielenie zawartości adresu n_1 przez zawartość adresu n_2 i umieszczenie wyniku pod adresem n_3 wykonujemy w następujący sposób:

1) $(n_1) \rightarrow A$,
2) $(A) : (n_2) \rightarrow A$,
3) $(A) \rightarrow n_3$.

Zastanówmy się jeszcze, jak obliczyć różnice wartości bezwzględnych zawartości adresu n_1 i n_2, a wynik umieścić pod adresem n_3. Dla wykonania powyższego zadania potrzebna nam jest jeszcze jedna pomocnicza komórka pamięci (tzw. komórka robocza) o adresie m. Nasze zadanie rozwiązujemy wówczas następująco:

---

(1) Przekroczenie zakresu przy obliczaniu wartości bezwzględnej może nastąpić tylko w przypadku, gdy $(A) = -1$.

1) $(n_2) \rightarrow A$,
2) $|(A)| \rightarrow A$,
3) $(A) \rightarrow m$,
4) $(n_1) \rightarrow A$,
5) $|(A)| \rightarrow A$,
6) $(A) - (m) \rightarrow A$,
7) $(A) \rightarrow n_3$.

Do grupy operacji arytmetycznych zaliczamy jeszcze dwie operacje.

Mnożenie przez 2^n: zawartość akumulatora pomnóż przez 2^n. Liczba n nie może być większa od 63 (adres Mod 64). Jeśli wynik jest liczbą ujemną, generuj sygnał $W = 1$, jeśli zaś wynik jest liczbą nieujemną, generuj sygnał $W = 0$. Jeśli nastąpiło przekroczenie zakresu, generuj sygnał $N = 1$.

Kod operacji: 20 n.

Zapis symboliczny: $(A) \cdot 2^n \rightarrow A$.

Dzielenie przez 2^n: zawartość akumulatora pomnóż przez 2^{-n}. Liczba n nie może być większa od 63 (adres Mod 64). Jeśli wynik jest liczbą ujemną, generuj sygnał $W = 1$, jeśli zaś wynik jest liczbą nieujemną, generuj sygnał $W = 0$. Generuj sygnał $N = 0$.

Kod operacji: 21 n.

Zapis symboliczny: $(A) : 2^n \rightarrow A$.

Zastanówmy się obecnie nad przykładem zastosowania operacji mnożenia przez 2^{-n}. Przypuśćmy, że chcemy pomnożyć zawartość adresu n_1 przez $\frac{1}{64}$ i wynik przesłać pod adres n_2; możemy zrealizować to dwoma sposobami:

a) korzystając z operacji mnożenia, pomnożyć zawartość adresu n_1 przez $\frac{1}{64}$ i wynik przesłać pod adres n_2,

b) skorzystać z operacji dzielenia przez 2^n, a mianowicie

1) $(n_1) \rightarrow A$,
2) $(A) : 2^6 \rightarrow A$,
3) $(A) \rightarrow n_2$;

przy korzystaniu z operacji mnożenia musimy pamiętać jeszcze liczbę $\frac{1}{64}$, a wykonanie naszego zadania dłużej by trwało, ponieważ we wszystkich trzech operacjach występują czasy oczekiwania. Natomiast korzystając z drugiego sposobu mamy czas oczekiwania tylko w dwóch operacjach, mianowicie w pierwszej i trzeciej.

2.3.3. Operacje logiczne. Przez operacje logiczne będziemy rozumieli operacje wykonywane na zawartościach akumulatora, przy czym każda pozycja akumulatora jest traktowana jako oddzielna zmienna zero-jedynkowa.

Koniunkcja: weź koniunkcję (1) zawartości akumulatora i zawartości adresu n.

(1) Przez koniunkcję dwu liczb binarnych α_i, β_i rozumiemy operację określoną następującymi wzorami:

$$\alpha_i \cap \beta_i = \begin{cases} 0, & \text{jeśli } \alpha_i = 0, \\ \beta_i, & \text{jeśli } \alpha_i = 1. \end{cases}$$

Uwaga: Jeśli adres n jest adresem krótkim, to maszyna traktuje to jakby na 17 mniej znaczących pozycjach zawartości adresu długiego były same zera.

Jeśli wynik jest liczbą różną od zera, generuj sygnał $W = 1$, jeśli zaś wynik jest zerem, generuj sygnał $W = 0$. Generuj sygnał $N = 0$.

Kod operacji: 30 n.

Zapis symboliczny: $(A) \cap (n) \rightarrow A$.

Przesuwanie cykliczne (patrz odnośnik do punktu 2.2.5) *w prawo*: zawartość akumulatora przesuń o n miejsc binarnych cyklicznie w prawo. Liczba n nie może być większa od 63 (adres Mod 64). Jeśli wynik jest liczbą ujemną, generuj sygnał $W = 1$, jeśli zaś wynik jest liczbą nieujemną, generuj sygnał $W = 0$. Generuj sygnał $N = 0$.

Kod operacji: 22 n.

Zapis symboliczny: $(A) : 2_n \rightarrow A$.

Przesuwanie cykliczne w lewo: zawartość akumulatora przesuń o n miejsc binarnych cyklicznie w lewo. Liczba n nie może być większa od 63 (adres Mod 64). Jeśli wynik jest liczbą ujemną, generuj sygnał $W = 1$, jeśli zaś wynik jest liczbą nieujemną, generuj sygnał $W = 0$. Generuj sygnał $N = 0$.

Kod operacji: 23 n

Zapis symboliczny: $(A) \cdot 2_n \rightarrow A$.

Rozpatrzmy przykład zastosowania powyższych operacji: przypuśćmy, że dla pewnego celu musimy w liczbie 17B, znajdującej się pod adresem krótkim n_1, położyć na pozycje binarne 7, 8, 9, 10 zera i wynik umieścić pod adresem krótkim n_2. Jak to wykonać? Zadanie to rozwiązujemy stosując operację koniunkcji; w tym celu załóżmy, że pod adresem krótkim m znajduje się liczba binarna, która ma na pozycjach 0, 1, 2, 3, 4, 5, 6 jedynki, na pozycjach 7, 8, 9, 10 zera i na pozycjach 12, 13, 14, 15, 16 jedynki.

1) $(n_1) \rightarrow A$,
2) $(A) \cap (m) \rightarrow A$,
3) $(A) \rightarrow n_2$.

Jak wiemy, w wyniku podzielenia dwóch liczb (oczywiście, o ile dzielenie to jest wykonalne) otrzymujemy wynik na 34 bardziej znaczących pozycjach akumulatora, reszta z dzielenia zaś zostaje umieszczona na 34 mniej znaczących pozycjach akumulatora. Przypuśćmy, że wynik dzielenia chcemy przesłać pod adres n_1, resztę zaś pod adres n_2 (oba adresy są oczywiście adresami długimi), możemy to wykonać na dwóch drogach:

a. Korzystając z operacji przesuwania cyklicznego w prawo

1) $(A) \rightarrow n_1$,
2) $(A) : 2_{34} \rightarrow A$,
3) $(A) \rightarrow n_2$.

Operację koniunkcji maszyna wykonuje na zawartości akumulatora:

$$\alpha_0 \cdot \alpha_1 \alpha_2 \ldots \alpha_{33}$$

i zawartości adresu n

$$\beta_0 , \beta_1 \beta_2 \ldots \beta_{33}$$

realizując koniunkcję kolejnych par α_i, β_i dla $i = 0, 1, 2, \ldots, 33$.

b. Korzystając z operacji przesuwania cyklicznego w lewo

1) $(A) \rightarrow n_1$,

2) $(A) \cdot 2_{34} \rightarrow A$,

3) $(A) \rightarrow n_2$.

2.3.4. Operacje organizacyjne. *Czytanie:* Odczytaj kolejny rządek taśmy perforowanej z uwzględnieniem położenia przełączników „blokada wejścia" (patrz Załącznik 2, pulpit sterowania) i wynik zsumuj logicznie z zawartości miejsca charakterystycznego akumulatora, przesuwając jednocześnie cyklicznie poprzednią zawartość akumulatora o pięć pozycji binarnych w prawo. Jeśli na którymś z uwarunkowanych miejsc (za pomocą przełączników „warunków wejścia Y", patrz Załącznik 2) pojawi się jedynka, to generuj sygnał $W = 1$, jeśli zaś na żadnym z uwarunkowanych miejsc nie pojawi się „jedynka" albo jeśli żadna pozycja nie jest uwarunkowana, generuj sygnał $W = 0$. Generuj sygnał $N = 0$.

Kod operacji: 24 n.

Zapis symboliczny: czytaj.

Drukowanie: wydrukuj znak odpowiadający zawartości miejsca charakterystycznego akumulatora, z uwzględnieniem położenia przełączników „blokada wyjścia" (patrz Załącznik 2, pulpit sterowania). Jeśli na którymś z uwarunkowanych miejsc za pomocą przełączników „warunek wyjścia Z" (patrz Załącznik 2) pojawi się jedynka, to generuj sygnał $W = 1$, jeśli zaś na żadnym z uwarunkowanych miejsc nie pojawi się jedynka albo jeśli żadna pozycja nie jest uwarunkowana, to generuj sygnał $W = 0$. Generuj sygnał $N = 0$.

Kod operacji: 25 n.

Zapis symboliczny: drukuj.

Przykładów na zastosowanie tych dwóch rozkazów nie będziemy omawiali, omówimy je dopiero przy programach wprowadzających i wyprowadzających w rozdz. 5.

Skok: do licznika rozkazów L prześlij część adresową wykonywanego rozkazu (tzn. następny rozkaz pobierz z komórki o adresie n). Nie zmieniaj zawartości rejestrów N i W.

Kod operacji: 02 n.

Zapis symboliczny: $n \rightarrow L$.

Powyższy rozkaz ma następujące zastosowanie: przypuśćmy, że po wykonaniu pewnej sekwencji rozkazów znajdujących się pod adresami $k, \ldots, k + p$, chcemy przejść do sekwencji, której pierwszy rozkaz znajduje się pod adresami n, wówczas pod adresem $k + p + 1$ umieszczamy rozkaz skokowy,

$$n \rightarrow L,$$

po którego wykonaniu maszyna zacznie wykonywać kolejne rozkazy sekwencji zaczynającej się pod adresem n.

Skok z podstawieniem: do licznika rozkazów L prześlij część adresową słowa odczytanego pod adresem n. Nie zmieniaj zawartości rejestrów N i W.

Kod operacji: 01 n.

Zapis symboliczny: $(n) \rightarrow L$.

Rozkaz skoku z podstawieniem ma podobne zastosowanie jak rozkaz skoku, z tą różnicą, że w powiązaniu z dalej omówionym rozkazem skoku ze śladem daje bardzo wygodną organizację wywoływania podprogramów.

Skok ze śladem: zawartość licznika rozkazów L prześlij pod adres n, do licznika rozkazów L prześlij zaś $n + 1$. Nie zmieniaj zawartości rejestrów N i W.

Kod operacji: 04 n.

Zapis symboliczny: $(L) \rightarrow n$,
$n + 1 \rightarrow L$.

Rozkaz ten jest ciekawym rozwiązaniem organizacyjnym, daje on duże ułatwienie przy organizacji programu głównego (rozdz. 4).

Rozpatrzmy przykład zastosowania rozkazów skok ze śladem i skok z podstawieniem. Przypuśćmy, że mamy sekwencję rozkazów umieszczonych pod adresami od $n + 1$ do $n + p$, którą w trakcie wykonywania programu wykonujemy wielokrotnie. W celu wywołania w odpowiednim miejscu programu głównego tej sekwencji, umieszczamy w programie głównym rozkaz skok ze śladem (o części adresowej równej n). Wykonanie tego rozkazu powoduje zapisanie pod adresem n zawartości licznika rozkazów, po czym maszyna zaczyna wykonywać kolejne rozkazy sekwencji począwszy od rozkazu zapisanego pod adresem $n + 1$. Jeżeli pod adresem $n + p + 1$ umieścimy rozkaz skok z podstawieniem (o części adresowej równej n), to po wykonaniu ostatniego rozkazu sekwencji (o adresie $n + p + 1$) maszyna pobierze rozkaz zapisany pod adresem $n + p + 1$, który spowoduje powrót do programu głównego, według stanu licznika rozkazów zapisanego pod adresem n.

Pierwszy skok warunkowy: jeżeli zawartość rejestru $W = 1$, to część adresową wykonywanego rozkazu prześlij do licznika rozkazów L; jeżeli zaś zawartość rejestru $W = 0$, nie zmieniaj zawartości licznika rozkazów L.

Uwaga: Rozkaz pierwszego skoku warunkowego zeruje rejestr W i nie zmienia zawartości rejestru N.

Kod operacji: 03 n.

Zapis symboliczny: $n \rightarrow L$,
$W = 1$.

Zajmiemy się omówieniem przykładów zastosowania pierwszego rozkazu skoku warunkowego. Przypuśćmy, że mamy odjąć od zawartości adresu n_1 zawartość adresu n_2 i jeśli wynik odejmowania jest ujemny (czyli $(n_1) < (n_2)$), przejść do sekwencji rozpoczynającej się od rozkazu zapisanego pod adresem n. W tym zaś przypadku, gdy wynik odejmowania jest nieujemny (czyli $(n_1) \geqslant (n_2)$). Mamy wykonać kolejny rozkaz:

1) $(n_1) \rightarrow A$,
2) $(A) - (n_2) \rightarrow A$,
3) $n \rightarrow L$,
$W = 1$.

Zupełnie analogicznie działa rozkaz drugiego skoku warunkowego.

Drugi skok warunkowy: jeśli zawartość rejestru $W = 0$, to część adresową wykonywanego rozkazu prześlij do licznika rozkazów L; jeśli zaś zawartość rejestru $W = 1$, nie zmieniaj zawartości licznika rozkażów L.

Uwaga: Rozkaz drugiego skoku warunkowego zeruje rejestr W, nie zmienia zawartości rejestru N.

Kod operacji: 06 n.

Zapis symboliczny: $n \rightarrow L$,
$W = 0$.

Skok ze śladem przy nadmiarze: jeśli zawartość rejestru $N = 1$, to zawartość licznika rozkazów (L) prześlij pod adres n, zaś $n + 1$ prześlij do licznika rozkazów L; jeśli natomiast zawartość rejestru $N = 0$, nie przesyłaj zawartości licznika rozkazów L pod adresem n oraz nie zmieniaj zawartości licznika rozkazów.

Uwaga: Rozkaz skoku ze śladem przy nadmiarze zeruje rejestr N i nie zmienia zawartości rejestru W.

Kod operacji: 37 n.

Zapis symboliczny: $(L) \rightarrow n$,
$n + 1 \rightarrow L$,
$N = 1$.

W przypadku gdy przełącznik W10 (patrz pulpit sterowania, Załącznik 2) znajduje się w dolnym położeniu, dla uniknięcia przekroczenia zakresu korzystamy z rozkazu skoku ze śladem przy nadmiarze; rozkaz ten działa podobnie do rozkazu pierwszego skoku warunkowego z tym, że w przypadku $N = 1$ nie pobiera rozkazu stojącego pod adresem n, lecz przesyła — podobnie jak rozkaz skoku ze śladem — zawartość licznika rozkazów pod adres n, po czym pobiera rozkaz stojący pod adresem $n + 1$.

Pobranie rozkazu: pobierz rozkaz zawarty w komórce o adresie krótkim n, wykonaj go i powróć do poprzedniej sekwencji rozkazów, o ile rozkaz wykonywany nie był rozkazem skokowym. Nie zmieniaj zawartości rejestrów N i W.

Kod operacji: 00 n.

Zapis symboliczny: $(n) \rightarrow D$.

Rozkaz ten jest opisem operacji wykonywanym przez maszynę w pierwszym takcie (punkt 2.2.4), jest to pobranie rozkazu do wykonania. W pewnych przypadkach wygodnie jest go stosować w programach.

Stop z pobraniem: prześlij rozkaz według wskazań części adresowej do rejestru dyspozytora D, po czym zatrzymaj maszynę. Po ponownym uruchomieniu w zależności od manipulacji ręcznej albo zostaje wykonany rozkaz znajdujący się w rejestrze D, po którym maszyna pobierze rozkaz według wskazań licznika rozkazów L, albo maszyna pobierze kolejny rozkaz według wskazań licznika L (patrz Załącznik 2). Nie zmieniaj zawartości rejestrów N i W.

Kod operacji: 05 n.

Zapis symboliczny: stop n.

Część druga

PROGRAMOWANIE

3. ZASADY PROGRAMOWANIA I KODOWANIA

[3.0. Uwagi wstępne, 3.1. Metoda adresów względnych, 3.2. Działania arytmetyczne na rozkazach, 3.3. Schematy blokowe, 3.4. Programy liniowe, 3.5. Programy z rozwidleniami, 3.6. Programy cykliczne i iteracyjne, 3.7. Sterowanie cyklami]

3.0. UWAGI WSTĘPNE

Zanim przejdziemy do omawiania zasad programowania i kodowania, zastanówmy się nad definicją programowania i nad definicją kodowania. Przez programowanie będziemy rozumieli budowanie algorytmów postępowania maszyny dla rozwiązania określonego zadania. Warunkami, jakie muszą spełniać te algorytmy, zajmiemy się w rozdz. 4, obecnie ograniczymy się do omówienia zasadniczych elementów tych algorytmów oraz ich kodowania. Ostatnie pojęcie musimy wyjaśnić bliżej. Przez kodowanie będziemy rozumieli zapisywanie algorytmów w języku zrozumiałym dla maszyny — w tak zwanym kodzie zewnętrznym maszyny. Program wprowadzający, który omówimy w punkcie 5.1.1, tłumaczy kod zewnętrzny na kod wewnętrzny maszyny. W UPMC omówionej przykładowo kod wewnętrzny maszyny jest kodem binarnym (dwójkowym), kod zewnętrzny zaś jest kodem ósemkowym, wobec czego przejście od kodu zewnętrznego do wewnętrznego jest bardzo proste. Przyjęte przez nas rozwiązanie stwarza jedną zasadniczą trudność, mianowicie wymaga od programującego dużego nakładu pracy; a co za tym idzie czas potrzebny na zaprogramowanie i zakodowanie problemu, który chcemy rozwiązać na maszynie, jest stosunkowo długi. Zaletą przyjętego przez nas rozwiązania jest prostota programu wprowadzającego (program wprowadzający składa się z 23 rozkazów), co przy małej pojemności pamięci jest ważne, ponadto takie rozwiązanie pozwala na stosunkowo szybkie wprowadzanie programu do maszyny, co również jest bardzo korzystne przy małej prędkości liczenia.

Z kodem maszyny typowej zapoznaliśmy się w rozdz. 2; dopóki nie znamy jednak jeszcze na pamięć tego kodu, sporządzimy tzw. tablicę kodową, która ułatwi nam szybkie przechodzenie od zapisu symbolicznego do kodu (tabl. 3-1).

Dotychczas nie zastanawialiśmy się nad metodą kodowania liczb, obecnie omówimy jedną ze stosowanych przez nas metod kodowania liczb, umożliwiającą wprowadzenie liczb razem z rozkazami. Metoda ta pozwoli nam formalnie na nierozróżnianie przy

Tablica kodowa operacji

	0	1	2	3	4	5	6	7
0	$(n)\rightarrow D$	$(n)\rightarrow$L	$n\rightarrow L$	$n\rightarrow L$ $W=1$	$(L)\rightarrow n$ $n+1\rightarrow L$	stop n	$n\rightarrow L$ $W=0$	niewykorzystany
	—	—	—	—	—	—	—	—
	—	—	—	—	—	—	—	—
1	$(A)+(n)\rightarrow A$	$(A)-(n)\rightarrow A$	$(n)\rightarrow A$	$-(n)\rightarrow A$	$(A)\rightarrow n$	$z(A)\rightarrow A$	$(M)\cdot(n)\rightarrow A$	$(A):(n)\rightarrow A_L$ reszta$\rightarrow A_p$
	$(A)<0$	$(A)<0$	$(A)\neq 0$	$(A)\neq 0$	$(A)<0$	$(A)<0$	$(A)<0$	$(A)<0$
	$(A)\geqslant 1$ albo $(A)<-1$	$(A)\geqslant 1$ albo $(A)<-1$	—	$(A)=1$	—	$(A)\geqslant 1$ albo $(A)<-1$	$(A)=1$	$\|(A)\|\geqslant\|(n)\|$
2	$(A)\cdot 2^n\rightarrow A$	$(A):2^n\rightarrow A$	$(A):2_n\rightarrow A$	$(A)\cdot 2_n\rightarrow A$	czytaj	drukuj	$\|(A)\|\rightarrow A$	$(n)\rightarrow M$
	$(A)<0$	$(A)<0$	$(A)<0$	$(A)<0$	$Y=1$	$Z=1$	$A\neq 0$	$(M)<0$
	$(A)\geqslant 1$ albo $(A)<-1$	—	—	—	—	—	$(A)=1$	—
3	$(A)\cap(n)\rightarrow A$	niewykorzystany	niewykorzystany	niewykorzystany	niewykorzystany	niewykorzystany	niewykorzystany	$(L)\rightarrow n$ $n+1\rightarrow L$ $N=1$
	$A\neq 0$							—
	—							—

wprowadzaniu liczb i rozkazów. Dowolną liczbę co do modułu mniejszą od jedności o długości 33 B w rozwinięciu binarnym (33 B odpowiada dokładności około 10 cyfr dziesiętnych) będziemy przedstawiali za pomocą pary rozkazów, które dla odróżnienia będziemy nazywali *pseudorozkazami*. Liczbę zapisaną za pomocą pary pseudorozkazów będziemy nazywali *liczbą przedstawioną w kodzie rozkazowym*. Zastanówmy się najpierw nad algorytmem przejścia z systemu dziesiętnego do systemu o zmiennej podstawie, jakim jest kod rozkazowy.

Niech będzie dana liczba rzeczywista x spełniająca warunek $2 > x \geqslant 0$ i niech będzie dany skończony ciąg liczb naturalnych większych od jedności $g_1, g_2, \ldots, g_n$. Liczbę x możemy napisać wówczas jednoznacznie w postaci

$$x = C_0 + \frac{C_1}{g_1} + \frac{C_2}{g_1 g_2} + \frac{C_3}{g_1 g_2 g_3} + \ldots + \frac{C_n}{g_1 g_2 \ldots g_n} + \frac{x_{n+1}}{g_1 g_2 \ldots g_n}, \tag{3-1}$$

gdzie

$$\left.\begin{array}{ll} C_0 = [x], & x_1 = x - C_0, \\ C_1 = [g_1 x_1], & x_2 = g_1 x_1 - C_1, \\ C_2 = [g_2 x_2], & x_3 = g_2 x_2 - C_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \\ C_n = [g_n x_n] & x_{n+1} = g_n x_n - C_n. \end{array}\right\} \tag{3-2}$$

W interesującym nas przypadku $n = 12$, a wyrazy ciągu g_i przyjmują wartości:

$$g_1 = 2, \quad g_2 = g_3 = \ldots = g_6 = 8, \quad g_7 = 4, \quad g_8 = g_9 = \ldots = g_{12} = 8. \tag{3-3}$$

W ten sposób podzieliliśmy 34-bitowe słowa długie maszyny na 13 grup:
1B + 1B + 3B + 3B + 3B + 3B + 3B + 2B + 3B + 3B + 3B + 3B + 3B = 34B.
Jeśli teraz dla naszej liczby x, korzystając z algorytmu (3-2) z wartościami na g_1 podanymi w (3-3), obliczamy $C_0, C_1 \ldots, C_{12}$ (zaokrąglając przy tym C_{12}), to otrzymamy kolejne cyfry przybliżonego rozwinięcia liczby x w kodzie rozkazowym. Przy przeliczeniu liczby z systemu dziesiątkowego do systemu kodu rozkazowego wygodnie jest korzystać z następującego schematu zapisu:

$$\left.\begin{array}{cccccccccccc} C_0 & C_1 & C_2 & C_3 & C_4 & C_5 & C_6 & C_7 & C_8 & C_9 & C_{10} & C_{11} & C_{12} \\ 2 & 8 & 8 & 8 & 8 & 8 & 8 & 4 & 8 & 8 & 8 & 8 & 8 \end{array}\right\} \tag{3-4}$$

Obliczając pomocnicze wyrażenie

$$C_1' = 2C_0 + C_1,$$

możemy napisać liczbę x za pomocą pary pseudorozkazów, z których pierwszy umieścimy pod adresem parzystym, drugi zaś pod adresem nieparzystym:

$$\left.\begin{array}{cccccc} C_7 & C_8 & C_9 & C_{10} & C_{11} & C_{12} \\ C_1' & C_2 & C_3 & C_4 & C_5 & C_6 \end{array}\right\} \tag{3-5}$$

Przedział liczb x ($2 > x \geqslant 0$) rozdzielimy na dwa podprzedziały: $1 > x \geqslant 0$ oraz $2 > x \geqslant 1$. Jeśli liczba y mniejsza co do modułu od jedności jest liczbą dodatnią, to przeliczamy z systemu dziesiętnego do systemu kodu rozkazowego liczbę $x = y$, $1 > x \geqslant 0$, jeśli zaś y jest liczbą ujemną, to przeliczamy liczbę $x = 2 + y$ ($2 > x \geqslant 1$).

Przeliczając liczby x z podprzedziału $< 2{,}1 >$ musimy pamiętać, że dla dokonania zaokrąglenia odejmiemy 4 od C_{13} (przy założeniu, że $g_{13} = 8$), nie zaś jak w przypadku podprzedziału $< 1{,}0 >$, gdzie dodajemy 4 do C_{13}.

3.1. METODA ADRESÓW WZGLĘDNYCH

Zdajemy sobie sprawę z następującej trudności kodowania: wpisując kolejne rozkazy musimy znać adresy stałe i adresy robocze, na to zaś aby znać te adresy, musimy mieć wypisane wszystkie sekwencje rozkazów programu.

Trudność tę nieraz możemy ominąć stosując tzw. adresy symboliczne, mianowicie oznaczając nieznany adres liczby a przez $< a >$. Metoda ta nie jest jednak najwygodniejsza, gdyż

1) przy dużej ilości danych jest bardzo nieprzejrzysta,

2) dla rozróżnienia słów długich i krótkich musimy używać dodatkowych symboli.

Trudność ta usuwa stosowanie tzw. adresów względnych. Adresy rozkazów wykonywanych kolejno zapisujemy jako $k + 0000$, $k + 0001$, $k + 0002$ itd. Analogicznie postępujemy z materiałem liczbowym i adresami roboczymi. Na przykład liczby umieszczamy pod kolejnymi adresami $a + 4002$, $a + 4004$ itd., adresy robocze oznaczamy $b + 0000$, $b + 0001$, $b + 0002$, itd.

Uwaga: Liczby $k, a, b, \ldots$ będziemy nazywali stałymi przeadresowania.

Po ułożeniu programu w adresach względnych przystępujemy do rozmieszczania go w pamięci wewnętrznej i zewnętrznej maszyny (w przypadku omawianej UPMC taśmę perforowaną możemy traktować jako pamięć zewnętrzną).

Rozpatrzmy kolejność rozmieszczenia programu w przypadku, gdy cały program mieści się w pamięci wewnętrznej. Najpierw rozmieszczamy program, następnie dane i wreszcie adresy robocze. Doświadczenie uczy, że znacznie wygodniej adresy robocze rozmieszczać niezależnie w pewnych ustalonych częściach pamięci. W maszynie naszej przyjęliśmy ścieżkę nr 1 (adresy 0100÷0177) za miejsce rozmieszczenia adresów roboczych(1).

W przypadku gdy liczba potrzebnych nam adresów roboczych przekracza ilość adresów na ścieżce nr 1, używamy jeszcze ścieżki nr 2 itd.

Do rozmieszczania programu w pamięci maszyny używamy specjalnych formularzy podzielonych na 64 części; każdemu formularzowi odpowiada jedna ścieżka na bębnie magnetycznym.

Po rozmieszczeniu rozkazów i materiału liczbowego przyjmujemy jednolity system adresów roboczych dla rozkazów i danych liczbowych.

Rozmieszczając program należy unikać pełnego wykorzystania pamięci, zostawiając od kilku do kilkunastu adresów wolnych, ponieważ niejednokrotnie po kontroli programu w maszynie należy poprawić program. Jeśli w wyniku kontroli okaże się, że

(1) Przy rozmieszczaniu adresów roboczych należy sprawdzić, czy nie ma kolizji z adresami roboczymi podprogramów używanych dla rozwiązania danego zagadnienia.

między rozkazami k i $k+1$ należy umieścić dodatkową grupę m rozkazów, gdzie $m+2 \leqslant n$, zaś n oznacza ilość adresów niewykorzystanych, to pod adresem $k+1$ umieszczamy rozkaz 02 $s+0000$, ($s+0000$ — pierwszy adres rezerwowy), pod adresem $s+0000$ umieszczamy rozkaz stojący poprzednio pod adresem $k+1$, pod adresami $s+0001 \div s+m$ umieszczamy dodatkowe m rozkazów, zaś pod adresem $s+m+0001$ umieszczamy 02 $k+2$.

Rozmieszczenie programu w pamięci rozpatrzymy na prostym przykładzie.

Przykład 3-1. Obliczyć wartość funkcji przy założeniu, że wszystkie parametry, wielkości pośrednie i wyniki są co do modułu mniejsze od jedności,

$$y = \frac{c_0 x - c_1 x^3 + c_2 x^5}{c_0 - c_3 x^2 + c_4 x^4 + c_5 x^6} \tag{3-6}$$

dla $x = x_1$.

Przekształcając otrzymamy postać dogodniejszą do prowadzenia obliczeń

$$y = \frac{[(c_2 x^2 - c_1)x^2 + c_0]x}{[(c_5 x^2 + c_4)x^2 - c_3]x^2 + c_0}. \tag{3-7}$$

Stałe rozmieścimy pod adresami długimi w kolejności podanej w tabl. 3-2.

Tablica 3-2

Adresy	$a+4000$	$a+4002$	$a+4004$	$a+4006$	$a+4010$	$a+4012$	$a+4014$
Zawartość adresów	c_0	c_1	c_2	c_3	c_4	c_5	x_1

Adresy robocze będziemy numerowali począwszy od $b+4000$,...

Pierwszy rozkaz: *Przesyłamy do rejestru M wielkość* x_1

$$k+0000 \quad 27 \quad a+4014.$$

Drugi rozkaz: *Tworzymy kwadrat* x_1

$$k+0001 \quad 16 \quad a+4014.$$

Po wykonaniu tego rozkazu mamy w akumulatorze dokładny iloczyn (długi).

Trzeci rozkaz: *Zaokrąglamy liczbę zawartą w akumulatorze*

$$k+0002 \quad 15 \quad 0000.$$

Zaokrąglenie to zmniejszy nam błąd maksymalny przy obliczeniu wyrażenia (3-7), dalsze zaokrąglenia możemy pominąć ze względu na naprzemienne znaki wyrażenia (3-7).

Czwarty rozkaz: *Przesyłamy przybliżoną wartość* x_1^2 *pod pierwszy (długi) adres roboczy* $b+4000$

$$k+0003 \quad 14 \quad b+4000.$$

Piąty rozkaz: *Przesyłamy przybliżoną wartość* x_1^2 *do rejestru M*

$k + 0004 \quad 27 \quad b + 4000.$

Szósty rozkaz: *Obliczamy* c_5x^2 *dla* $x = x_1$

$k + 0005 \quad 16 \quad a + 4012.$

Siódmy rozkaz: *Obliczamy* $c_5x^2 + c_4$ *dla* $x = x_1$

$k + 0006 \quad 10 \quad a + 4010.$

Ósmy rozkaz: *Umieszczamy wynik pod drugim (długim) adresem roboczym* $b + 4002$

$k + 0007 \quad 14 \quad b + 4002.$

Dziewiąty rozkaz: *Obliczamy* $(c_5x^2 + c_4)x^2$

$k + 0010 \quad 16 \quad b + 4002.$

Dziesiąty rozkaz: *Obliczamy* $(c_5x^2 + c_4)x^2 - c_3$

$k + 0011 \quad 11 \quad a + 4006.$

Jedenasty rozkaz: *Umieścimy wynik pod adresem roboczym* $b + 4002$

$k + 0012 \quad 14 \quad b + 4002.$

Dwunasty rozkaz: *Obliczamy* $[(c_5x^2 + c_4)x^2 - c_3]x^2$

$k + 0013 \quad 16 \quad b + 4002.$

Trzynasty rozkaz: *Obliczamy* $[(c_5x^2 + c_4)x^2 - c_3]x^2 + c_0$

$k + 0014 \quad 10 \quad a + 4000.$

Czternasty rozkaz: *Umieścimy wynik pod adresem roboczym* $b + 4004$ (zapisujemy pod adresem $[(c_5x^2 + c_4)x^2 - c_3]x^2c_0$ dla $x = x_1$)

$k + 0015 \quad 14 \quad b + 4004.$

Piętnasty rozkaz: *Obliczamy* c_2x^2 *dla* $x = x_1$

$k + 0016 \quad 16 \quad a + 4004.$

Szesnasty rozkaz: *Obliczamy* $c_2x^2 - c_1$

$k + 0017 \quad 11 \quad a + 4002.$

Siedemnasty rozkaz: *Umieszczamy wynik pod adresem roboczym* $b + 4002$

$k + 0020 \quad 14 \quad b + 4002.$

Osiemnasty rozkaz: *Obliczamy* $(c_2x^2 - c_1)x^2$

$k + 0021 \quad 16 \quad b + 4002.$

Dziewiętnasty rozkaz: *Obliczamy* $(c_2x^2 - c_1)x^2 + c_0$

$$k + 0022 \quad 10 \quad a + 4000.$$

Dwudziesty rozkaz: *Umieszczamy wynik pod adresem roboczym* $b + 4002$

$$k + 0023 \quad 14 \quad b + 4002.$$

Dwudziesty pierwszy rozkaz: *Przesyłamy do rejestru M* x_1

$$k + 0024 \quad 27 \quad a + 4014.$$

Dwudziesty drugi rozkaz: *Obliczamy licznik wyrażenia* (3-7)

$$k + 0025 \quad 16 \quad b + 4002.$$

Dwudziesty trzeci rozkaz: *Obliczamy wartość y dla* $x = x_1$

$$k+0026 \quad 17 \quad b+4004\,.$$

Dwudziesty czwarty rozkaz: *Zatrzymujemy maszynę* (*stop*)

$$k+0027 \quad 05 \quad 0000$$

Zakładając, że k jest parzyste, z łatwością otrzymamy związek

$$a = 30+k, \tag{3-8}$$

$$\left.\begin{matrix} k+0030 \\ k+0031 \end{matrix}\right\} \quad k+4030 = <c_0>\,,$$

$$\left.\begin{matrix} k+0032 \\ k+0033 \end{matrix}\right\} \quad k+4032 = <c_1>\,,$$

$$\left.\begin{matrix} k+0034 \\ k+0035 \end{matrix}\right\} \quad k+4034 = <c_2>\,,$$

$$\left.\begin{matrix} k+0036 \\ k+0037 \end{matrix}\right\} \quad k+4036 = <c_3>\,,$$

$$\left.\begin{matrix} k+0040 \\ k+0041 \end{matrix}\right\} \quad k+4040 = <c_4>\,,$$

$$\left.\begin{matrix} k+0042 \\ k+0043 \end{matrix}\right\} \quad k+4042 = <c_5>\,,$$

$$\left.\begin{matrix} k+0044 \\ k+0045 \end{matrix}\right\} \quad k+4044 = <x_1>\,.$$

Adresy robocze rozmieścimy na ścieżce nr 1 począwszy od pierwszej komórki tej ścieżki, czyli

$$b = 100\,.$$

Pozostaje nam jeszcze przeanalizować wykorzystanie początkowych komórek roboczych. W tym celu sporządzamy harmonogram (tabl. 3-3).

IBJ

Dn.. 29.01.59

ARKUSZ PROGRAMOWY EM

kurs programowania

str.

Program dla obliczania

$$y = \frac{c_2 x^2 - c_1 x^2 + c_0 x}{c_3 x^3 + c_2 x^2 + c_1 x^2 + c_0},$$

dla $x = x_1$, $\langle x_1 \rangle = k + 4044$,

Uwaga: k — parzyste; adresy robocze 4100 ÷ 4104.

k+000 0	27	4044	k	k+004 0			c_3	
1	16	4044	k	1				
2	15	0000		2			c_0	
3	14	4100		3				
4	27	4100		4			x_1	
5	16	4042	k	5				
6	10	4040	k	6				
7	14	4102		7				
k+001 0	16	4102		0				
1	11	4036	k	1				
2	14	4102		2				
3	16	4102		3				
4	10	4030	k	4				
5	14	4100		5				
6	16	4034	k	6				
7	11	4032	k	7				
k+002 0	14	4102		0				
1	16	4102		1				
2	10	4030	k	2				
3	14	4102		3				
4	27	4044	k	4				
5	16	4102		5				
6	17	4100		6				
7	05	0000		7				
k+003 0		c_0		0				
1				1				
2		c_1		2				
3				3				
4		c_2		4				
5				5				
6		c_3		6				
7				7				

Rys. 3-1. Arkusz programowy EM

Tablica 3-3

Harmonogram wykorzystania komórek roboczych

Kolejny rozkaz programu	4100	4102	4104
k+0000			
1			
2			
3	× × ×		
4	× × ×		
5			
6			
7		× × ×	
k+0010		× × ×	
1			
2		× × ×	
3		× × ×	

Kolejny rozkaz programu	4100	4102	4104
k+0014			
5			× × ×
6			× × ×
7			× × ×
k+0020		× × ×	× × ×
1		× × ×	× × ×
2			× × ×
3		× × ×	× × ×
4		× × ×	× × ×
5		× × ×	× × ×
6			
7			

Uwaga: Zakreślone zostały pola w naszym harmonogramie odpowiadające okresowi czasu, w którym korzystamy z danej komórki roboczej.

Na podstawie sporządzonego harmonogramu widzimy, że możemy ograniczyć ilość komórek roboczych do dwóch. Mianowicie zamiast używać komórek 4100 i 4104, możemy korzystać tylko z komórki 4100. Zastępujemy więc we wszystkich rozkazach programu, w których występuje adres 4104, adresem 4100. Otrzymany w ten sposób program możemy jednolicie adresować używając adresów względnych ze stałą przeadresowania k (przy założeniu, że k jest — parzyste). Otrzymujemy w ten sposób:

k+0000	27	k+4044
1	16	k+4044
2	15	0000
3	14	4100
4	27	4100
5	16	k+4042
6	10	k+4040
7	14	4102
k+0010	16	4102
1	11	k+4036
2	14	4102
3	16	4102
4	10	k+4030
5	14	4100
6	16	k+4034
7	11	k+4032

$k+0020$	14	4102
1	16	4102
2	10	$k+4030$
3	14	4102
4	27	$k+4044$
5	16	4102
6	17	4100
7	05	0000

Obecnie możemy zapisać nasz program na arkuszu programowym. W paru słowach postaramy się omówić stosowany przez nas arkusz programowy. Dalej będziemy nazywali ten arkusz — arkuszem programowym EM (E oznacza, że jest to arkusz przystosowany do kodu ósemkowego, *M* — oznacza, że arkusz ten jest przystosowany do zapisania rozkazów w adresach względnych). Na stosowanym arkuszu programowym liczby są zapisywane tak, jak to wspominaliśmy w punkcie 3.0, tj. w postaci dwóch pseudorozkazów. Ze względu na stosowany przez nas program wprowadzający na arkuszu programowym EM stałą przeadresowania w części adresowej rozkazu piszemy w odwrotnej kolejności niż dotychczas, mianowicie najpierw piszemy część liczbową adresu, a następnie stałą przeadresowania, znak dodawania opuszczamy. Prawidłowo wypełniony arkusz programowy EM jest przedstawiony na rys. 3-1. Jako przykładu do wypełnienia arkusza używaliśmy zakodowanego programu przedstawionego w niniejszym punkcie. Przyjęty przez nas program wprowadzający (punkt 5.1) pozwala na jednoczesne korzystanie z dwunastu stałych przeadresowania, oznaczonych literami *X, N, H, L, M, S, G, B, D, F, J, K.*

3.2. DZIAŁANIA ARYTMETYCZNE NA ROZKAZACH

Niejednokrotnie w czasie obliczeń zachodzi konieczność zmieniania wartości rozkazu (jego części adresowej lub też jego części operacyjnej). Oznaczając przez r część operacyjną rozkazu, a przez n część adresową rozkazu, liczbę odpowiadającą rozkazowi możemy zapisać w postaci

$$2^{-4}r+2^{-16}n; \tag{3-9}$$

przy czym r jest liczbą pięciobitową, n zaś jest liczbą dwunastobitową.

Dla powiększenia części adresowej rozkazu o 1 należy do liczby postaci (3-9) dodać 2^{-16}, dla powiększenia zaś części adresowej o 2, należy dodać do liczby postaci (3-9) 2^{-15}. Podobnie dla zmniejszenia części adresowej rozkazu o 1 należy dodać do liczby postaci (3-9) $2-2^{16}$, zaś dla zmniejszenia części adresowej rozkazu o 2 należy dodać do liczby postaci (3-9) $2-2^{-15}$.

Rozkazy o kodach części operacyjnej $00\div17$ są traktowane jako liczby dodatnie, rozkazy zaś o kodach części operacyjnej $20\div37$ są traktowane jako liczby ujemne. Ze względu na omawiane w punkcie 2.1 własności dodawania w arytmetyce uzupełnienio-

wej, nie zachodzi potrzeba rozróżniania przy wykonywaniu działań arytmetycznych na rozkazach znaku liczb odpowiadających rozkazom.

Przypuśćmy, że w trakcie pewnych obliczeń musimy zmienić część operacyjną rozkazu, np. rozkaz dodawania $10\,n$, musimy zastąpić rozkazem odejmowania $11\,n$; w tym celu dodajemy do liczby odpowiadającej rozkazowi $10\,n$ 2^{-4}.

3.3. SCHEMATY BLOKOWE

Celem schematów blokowych jest przedstawienie zasadniczych czynności programu przy użyciu języka graficznego. W dalszym ciągu zajmiemy się omówieniem zasad tworzenia tych schematów. Warto podkreślić, że schematy te mają dwojaką przydatność:

1) ułatwiają kodowanie problemu,

2) umożliwiają łatwe zorientowanie się w organizacji programu.

Jak wiadomo, program zakodowany jest tak mało czytelny, że nawet osoba, która układała ten program po upływie pewnego czasu zapomina wiele szczegółów i dla ponownego zorientowania się w programie musi poświęcić wiele czasu.

Podziału programu na pewne prostsze części możemy dokonać na wielu drogach; chodzi jednak o to, aby podział ten spełniał pewne warunki. Warunki te są następujące:

1. Elementy podziału muszą być niezależnie kodowalne.

2. Podział musi uwzględniać specyfikę obliczeń automatycznych, tzn. musi odróżniać kroki algorytmów numerycznych od pomocniczych operacji, służących np. do przekształcenia tych algorytmów.

3. Podział musi uwzględniać wszystkie dopuszczalne w programie kolejności wykonywania sekwencji rozkazów lub zespołów tych sekwencji oraz kryteria wyboru tych kolejności.

W dalszym ciągu będziemy rozróżniali trzy rodzaje czynności występujących w programie:

1) czynności arytmetyczne — obliczenia arytmetyczne wykonywane na liczbach lub rozkazach,

2) czynności logiczne — sprawdzanie warunków, kryteria wyboru kolejności itp.,

3) czynności organizacyjne — przesyłanie liczb i rozkazów, wprowadzanie i wyprowadzanie z maszyny itp.

Rozwiązanie zagadnienia, dla którego chcemy zbudować schemat blokowy, składa się z czynności $a_1, a_2, \ldots, a_m$. Każdą z tych czynności możemy zakwalifikować jako czynność arytmetyczną, logiczną albo organizacyjną. Ze względu na podobną strukturę czynności arytmetyczne i organizacyjne obejmiemy wspólną nazwą operatorów. Natomiast czynności logiczne nazwiemy predykatami. Operatory i predykaty są reprezentowane w programie jako pewne zespoły rozkazów. Na to aby operatory i predykaty spełniały podane trzy warunki, powinny być spełnione następujące wymagania:

a. Warunek uporządkowania operatora. Sterowanie z zewnątrz może być przekazane tylko pierwszemu rozkazowi operatora, przekazanie sterowania na zewnątrz (koniec realizowania operatora) może znajdować się tylko na końcu operatora.

wej, nie zachodzi potrzeba rozróżniania przy wykonywaniu działań arytmetycznych na rozkazach znaku liczb odpowiadających rozkazom.

Przypuśćmy, że w trakcie pewnych obliczeń musimy zmienić część operacyjną rozkazu, np. rozkaz dodawania $10\,n$, musimy zastąpić rozkazem odejmowania $11\,n$; w tym celu dodajemy do liczby odpowiadającej rozkazowi $10\,n$ 2^{-4}.

3.3. SCHEMATY BLOKOWE

Celem schematów blokowych jest przedstawienie zasadniczych czynności programu przy użyciu języka graficznego. W dalszym ciągu zajmiemy się omówieniem zasad tworzenia tych schematów. Warto podkreślić, że schematy te mają dwojaką przydatność:

1) ułatwiają kodowanie problemu,

2) umożliwiają łatwe zorientowanie się w organizacji programu.

Jak wiadomo, program zakodowany jest tak mało czytelny, że nawet osoba, która układała ten program po upływie pewnego czasu zapomina wiele szczegółów i dla ponownego zorientowania się w programie musi poświęcić wiele czasu.

Podziału programu na pewne prostsze części możemy dokonać na wielu drogach; chodzi jednak o to, aby podział ten spełniał pewne warunki. Warunki te są następujące:

1. Elementy podziału muszą być niezależnie kodowalne.

2. Podział musi uwzględniać specyfikę obliczeń automatycznych, tzn. musi odróżniać kroki algorytmów numerycznych od pomocniczych operacji, służących np. do przekształcenia tych algorytmów.

3. Podział musi uwzględniać wszystkie dopuszczalne w programie kolejności wykonywania sekwencji rozkazów lub zespołów tych sekwencji oraz kryteria wyboru tych kolejności.

W dalszym ciągu będziemy rozróżniali trzy rodzaje czynności występujących w programie:

1) czynności arytmetyczne — obliczenia arytmetyczne wykonywane na liczbach lub rozkazach,

2) czynności logiczne — sprawdzanie warunków, kryteria wyboru kolejności itp.,

3) czynności organizacyjne — przesyłanie liczb i rozkazów, wprowadzanie i wyprowadzanie z maszyny itp.

Rozwiązanie zagadnienia, dla którego chcemy zbudować schemat blokowy, składa się z czynności $a_1, a_2, \ldots, a_m$. Każdą z tych czynności możemy zakwalifikować jako czynność arytmetyczną, logiczną albo organizacyjną. Ze względu na podobną strukturę czynności arytmetyczne i organizacyjne obejmiemy wspólną nazwą operatorów. Natomiast czynności logiczne nazwiemy predykatami. Operatory i predykaty są reprezentowane w programie jako pewne zespoły rozkazów. Na to aby operatory i predykaty spełniały podane trzy warunki, powinny być spełnione następujące wymagania:

a. Warunek uporządkowania operatora. Sterowanie z zewnątrz może być przekazane tylko pierwszemu rozkazowi operatora, przekazanie sterowania na zewnątrz (koniec realizowania operatora) może znajdować się tylko na końcu operatora.

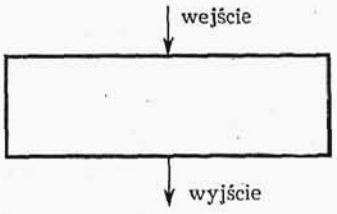
b. Warunek związania operatora. Jeżeli był wykonany pierwszy rozkaz operatora, to muszą być wykonane wszystkie pozostałe.

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

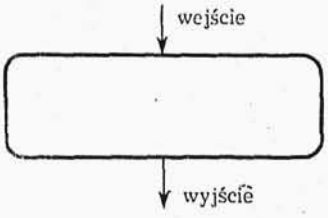
d. Warunek uporządkowania predykatu. Sterowanie z zewnątrz może być przekazane tylko pierwszemu rozkazowi predykatu, warunkowych rozkazów przekazanie sterowania na zewnątrz predykat musi mieć przynajmniej jeden.

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

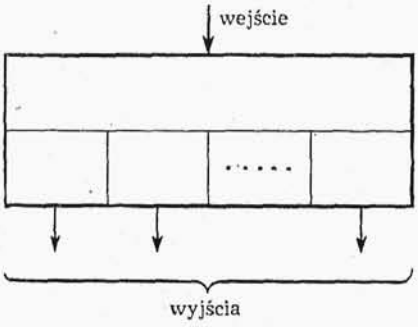
Definicja 2. Będziemy mówili krótko — wyjście operatora (predykatu), zamiast mówić rozkaz przekazania sterowania z operatora (predykatu) na zewnątrz.



Rys. 3-2. Graficzne przedstawienie operatora



Rys. 3-3. Graficzne przedstawienie operatora przeadresowywanego



Rys. 3-4. Graficzne przedstawienie predykatu

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

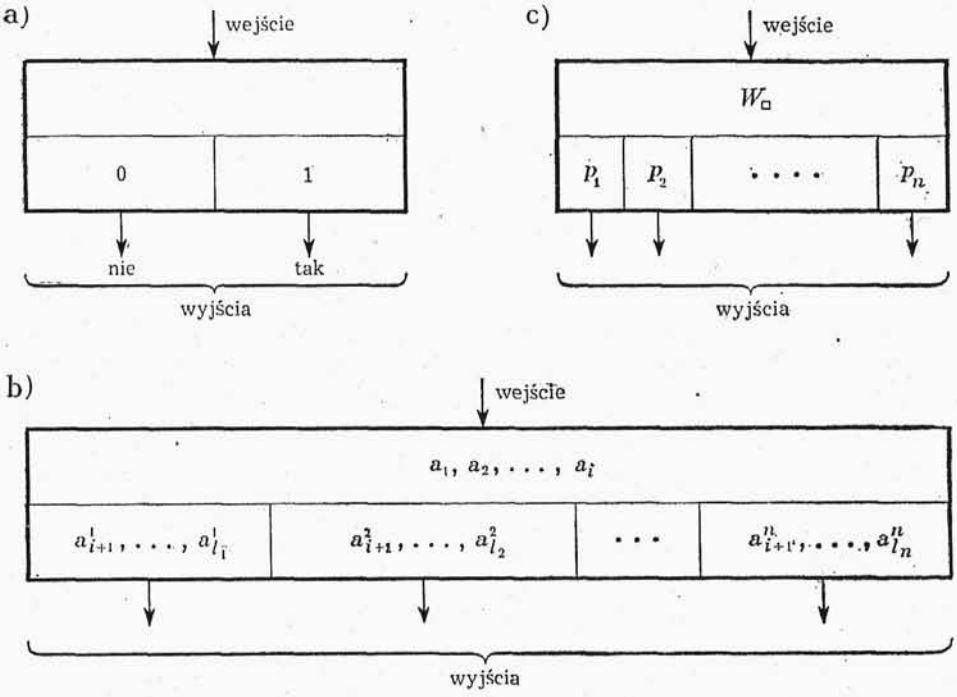
Te operatory, w których będziemy w trakcie dokonywania programu podstawiali zmienne parametry, będziemy przedstawiali graficznie jak na rys. 3-3.

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

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

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

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



Rys. 3-5. Graficzne przedstawienie predykatów

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

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

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

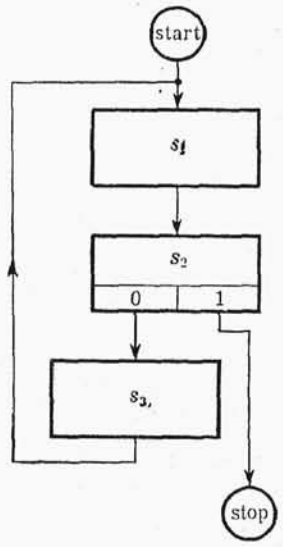
Schematy blokowe tworzymy łącząc wyjścia operatorów i predykatów z wejściami w kolejności zadanej nam przez formuły, które chcemy zrealizować. Punkty wspólne połączeń oznaczamy za pomocą kropek dla odróżnienia ich od skrzyżowań połączeń.

Przy układaniu schematów blokowych wyróżniamy dwa etapy:

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

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

Inaczej mówiąc, schemat liniowy ustala system adresowania naszego programu.



Rys. 3-6. Rysunek do przykładu 3-2

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

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

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

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

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

(1) Przykład ten podał S. Paszkowski.

3.4. PROGRAMY LINIOWE

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

3.5. PROGRAMY Z ROZWIDLENIAMI

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

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

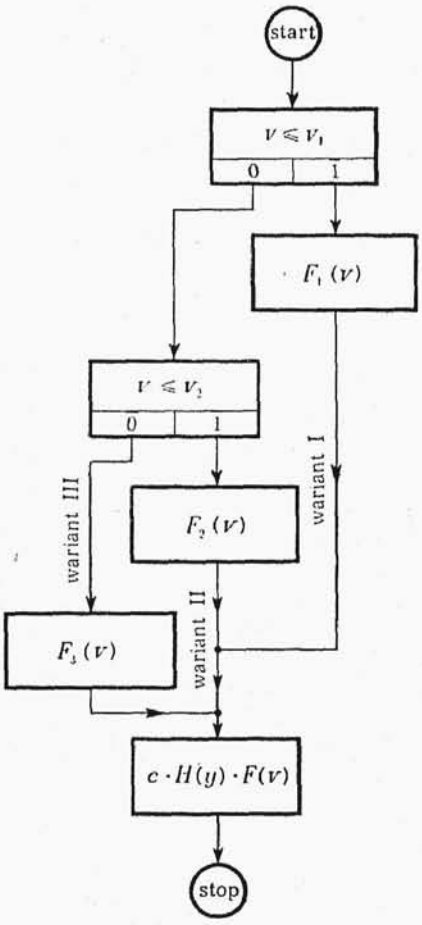
$$\mathcal{J} = c\,H(y)\;F(v).$$

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

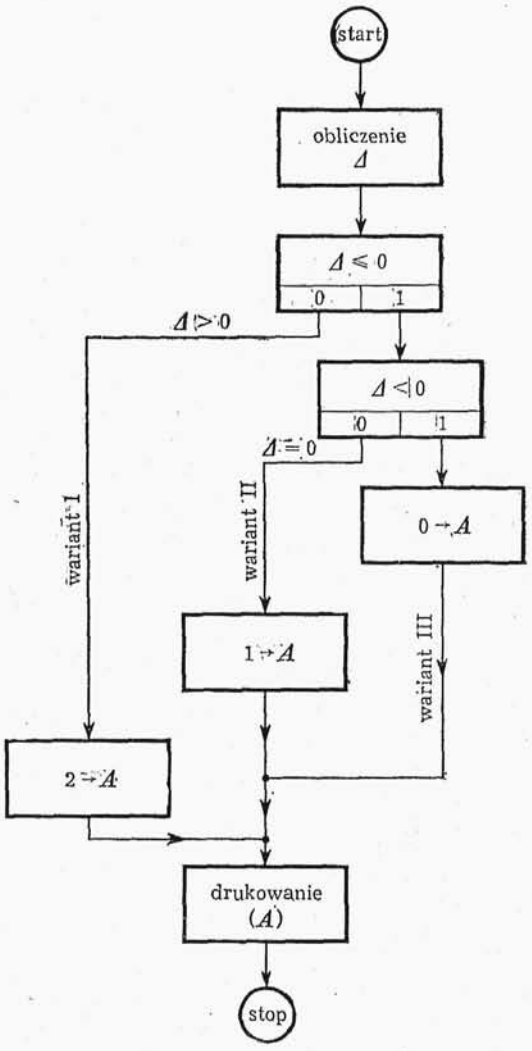
$$\begin{aligned} F(v) &= F_1(v) = a_1v^2 + b_1v + c_1 \quad \text{dla} \quad v \leqslant v_1, \\ F(v) &= F_2(v) = a_2v^2 + b_2v + c_2 \quad \text{dla} \quad v_1 < v \leqslant v_2, \\ F(v) &= F_4(v) = a_3v^2 + b_3v + c_3 \quad \text{dla} \quad v > v_2. \end{aligned}$$

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

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



Rys. 3-7. Rysunek do przykładu 3-3



Rys. 3-8. Rysunek do przykładu 3-4

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

Tablica 3-4

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

Tablica 3-5

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

Przy założeniu, że k jest parzyste, otrzymamy

$$p = k + 0024;$$

przeadresowując według powyższego związku otrzymany program w ostatecznej postaci.

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

3.6. PROGRAMY CYKLICZNE I ITERACYJNE

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

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

Z punktu widzenia obliczeń na maszynach cyfrowych ważne jest rozróżnienie:

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

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

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

W programach cyklicznych wyróżniamy pięć zasadniczych elementów:

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

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

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

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

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

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

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

B — operator służący do obliczenia przybliżenia początkowego,

C — operator wykonujący kolejny krok iteracyjny,

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

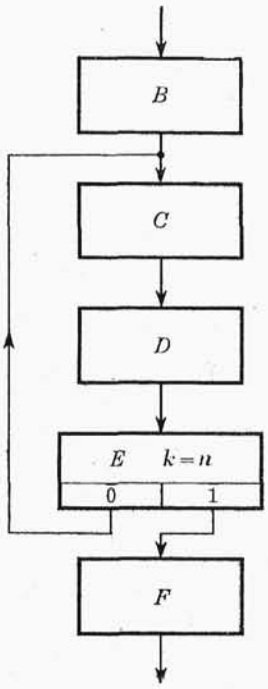
E — predykat sprawdzający dokładność wyniku po kolejnej iteracji, w zależności od tego powoduje wykonanie następnego powtórzenia iteracji albo przekazuje sterowanie kolejnemu operatorowi programu.

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

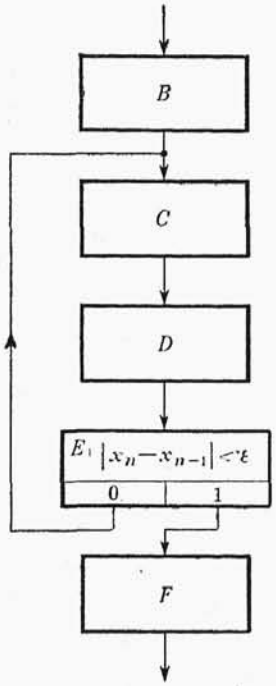
Schemat blokowy jednej z możliwych organizacji iteracji pokazany jest na rys. 3-10.

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

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



Rys. 3-9. Schemat blokowy cyklu



Rys. 3-10. Schemat blokowy iteracji

Przez czynności organizacyjne będziemy rozumieli

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

2) przeadresowanie.

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

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

Rozróżnienia programów cyklicznych od programów iteracyjnych dokonujemy z dwóch powodów:

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

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

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

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

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

z przybliżeniem początkowym

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

Tablica 3-6

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

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

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

3.7. STEROWANIE CYKLAMI

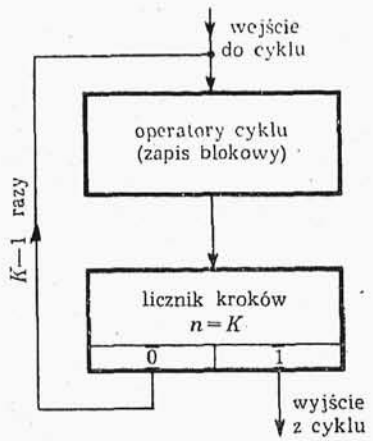
3.7.0. W niniejszym punkcie zajmiemy się szczegółowo sterowaniem cyklami; sterowanie rozwiedleniami omówimy dosyć marginesowo, przy okazji uwag nad poszczególnymi metodami sterowania cyklami.

W zasadzie rozróżniamy cztery metody sterowania cyklami;

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

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

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



Rys. 3-11. Rysunek do przykładu

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

Przykład licznika kroków:

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

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

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

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

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

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

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

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

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

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

Tablica 3-7

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

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

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

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

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

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

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

Tablica 3-8

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

Tablica 3-9

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

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

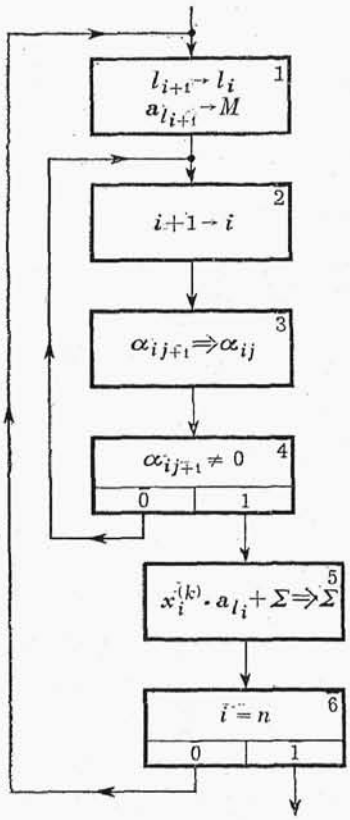
$$(\ldots(a_n x + a_{n-1})\,x + a_{n-2})\,x + \ldots a_1)\,x + a_0. \tag{3-15}$$

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

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

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

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



Rys. 3-12. Rysunek do przykładu 3-7

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

1) sterowanie cyklami,

2) sterowanie cyklem pracującym w cyklu.

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

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

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

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

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

Tablica 3-10

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

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

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

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

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

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

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

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

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),

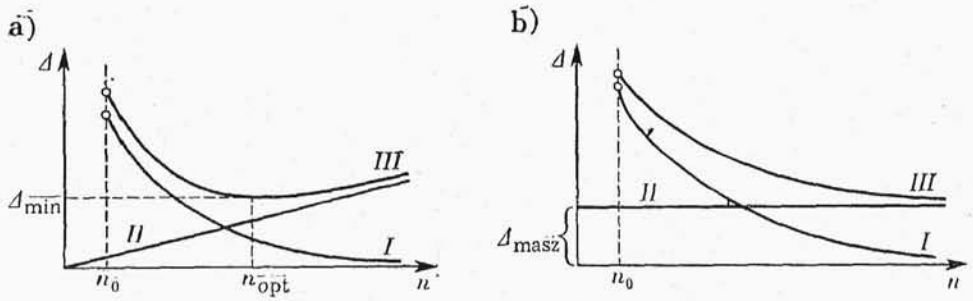
5. Metoda powinna być optymalna ze względu na czas liczenia przy ustalonej pojemności pamięci.

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.

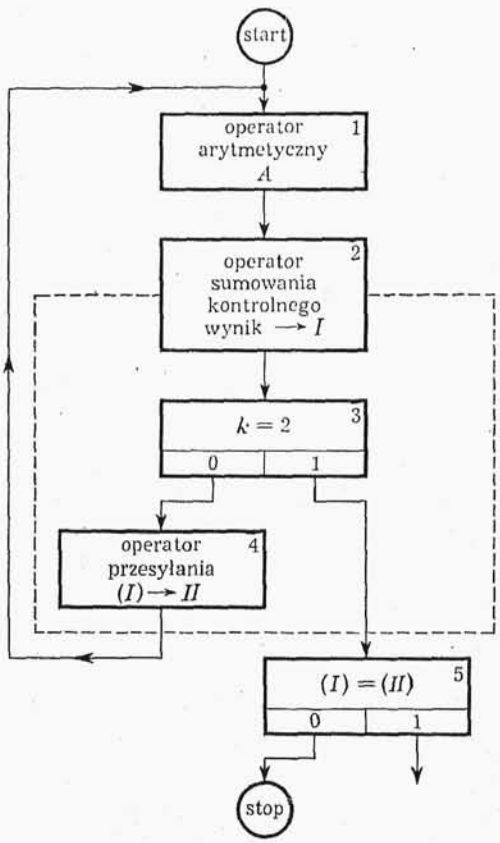
Definicja 1. Przez podprogram będziemy rozumieli zespół operatorów i predykatów, przeznaczony do wykonania zamkniętego fragmentu obliczeń (gdzie przez zamknięty fragment obliczeń rozumiemy fragment obliczeń wykonywanych według ustalonego lub nastawionego uprzednio algorytmu).

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

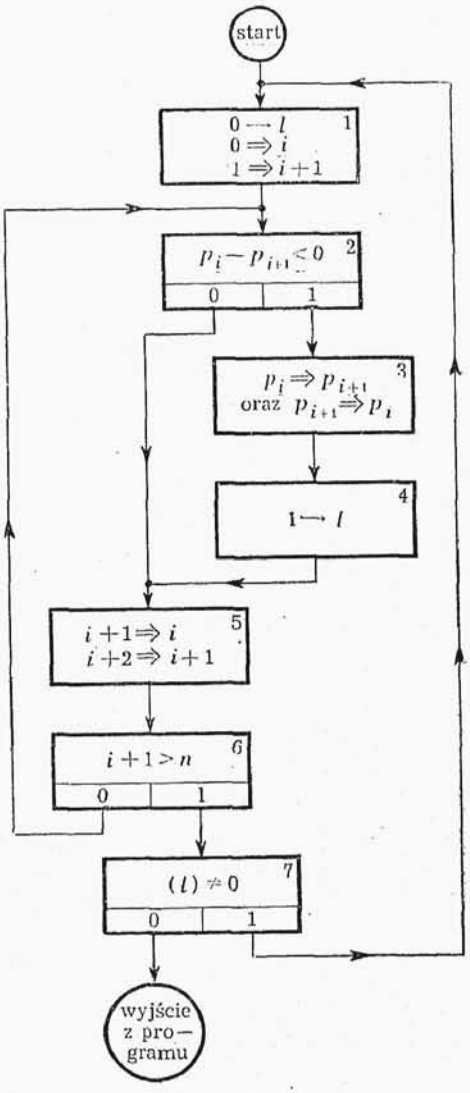
Podprogram podany w tabl. 4-2 możemy skrócić o dwa rozkazy, przyjmując następującą konwencję: stanem normalnym dalekopisu jest nastawienie go na drukowanie cyfr i znaków, po każdorazowym drukowaniu liter sprowadzamy dalekopis do stanu normalnego.

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

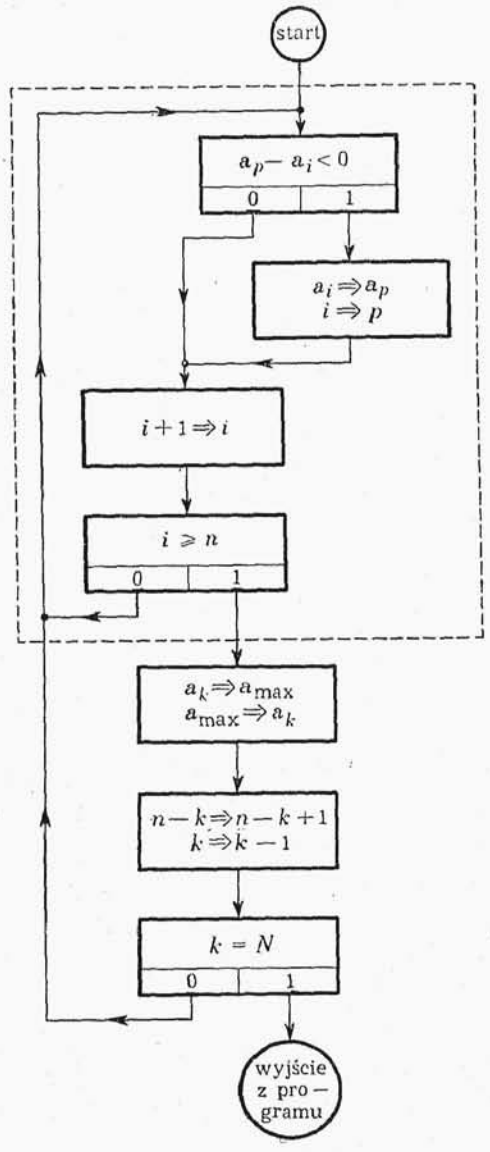
Operator lub predykat	Kolejny adres	Kolejny rozkaz	Objaśnienia
1	k+0000	12 <zero>	
	1	14 b+0001	wyzerowanie wskaźnika przestawień
	2	12 $<12<p_0>>$	
	3	04 p+0000	wywołanie podprogramu przeadresowania

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.

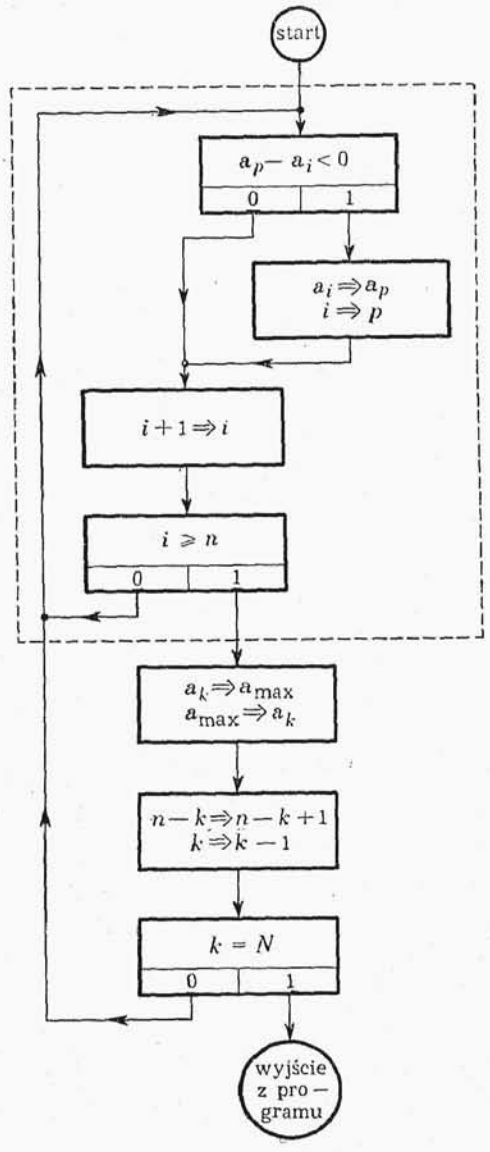
W czasie analizy problemu, w przypadku programowania ze stałym przecinkiem, wyróżniamy następujące etapy:

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-



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-

niczymy się albo do bardzo przybliżonej analizy zmienności przedziałów, albo określimy przedziały z fizycznego sensu problemu.

Sprowadzenie sumy do przedziału $<-1, 1-2^{-33}>$, gdy każdy ze składników sumy należy również do przedziału $<-1, 1-2^{-33}>$, można wykonać kilkoma metodami. Podamy obecnie jedną z takich metod podaną przez J. Kotlarskiego, dość przydatną (jak to dalej pokażemy na przykładzie) dla całkowania numerycznego. Metoda ta zapewnia maksymalną dokładność przy obliczeniach sum metodą stałego przecinka.

4.2.1. Niech będzie dany układ N liczb $a_1, a_2, a_3, \ldots, a_N$, przy czym $-1 \leqslant a_n < 1$ dla $n = 1,2,\ldots,N$. Zbudujemy ciąg średnich arytmetycznych

$$S_n = \frac{a_1 + a_2 + \ldots + a_n}{n} \quad \text{dla} \quad n = 1,2,\ldots,N;$$

wyrazy tego ciągu możemy obliczyć korzystając z wzoru rekurencyjnego

$$S_1 = a_1,$$

$$S_{n+1} = S_n + \frac{a_{n+1} - S_n}{n+1} \quad \text{dla} \quad n = 1,2,\ldots, N-1.$$

Program realizujący powyższy wzór rekurencyjny jest prosty, składa się on z 14 rozkazów. Związek między S_N a $\sum_{i=1}^{N} a_i$ jest następujący:

$$\sum_{i=1}^{N} a_i = NS_N.$$

Jak widać, $-1 \leqslant S_n < 1$ dla $n = 1,2,\ldots, N$.

W omawianym programie będziemy korzystali z następujących komórek pamięci:

$$\begin{aligned} \text{adres roboczy} &= b + 4000, \\ < S_n > &= b + 4004, \\ < \tfrac{1}{2} a_n > &= b + 4002, \\ < n \cdot 2^{-16} > &= b + 0006. \end{aligned}$$

Pod adresem $b + 0006$ przed rozpoczęciem obliczeń znajduje się 2^{-16}.

Program będzie wymagał jednej stałej równej 2^{-16}.

Zakładając, że a_n znajduje się w akumulatorze, otrzymamy następujący początek programu:

$k + 0000$		miejsce na ślad,
1	21	0001,
2	14	$b + 4002$,
$k + 0003$	12	$b + 0006$,
4	10	$< 2^{-16} >$,
5	14	$b + 0006$.

Jak łatwo widzieć, zawsze prawdziwe jest oszacowanie (przy założeniu, że $-1 \leqslant a < 1$ dla $i = 1,2,3,\ldots, N$):

$$-1 \leqslant \frac{a_{n+1} - S_n}{n+1} < 1 \qquad \text{dla } n = 1,2,\ldots, N-1;$$

$$\frac{a_{n+1} - S_n}{n+1} = \frac{2^{-16}(a_{n+1} - S_n)}{2^{-16}(n+1)} = \frac{2^{-15}\left(\frac{a_{n+1}}{2} - \frac{S_n}{2}\right)}{2^{-16}(n+1)}.$$

Wykonane przekształcenie zapewnia nam zawieranie się wszystkich wyników w przedziale $< -1, +1)$

$k+0006$	13	$b+4004$	obliczenie
7	21	0001	
$k+0010$	10	$b+4002$	$2^{-15}\left(\frac{a_{n+1}}{2} - \frac{S_n}{2}\right)$,
1	21	0017	
2	17	b+0006	$\frac{a_{n+1} - S_n}{n+1}$,
3	10	$b+4004$	S_n,
4	14	$b+4004$	$S_{n+1} \Longrightarrow S_n$,
5	01	$k+0000$	powrót do programu głównego.

Przykład 4-1. Obliczyć całkę:

$$\mathcal{J} = \int_0^1 \arcsin \sqrt{\frac{x}{1+x}}\, dx$$

z dokładnością $\varepsilon = 0{,}00001$, stosując przybliżoną metodę numerycznego całkowania, mzn metodę prostokątów. Analiza przedziałów zmienności kolejnych funkcji występujących pod całką $\mathcal{J}$ prowadzi do następujących oszacowań:

$$0 \leqslant \frac{x}{1+x} \leqslant \frac{1}{2},$$

$$0 \leqslant \sqrt{\frac{x}{1+x}} \leqslant \frac{\sqrt{2}}{2},$$

$$0 \leqslant \arcsin \sqrt{\frac{x}{1+x}} \leqslant \frac{\pi}{4}.$$

Jak widać z powyższych oszacowań, jedynym potrzebnym przekształceniem jest podzielenie w ułamku $\frac{x}{1+x}$ licznika i mianownika przez 4. Wprowadzając nową zmienną $t = \frac{1}{4}x$, otrzymamy

$$\mathcal{J} = 4\int_0^{\frac{1}{4}} \arcsin \sqrt{\frac{t}{\frac{1}{4}+t}}\, dt.$$

Jeśli oznaczymy funkcję podcałkową przez $f(t)$, to formuła prostokątów dla całki

$$\int_a^b f(t)\, dt,$$

ma postać

$$\int_a^b f(t)\mathrm{d}t \approx h\,[f(t_0) + f(t_1) + \ldots + f(t_{n-1})];$$

gdzie $t_0 = a$, $t_n = b$, $h = \dfrac{b-a}{n}$ oraz $t_j = t_0 + hj$ dla $j = 1,2,\ldots,n-1$.

W naszym przypadku przyjmiemy:

$$h = 0{,}00025, \quad t_0 = 0, \quad t_n = \tfrac{1}{4}, \quad t_j = j \cdot 0{,}00025.$$

Do obliczeń będziemy używali trzech podprogramów:

1) podprogramu do obliczania pierwiastka kwadratowego,
2) podprogramu do obliczania arc sin y,
3) podprogramu sumowania metodą średnich arytmetycznych.

Stosując metodę sumowania średnich arytmetycznych zauważmy, że przybliżona wartość całki $\mathcal{J}$ jest równa:

$$\mathcal{J} = 4hnS_n$$

gdzie S_n jest średnią arytmetyczną liczb $f(t_0),\ldots,f(t_{n-1})$.

Ponieważ $n = 1000$, zaś $h = 0{,}00025$, mamy więc

$$\mathcal{J} = S_n.$$

Z powyższego wzoru widać, że nie musimy obliczać wartości sumy, wystarczy obliczyć jej średnią arytmetyczną.

Podprogram dla obliczania pierwiastka kwadratowego składa się z 51 słów krótkich, wywołanie podprogramu wygląda następująco: liczbę, którą chcemy pierwiastkować, umieszczamy w akumulatorze, po czym wykonujemy operację 04 k + 0000 (gdzie k jest stałą przeadresowania podprogramu), po wykonaniu podprogramu wynik zostaje w akumulatorze. Podprogram korzysta z ośmiu miejsc roboczych krótkich o adresach 0100 ÷ 0107.

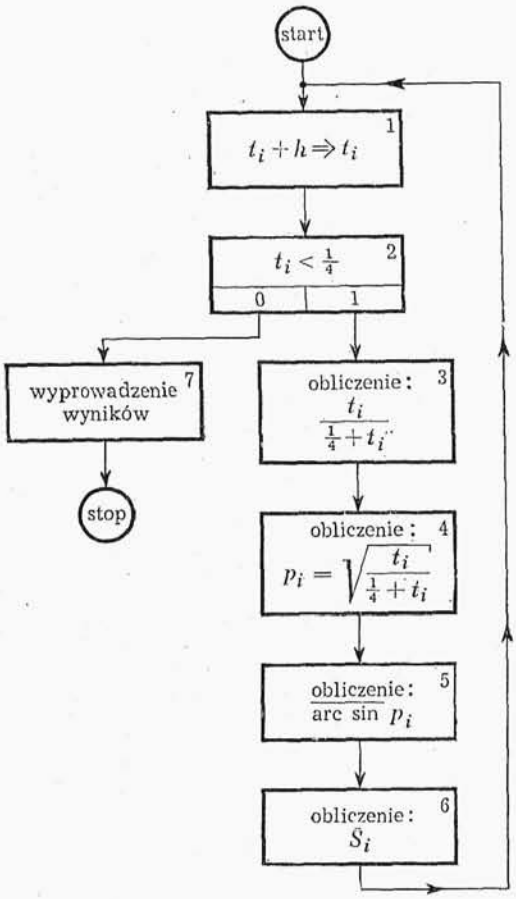
Podprogram dla obliczania arc sin y, składa się z 48 słów krótkich, wywołanie podprogramu wygląda następująco: argument umieszczamy w akumulatorze, po czym wykonujemy operację 04 k' + 0000 (gdzie k' jest stałą przeadresowania podprogramu), po wykonaniu podprogramu wynik zostaje w akumulatorze. Podprogram korzysta z czterech miejsc roboczych krótkich o adresach 0100 ÷ 0103.

Podprogram dla sumowania metodą średnich arytmetycznych składa się z 14 słów krótkich; wywołanie podprogramu wygląda następująco: kolejny wyraz przeznaczony do sumowania umieszczamy w akumulatorze, po czym wykonujemy operację 04 k'' +0000 (gdzie k'' jest stałą przeadresowania programu), po wykonaniu podprogramu S_n zostaje pod adresem 4114, $n \cdot 2^{-16}$ zaś zostaje pod adresem 0116. Podprogram korzysta z siedmiu miejsc roboczych krótkich o adresach 0110 ÷ 0116.

Schemat blokowy płaski naszego programu przedstawiony jest na rys. 4-5.

Ze względu na brak miejsca nie będziemy rysowali schematu blokowego liniowego, natomiast bezpośrednio na podstawie schematu płaskiego zakodujemy program główny, przyjmując, że operatory ostatecznego sformułowania wyniku i wyprowadzenia wy-

niku z maszyny umieścimy za pętlą cyklu, podprogramy zaś w kolejności wyżej wymienionej umieścimy za programem głównym.



Rys. 4-5. Rysunek do przykładu 4-1

Przyjmując, że program główny ma miejsca robocze podane w tabl. 4-8, kolejne operatory i predykaty programu głównego, po zakodowaniu, przedstawiamy w tabl: 4-9.

Musimy pamiętać, że miejsca robocze $b + 4002$, 4114 i 0116 przed wykonaniem programu należy wyzerować, możemy dokonać tego za pomocą operatora przygotowania, który umieszczamy przed programem głównym.

Tablica 4-8

Adres	$b+4000$	$b+4002$	$b+4004$	$b+0006$	$b+0007$
Zawartość adresu	h	t_i	$t_i+\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{2}$

Tablica 4-9

Program opracowany przy założeniu, że w chwili początkowej pod adresem $b + 4002$ znajduje się zero

Operator lub predykat	Kolejny adres	Kolejny rozkaz		Objaśnienia
1	$g+0000$	12	$b+4002$	przy wprowadzeniu programu do maszyny musimy wyzerować adres $b+4002$
	1	10	$b+4000$	
	2	14	$b+4002$	$t_i - \frac{1}{4} < 0$
2	3	11	$b+0006$	
	4	06	$g+0015$	
3	5	10	$b+0007$	$\left(t_i - \frac{1}{4}\right) + \frac{1}{2} = t_i + \frac{1}{4}$
	6	14	$b+4004$	$t_i + \frac{1}{4} \rightarrow b+4003$
	7	12	$b+4002$	$t_i \rightarrow A$
	$g+0010$	17	$b+4004$	$\frac{t_i}{\frac{1}{4} + t_i}$
4	1	04	$k+0000$	wywołanie podprogramu pierwiastka
5	2	04	$k'+0000$	wywołanie podprogramu arc sin y,
6	3	04	$k''+0000$	wywołanie podprogramu obliczenia S_n
	4	02	$g+0000$	
7	5	04	<podprogram wyprowadzający>	
8	6	05	0000	stop

Podprogramu przeliczenia i drukowania wyników nie będziemy tu omawiać, zajmiemy się nim szczegółowo w rozdz. 5.

Rozmieszczając miejsca robocze programu głównego w pamięci maszyny musimy pamiętać, że $b > 116'$.

4.3. PROGRAMY OBLICZENIOWE O ZMIENNYM PRZECINKU

4.3.0. Przez postać normalną (dwójkowo-dwójkową) liczby binarnej N będziemy rozumieli postać[1]:

$$N = 2^c \cdot m,$$

[1] Postacią normalną liczby binarnej nazywana jest również postać tzw. dwójkowo-dziesiętna, w której liczba N jest postaci

$$N = 10^x \cdot a,$$

gdzie x jest liczbą całkowitą, $-10 < a < 10$ oraz $1 \leqslant |a| < 10$ (Bibliografia [19]).

gdzie c całkowite, oraz

$$-1 < m \leqslant -\tfrac{1}{2} \text{ albo } \tfrac{1}{2} \leqslant m < 1, \tag{4-1}$$

przy czym liczby c i m spełniające podane wyżej warunki są nazywane odpowiednio cechą i mantysą liczby N.

Oczywiście, realizując taką postać w maszynie ustalimy przedział zmienności c; najczęściej stosowanym przedziałem jest przedział $<-32, +31>$. W niektórych maszynach bywa używany przedział większy, np. $<-64, +63>$.

Operacje arytmetyczne na liczbach postaci normalnej będziemy nazywali operacjami w zmiennym przecinku.

4.3.1. Postać liczb. W omawianym niżej podprogramie liczby $N \neq 0$ przedstawione są w postaci normalnej. Jeśli $N = 0$, to przyjmujemy, że $c = m = 0$.

Zapis liczby zmiennoprzecinkowej N, składający się z cechy c i mantysy m, stanowi w pamięci maszyny 34-bitową (długą) liczbę postaci:

$$2^{-5}c^* + 2^{-6}m^*, \tag{4-2}$$

gdzie

$$c^* = \begin{cases} c, & \text{jeśli } c \geqslant 0, \\ 64 + c, & \text{jeśli } c < 0, \end{cases} \qquad m^* = \begin{cases} m, & \text{jeśli } m \geqslant 0, \\ 2 + m, & \text{jeśli } m < 0. \end{cases}$$

Inaczej mówiąc, na bardziej znaczących 6 bitach słowa długiego zapamiętana jest cecha, a na pozostałych 28 bitach mantysa — obie w arytmetyce uzupełnieniowej. Oprócz zera można w ten sposób przedstawić (z około ośmioma znaczącymi cyframi dziesiętnymi) liczby spełniające nierówności (4-1) i nierówność

$$-32 \leqslant c \leqslant 31, \tag{4-3}$$

tj. liczby o wartości bezwzględnej z przedziału

$$<2^{-32}, 2^{31}> \approx <1{,}16 \cdot 10^{-10}, 2{,}14 \cdot 10^{9}>.$$

4.3.2. Program działań w zmiennym przecinku będzie się składał z następujących części:

1. Podprogramu rozformowania, który rozdzieli cechy i mantysy obu argumentów działania.

2. Podprogramów wykonujących poszczególne działania arytmetyczne i podających wynik w postaci pary pseudocechy γ i pseudomantysy μ, według następujących wzorów:

dla dodawania

$$\begin{array}{llll} \text{jeśli } c_1 \geqslant c_2, & \gamma = c_1 + 1, & \mu = \tfrac{1}{2}m_1 + 2^{-c_1+c_2-1}m_2, & \\ \text{jeśli } c_2 < c_1 & \gamma = c_2 + 1, & \mu = 2^{c_1-c_2-1}m_1 + \tfrac{1}{2}m_2; & \end{array} \tag{4-4}$$

dla odejmowania

$$\begin{array}{llll} \text{jeśli } c_1 \geqslant c_2, & \gamma = c_1 + 1, & \mu = \tfrac{1}{2}m_1 - 2^{-c_1+c_2-1}m_2, & \\ \text{jeśli } c_2 < c_1, & \gamma = c_2 + 1, & \mu = 2^{c_1-c_2-1}m_1 - \tfrac{1}{2}m_2; & \end{array} \tag{4-5}$$

dla mnożenia

$$\gamma = c_1 + c_2, \quad \mu = m_1 m_2; \tag{4-6}$$

dla dzielenia

$$\gamma = c_1 - c_2 + 1, \quad \mu = \frac{m_1}{2m_2}. \tag{4-7}$$

3. Podprogramu normalizującego pseudomantysę wyniku; obliczenie cechy wyniku następuje poprzez zsumowanie pseudocechy z liczbą całkowitą v, tak dobraną, że liczba $2^v\mu$ jest mantysą w sensie nierówności (4-1).

4. Podprogramu sformowania wyniku, czyli zakodowania go jako liczby postaci (4-2), jeśli cecha wyniku spełnia nierówność (4-3). W przypadku gdy cecha wyniku jest większa od 31, podprogram sygnalizuje to przez wywołanie podprogramu dzwonka. W przypadku gdy cecha wyniku jest mniejsza od—32, wynik kładziemy równy zeru.

4.3.3. Sposób posługiwania się programem. Podprogram działania arytmetycznego jest wywoływany za pomocą rozkazu skoku ze śladem. W chwili wykonania tego rozkazu argumenty działania powinny znajdować się: jeden w akumulatorze, drugi w komórce 4100. W związku z tym rozróżniamy dwa rodzaje działań: działania symetryczne, tj. dodawanie, mnożenie, i działania niesymetryczne, tj. odejmowanie i dzielenie „lewe" (odjemna lub dzielna w akumulatorze, odjemnik lub dzielnik w komórce 4100) i „prawe" (odjemna lub dzielna w komórce 4100, odjemnik lub dzielnik w akumulatorze). Przy działaniach symetrycznych, tj. dodawaniu i mnożeniu, jest obojętne, który składnik lub czynnik znajduje się w akumulatorze, a który w komórce 4100.

Poszczególne działania wywołuje się za pomocą następujących rozkazów:

dodawanie za pomocą rozkazu	04 k + 0000,
„lewe" odejmowanie za pomocą rozkazu	04 k + 0066,
„prawe" odejmowanie za pomocą rozkazu	04 k + 0050,
mnożenie za pomocą rozkazu	04 k + 0077,
„lewe" dzielenie za pomocą rozkazu	04 k + 0136,
„prawe" dzielenie za pomocą rozkazu	04 k + 0116.

(podprogram zmiennoprzecinkowych działań arytmetycznych mieści się w komórkach $k + 0000 \div k + 0223$).

Wynik dowolnego działania arytmetycznego znajduje się po jego wykonaniu na miejscu obu argumentów, tj. w akumulatorze i w komórce 4100.

Dla wykonania zmiennoprzecinkowego działania arytmetycznego, przy wykonywaniu ciągu działań arytmetycznych należy

1) przesłać argumenty do „pozycji początkowych",
2) wywołać podprogram tego działania,
3) przesłać wynik działania do odpowiedniej komórki.

W wielu przypadkach można (m.in. dzięki wprowadzeniu dwóch rodzajów odejmowań i dzieleń) skrócić lub w ogóle opuścić rozkazy 1) i 3).

Dla zilustrowania tej uwagi rozpatrzmy następujący przykład. Należy obliczyć wyrażenie

$$\left(\frac{a-b}{c}+\frac{d}{e-fg}\right)^2,$$

w którym liczby a, b, c, d, f, e, g są dane w zmiennoprzecinkowej postaci. Można to zrobić za pomocą następującego ciągu rozkazów:

12 <a>
14 4100
12 <b>
} przesłanie $a \rightarrow 4100$, $b \rightarrow A$ (A akumulator),

04 k+0050 „prawe" odejmowanie (odjemnik w akumulatorze) $a-b$,

12 <c> przesłanie $c \rightarrow A$,

04 k+0116 „prawe" dzielenie (dzielnik w akumulatorze) $\frac{a-b}{c}$,

14 R przesłanie $\frac{a-b}{c}$ do komórki roboczej R,

12 <f>
14 4100
12 <g>
} przesłanie $f \rightarrow 4100$, $g \rightarrow A$,

04 k+0077 mnożenie gf,

12 <e> przesłanie $e \rightarrow A$,

04 k+0066 „lewe" odejmowanie (odjemna w akumulatorze) $e-fg$,

12 <d> przesłanie $d \rightarrow A$,

04 k+0136 „lewe" dzielenie (dzielna w akumulatorze) $\frac{d}{e-fg}$,

12 R przesłanie $(R) = \frac{a-b}{c} \rightarrow A$,

04 k+0000 dodawanie $\frac{a-b}{c}+\frac{d}{e-fg}$,

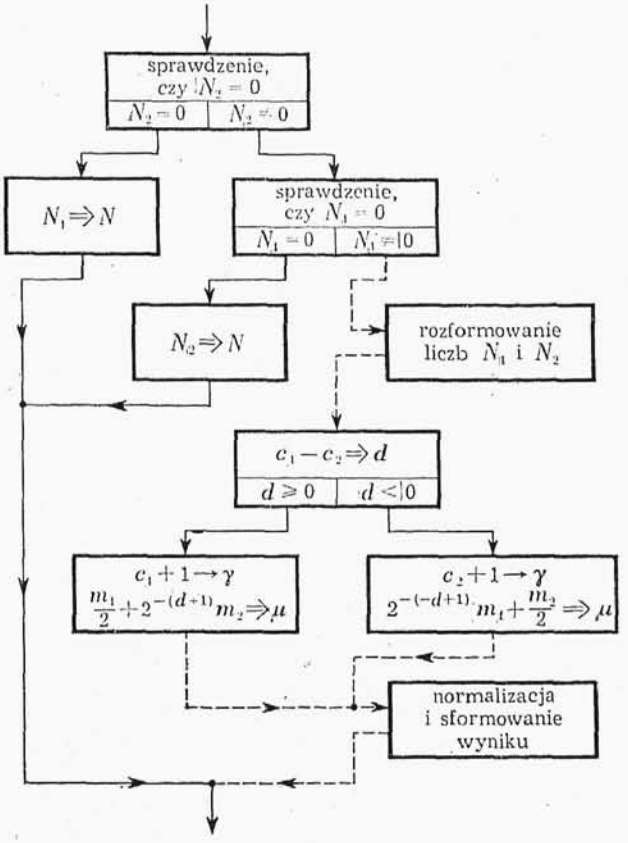
04 k+0077 mnożenie $\left(\frac{a-b}{c}+\frac{d}{e-fg}\right)\left(\frac{a-b}{c}+\frac{d}{e-fg}\right)$.

4.3.4. Uwagi. Program działań arytmetycznych na liczbach postaci zmiennoprzecinkowej wykorzystuje komórki robocze 4100, 4102, 4104 oraz parametry

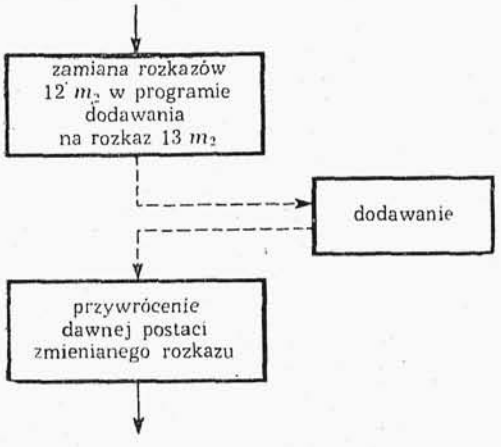
01 0000 = (0066),
00 0001 = (0076),
00 0000 = (0077),

z pamięci stałej. Część tego programu można wykorzystać dla przekształcenia liczb danych w postaci stałoprzecinkowej do postaci zmiennoprzecinkowej. Przed wywołaniem tego podprogramu za pomocą rozkazu 04 k+0162, należy wyzerować komórkę 0104 i umieścić przekształcaną liczbę w akumulatorze. Wynik przekształcenia znajduje się także w komórce 4100.

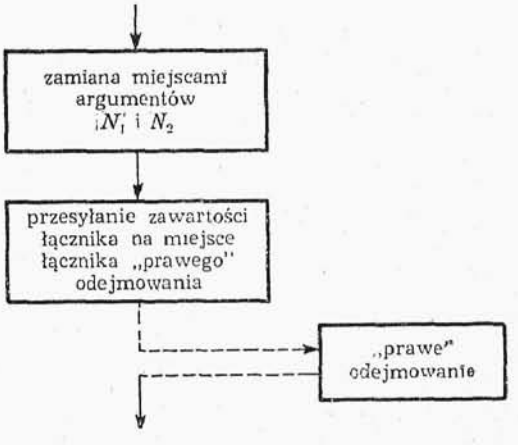
W tablicy 4-10 podajemy średnią liczbę rozkazów składającą się na wykonanie działań arytmetycznych dla poszczególnych postaci liczb. Dane przedstawione w tabl. 4-10



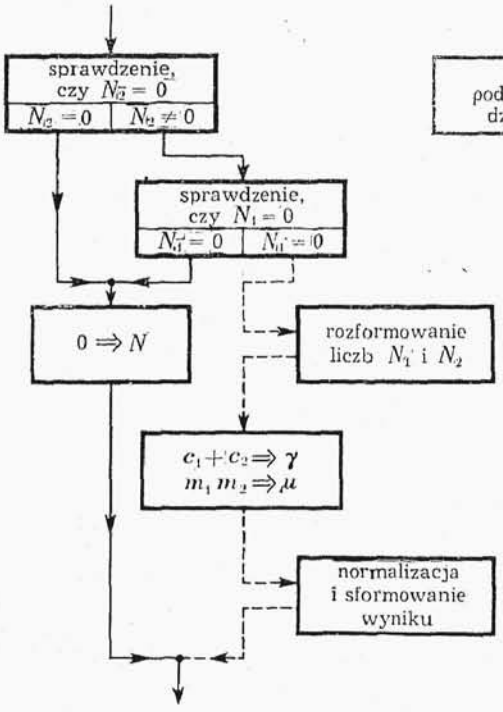
Rys. 4-6. Schemat blokowy sumowania w zmiennym przecinku



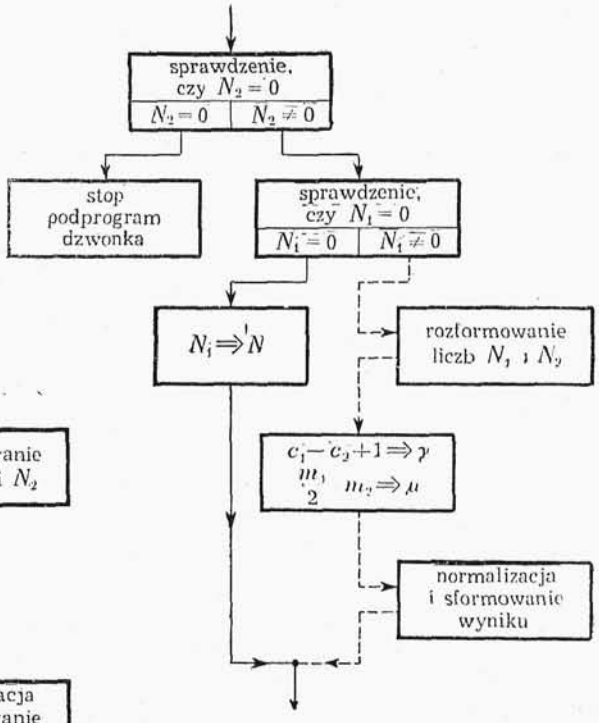
Rys. 4-7. Schemat blokowy podprogramu „prawego" odejmowania zmiennoprzecinkowego



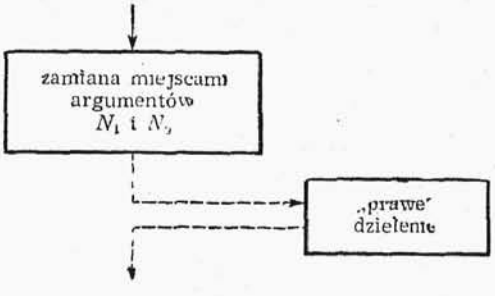
Rys. 4-8. Schemat blokowy podprogramu „lewego" odejmowania zmiennoprzecinkowego



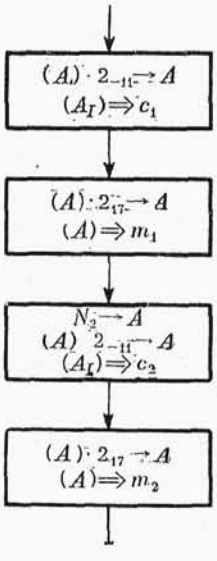
Rys. 4-9. Schemat blokowy podprogramu mnożenia liczb zmiennoprzecinkowych



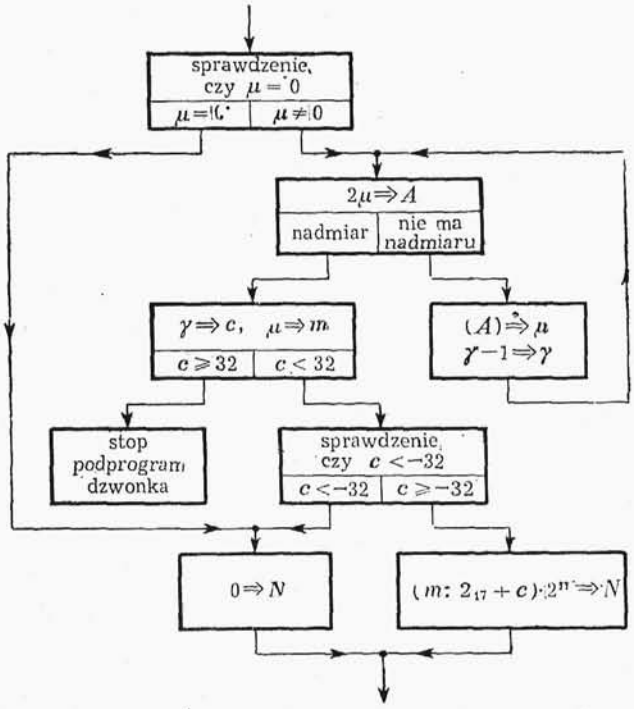
Rys. 4-10. Schemat blokowy podprogramu „prawego" dzielenia liczb zmiennoprzecinkowych



Rys. 4-11. Schemat blokowy podprogramu „lewego" dzielenia liczb zmiennoprzecinkowych



Rys. 4-12. Schemat blokowy podprogramu rozformowania argumentów liczb zmiennoprzecinkowych



Rys. 4-13. Schemat blokowy podprogramu normalizacji i sformowanie wyniku działań zmiennoprzecinkowych

Tablica 4-10

Działanie	Przypadek	Liczba rozkazów
dodawanie	$N_1, N_2 \neq 0$ $N_1 = 0$ lub $N_2 = 0$	$46+7v$ 7
„lewe" odejmowanie	$N_1, N_2 \neq 0$ $N_1 = 0$ lub $N_2 = 0$	$67+7v$ 28
„prawe" odejmowanie	$N_1, N_2 \neq 0$ $N_1 = 0$ lub $N_2 = 0$	$60+7v$ 21
mnożenie	$N_1, N_2 \neq 0$ $N_1 = 0$ lub $N_2 = 0$	40 6
„lewe" dzielenie	$N_1, N_2 \neq 0$ $N_2 = 0$	51 15
„prawe" dzielenie	$N_1, N_2 \neq 0$ $N_1 = 0$	43 7
Przejście od liczb stałoprzecinkowych do zmiennoprzecinkowych		$14+7v$

pozwalają na oszacowanie czasów potrzebnych na wykonanie działań zmiennoprzecinkowych.

Liczba całkowita nieujemna v wyznaczająca czas przejścia od pseudomantysy (ewentualnie od liczby stałoprzecinkowej do jej mantysy) jest z definicji równa liczbie zer w pseudomantysie między przecinkiem a jej pierwszą cyfrą znaczącą (w układzie dwójkowym).

Na rysunkach 4-6÷4-13 przedstawione są schematy blokowe poszczególnych części jednoadresowego programu zmiennego przecinka, liniami przerywanymi oznaczono wywoływanie podprogramów i powrót po wykonaniu podprogramu. W załączniku 4 (umieszczonym na końcu książki) podany jest omawiany wyżej program działań zmiennego przecinka.

4.3.5. Przykład programu korzystającego z jednoadresowych podprogramów zmiennego przecinka[1]. Niech będzie dany układ n równań algebraicznych liniowych:

$$X = AX+F, \tag{4-8}$$

Elementy macierzy A i współrzędne wektora F są liczbami zmiennoprzecinkowymi z przedziału przyjętego przez jednoadresowy program zmiennego przecinka (punkt 4.3.1), gdzie $i,j, = 1,2, \ldots, n$.

Przypuśćmy, że rozwiązanie układu (4-8) chcemy znaleźć za pomocą metody iteracyjnej Seidla określonej za pomocą wzorów

$$x_i^{(k)} = \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} + \sum_{j=1}^{n} a_{ij}x_j^{(k-1)} + f_i, \tag{4-9}$$

gdzie $i = 1,2, \ldots, n$, $k = 1,2, \ldots, l$.

Zakładamy oczywiście, że metoda jest zbieżna. Przybliżenie początkowe $X^{(0)}$ określamy dowolnie; przyjmując na przykład, że równa się ono wektorowi wyrazów wolnych F.

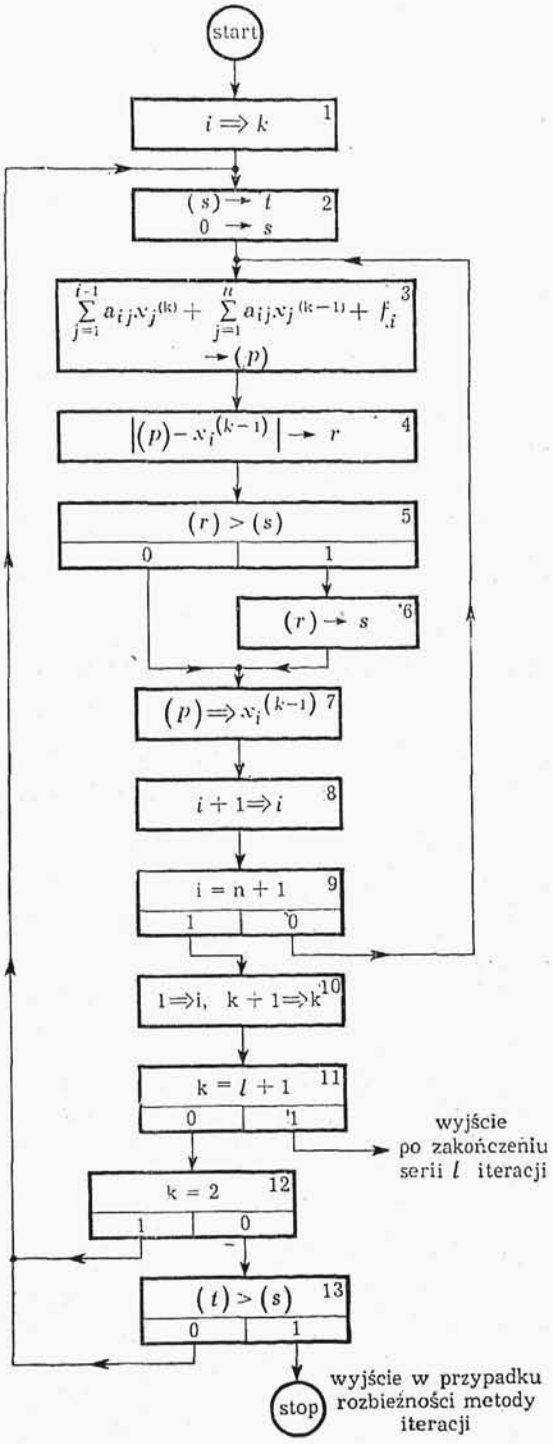
Za kryterium rozbieżności przyjmiemy następujący związek:

$$\max_{1\leqslant i\leqslant n} \left| x_i^{(k)} - x_i^{(k-1)} \right| \geqslant \max_{1\leqslant i\leqslant n} \left| x_i^{(k-1)} - x_i^{(k-2)} \right|. \tag{4-10}$$

Program realizujący metodę iteracyjną Seidla korzysta z n^2+2n komórek pamięci (długich), w których kolejno znajdują się wielkości $f_1, f_2, \ldots, f_n, x_1, x_2, \ldots, x_n, a_{11}, a_{12}, \ldots, a_{1n}, \ldots, a_{n1}, a_{n2}, \ldots, a_{nn}$. Ponadto program korzysta z czterech komórek długich oznaczonych literami p,r, c, t oraz z jednej komórki krótkiej, w której umieszczamy bieżącą wartość k (liczbę wykonanych kroków iteracji). Przed wywołaniem podprogramu, program główny umieszcza parametry $00\ l+1$ (l—ilość kroków iteracji, które należy wykonać) oraz 12 a_{11} odpowiednio pod adresami 0106, 0107, w akumulatorze zaś parametr $2n \cdot 2^{-17}$ (n-ilość równań). Przyjęto, że komórki robocze mają następujące adresy:

$$p = 4112,$$
$$r = 4114,$$
$$s = 4116,$$
$$t = 4120,$$
$$k = 0111.$$

[1] Przykład opracowany przez Z. Jankowską.



Rys. 4-14. Rysunek do przykładu

Tablica 4-11

Operator lub predykat	Kolejny adres	Kolejny rozkaz		Objaśnienia
1	$k+0000$	77	7777	miejsce na ślad
	1	14	0110	$2n \longrightarrow 0110$
	2	12	0076	$l \longrightarrow k$
	3	14	0111	
2	4	12	4116	$(s) \longrightarrow t$
	5	14	4120	
	6	12	0077	$0 \longrightarrow s$
	7	14	4116	
	$k+0010$	12	0107	formowanie rozkazów w pętli sumowania
	1	14	$k+0024$	
	2	11	0110	
	3	14	$k+0026$	
	4	14	$k+0044$	
	5	10	0031	formowanie rozkazów w pętli równania
	6	14	$k+0057$	
	7	11	0031	formowanie rozkazów w pętli równania
	$k+0020$	11	0110	
	1	14	$k+0022$	
3	2	12	7777	przesłanie x_i do p
	3	14	4112	
	4	12	7777	
	5	14	4100	
	6	12	7777	
	7	04	$k_{019}+77$	wywołanie mnożenia zmiennoprzecinkowego
	$k+0030$	12	4112	
	1	04	k_{019}	wywołanie dodawania zmiennoprzecinkowego
	2	14	4112	
3	3	12	$k+0024$	k_{019} — stała przeadresowania podprogramu zmiennoprzecinkowego
	4	10	0067	
	5	14	$k+0024$	
	6	11	0110	
	7	14	$k+0026$	
	$k+0040$	11	0107	
	1	03	$k+0024$	
4	2	12	4112	
	3	14	4100	przygotowanie odejmowania zmiennoprzecinkowego
	4	12	7777	
	5	04	$k_{019}+66$	wywołanie „lewego“ odejmowania
	6	04	$k_{019}+226$	wywołanie operacji wartości bezwzględnych zmiennoprzecinkowych
	7	14	4114	
5	$k+0050$	12	4116	
	1	04	$k_{019}+50$	wywołania „prawego“ odejmowania zmiennoprzecinkowego
	2	23	0000	
	3	03	$k+0056$	

Tablica 4-11 (d. c.)

Operator lub predykat	Kolejny adres	Kolejny rozkaz		Objaśnienia
6	$k+0054$	12	4114	przesłanie zawartości r do s
	5	14	4116	
7	6	12	4112	przesłanie zawartości p na miejsce x_i
	7	14	7777	
8	$k+0060$	12	$k+0026$	
	1	11	0110	
	2	14	$k+0026$	
	3	12	$k+0057$	
	4	10	0067	przeadresowanie pętli
	5	14	$k+0057$	
	6	12	$k+0022$	
	7	10	0067	
	$k+0070$	14	$k+0022$	
	1	11	0110	
	2	14	$k+0044$	
9	$k+0073$	11	0107	wyjście z pętli równania lub powtórzenie jej
	4	03	$k+0022$	
10	5	12	0111	powiększenie zawartości licznika iteracji
	6	10	0076	
	7	14	0111	
11	$k+0100$	11	0106	sprawdzenie krotności iteracji
	1	03	$k+0103$	
	2	01	$k+0000$	
12	3	10	0106	sprawdzenie, czy zachodzi warunek $k = 2$
	4	11	0067	
	5	03	$k+0004$	
13	$k+0106$	12	4120	
	7	14	4100	
	$k+0110$	12	4116	sprawdzenie zbieżności iteracji, wywołanie „lewego" odejmowania zmiennoprzecinkowego
	1	04	$k_{019}+66$	
	2	23	0006	
	3	03	$k+0004$	

Schemat blokowy programu jest podany na rys 4-14. Program ten jest ekonomiczny tylko w przypadku, gdy znaczna większość współczynników w macierzy $||a_{ij}||$ jest różna od zera.

W przypadku gdy wśród współczynników macierzy $||a_{ij}||$ jest dużo zer, stosujemy program bardziej złożony, wybierający niezerowe współczynniki macierzy $||a_{ij}||$ za pomocą skal logicznych.

Szczegółowo omówimy kolejne rozkazy programu w tabl. 4-11.

4.4. METODA PROGRAMOWANEGO PRZECINKA

Dotychczas omówiliśmy dwie metody obliczeniowe: stało- i zmiennoprzecinkową. Jak już podkreśliliśmy, zasadniczą wadą pierwszej z tych metod jest mała dokładność obliczeń, wadą drugiej zaś jest mała prędkość liczenia. Obecnie omówimy jeszcze jedną metodę zwaną metodą programowanego przecinka lub metodą zmiennych skali. Idea tej metody jest następująca: Rozbijamy obliczenia na takie etapy, aby przy wykonywaniu każdego z tych etapów można było prowadzić obliczenia ze stałym przecinkiem, odpowiednie przenormowania liczb następuje przy przejściu z jednego etapu do drugiego. Oczywiście, jeśli całe obliczenia będziemy traktowali jako jeden etap, to będziemy po prostu rozwiązywali zagadnienie w stałym przecinku, jeśli zaś będziemy każde działanie arytmetyczne traktowali jako odrębny etap obliczeniowy, będziemy prowadzili obliczenia ze zmiennym przecinkiem. Podobnie jak przy obliczeniach stałoprzecinkowych, wprowadzimy obecnie mnożniki skalujące, jeśli mamy jakąś wielkość y_i, co do modułu większą lub równą jedności, będziemy w obliczeniach zastępowali ją wielkością $Y_i = \mu_i y_i$ gdzie $|Y_i|<1$, zaś $0<\mu_i<1$. Podobnie dla wielkości x_i bliskich zeru wprowadzimy wielkość $X_i = \frac{x_i}{\nu_i}$, gdzie $0<r\leqslant|X_i|<1$, zaś $0<\nu_i<1$, r jest wyznaczone przez założoną dokładność. Wielkości μ_i, ν_i będziemy nazywali mnożnikami skalującymi.

W odróżnieniu od stałego przecinka, gdzie z góry ustaliliśmy wartość mnożników skalujących, przy obliczeniach z programowanym przecinkiem ustalamy tylko kryteria wyboru przez program tych mnożników. Przedstawiona dalej metoda programowanego przecinka jest stosowana w Centrum Obliczeniowym Uniwersytetu Moskiewskiego (Bibliografia [18]).

Wielkości występujące w obliczeniach przedstawimy w postaci:

$$y_i = 2^{p_i} Y_i, \tag{4-11}$$

gdzie p_i jest liczbą całkowitą, Y_i zaś spełnia nierówność:

$$|Y_i|<1. \tag{4-12}$$

Mnożnikami skalującymi w tym przypadku są wielkości 2^{-p_i}.

W odróżnieniu od omawianego w punkcie 4.3 zmiennego przecinka wielkości y_i będą przechowywane w dwóch komórkach pamięci: Y_i w komórce długiej i p_i w komórce krótkiej. Wielkości Y_i będziemy nazywali mantysą liczby y_i, wielkość p_i zaś skalą liczby y_i. Proces rozwiązywania przez maszynę zagadnienia rozbijamy na takie etapy, aby skale przy realizacji każdego z etapów były ustalone. Przy przejściu od jednego etapu obliczeń do następnego następuje przeskalowanie wielkości. Czynność tę wykonuje specjalna część programu. Warunki określające prawidłowy wybór skali określamy następująco.

Wielkości Y_i po przeskalowaniu spełniają nierówność

$$0<r\leqslant|Y_i|<R\leqslant 1, \tag{4-13}$$

przy czym wielkość R dobiera się tak, aby w czasie wykonywania kolejnego etapu obliczeń nie nastąpiło przekroczenie zakresu, wielkość r zaś określa się na podstawie założeń dotyczących dokładności obliczeń.

Podobnie jak dla działań zmiennoprzecinkowych (punkt 4.3) działania arytmetyczne na liczbach postaci $x = 2^p X$, $y = 2^q Y$ określone są wzorami

$$xy = 2^{p+q}(XY), \tag{4-14}$$

$$\frac{x}{y} = 2^{p-q}\left(\frac{X}{Y}\right), \tag{4-15}$$

$$x \pm y = 2^p(X \pm 2^{-(p-q)} Y), \quad \text{jeśli} \quad p \geqslant q, \tag{4-16}$$

$$x \pm y = 2^q(2^{-(p-q)} X \pm Y), \quad \text{jeśli} \quad p < q. \tag{4-17}$$

Przy dzieleniu należy pamiętać, że skale p i q należy dobrać tak, aby $\left|\frac{X}{Y}\right| < 1$, podobnie przy dodawaniu i odejmowaniu należy tak dobrać skale, aby przy wykonaniu działań w nawiasach nie nastąpiło przekroczenie zakresu.

Należy podkreślić, że nie wszystkie liczby występujące w obliczeniach przedstawiamy w postaci par: *mantysa—skala*; mianowicie wszystkie stałe występujące w obliczeniach, podobnie jak przy obliczeniach stałoprzecinkowych, mnożymy przez tak dobrane mnożniki, aby liczby te były co do modułu mniejsze od jedności, np. $\frac{e}{4}$, $\frac{\pi}{4}$, $\ln 2$, $\frac{2}{\pi}$ itp. Odwrotności mnożników, które użyliśmy do przenormowania stałych, uwzględniamy przekształcając wielkości zmienne, które w trakcie obliczeń są mnożone przez wyżej wymienione stałe.

Szczególnie wygodnie jest stosować metodę programowanego przecinka przy całkowaniu równań różniczkowych zwyczajnych oraz przy rozwiązywaniu równań przestępnych. Trudniej jest stosować tę metodę przy rozwiązywaniu układów równań algebraicznych liniowych. W zasadzie można jednak wszelkie problemy obliczeniowe rozwiązywać za pomocą programowanego przecinka, nie zawsze jednak jest to opłacalne, ze względu na ilość pracy, jaką trzeba włożyć dla odpowiedniego przekształcenia algorytmu.

Jako przykład rozpatrzymy całkowanie równań różniczkowych zwyczajnych metodą Rungego—Kutty—Gilla (Bibliografia [19]). Program dla tej metody z programowanym przecinkiem składa się z czterech części:

1) sterowanie obliczeniami i kontrola prawidłowości obliczeń,
2) obliczenie prawych stron równań,
3) wykonanie jednego kroku całkowania,
4) kontrola i zmiana skali po wykonaniu każdego kroku obliczeniowego.

Dany jest układ równań

$$\left.\begin{aligned} y_1' &= f_1(y_1, y_2, \ldots, y_n), \\ y_2' &= f_2(y_1, y_2, \ldots, y_n), \\ &\cdots\cdots\cdots\cdots \\ y_n' &= f_n(y_1, y_2, \ldots, y_n), \end{aligned}\right\} \tag{4-18}$$

W przedstawionym wyżej układzie zmienna niezależna nie występuje w postaci jawnej. W przypadku układu, w którym zmienna niezależna występuje w postaci jawnej,

przez dołączenie jednego równania, w którym pochodna zmiennej niezależnej jest równa jedności, przechodzimy do układu podanej postaci.

Wzory na numeryczne całkowanie metodą Rungego-Kutty-Gilla dla kroku o długości h mają postać:

$$\left.\begin{aligned} k_{i0} &= 2^m h f_i(y_{00}, y_{10}, \dots, y_{n0}),\\ r_{i1} &= \tfrac{1}{2}(k_{i0}-q_{i0}),\\ y_{i1} &= y_{i0}+2^{-m}r_{i1},\\ q_{i1} &= q_{i0}+3\cdot 2^m(2^{-m}r_{i1})-\tfrac{1}{2}k_{i0}, \end{aligned}\right\} \quad \text{dla } i=1,2,\dots,n, \tag{4-19a}$$

$$\left.\begin{aligned} k_{i1} &= 2^m h f_i(y_{01}, y_{11}, \dots, y_{n1}),\\ r_{i2} &= (1-\sqrt{\tfrac{1}{2}})\,(k_{i1}-q_{i1}),\\ y_{i2} &= y_{i1}+2^{-m}r_{i2},\\ q_{i2} &= q_{i1}+3\cdot 2^m(2^{-m}r_{i2})-(1-\sqrt{\tfrac{1}{2}})k_{i1}, \end{aligned}\right\} \quad \text{dla } i=1,2,\dots,n, \tag{4-19b}$$

$$\left.\begin{aligned} k_{i2} &= 2^m h f_i(y_{02}, y_{i2}, \dots, y_{n2}),\\ r_{i3} &= (1+\sqrt{\tfrac{1}{2}})(k_{i2}-q_{i2}),\\ y_{i3} &= y_{i2}+2^{-m}r_{i3},\\ q_{i3} &= q_{i2}+3\cdot 2^m(2^{-m}r_{i3})-(1+\sqrt{\tfrac{1}{2}})k_{i2}, \end{aligned}\right\} \quad \text{dla } i=1,2,\dots,n, \tag{4-19c}$$

$$\left.\begin{aligned} k_{i3} &= 2^m h f_i(y_{03}, y_{13}, \dots, y_{n3}),\\ r_{i4} &= \tfrac{1}{6}(k_{i3}-2q_{i3}),\\ y_{i4} &= y_{i3}+2^{-m}r_{i4},\\ q_{i4} &= q_{i3}+3\cdot 2^m(2^{-m}r_{i4})-\tfrac{1}{2}k_{i3}, \end{aligned}\right\} \quad \text{dla } i=1,2,\dots,n, \tag{4-19d}$$

Na początku obliczeń y_{i0} przyjmujemy równe warunkom początkowym, q_{i0} zaś kładziemy równe zeru. Przy przejściu od s-tego kroku całkowania do $s+1$-ego kładziemy y_{i0} i q_{i0} kroku $s+1$ równe y_{i4} i q_{i4} kroku s-tego. Wielkość m dobieramy tak, aby przy obliczeniu $2^m h f_i\,(y_1, \dots, y_n)$ dla $i = 1,2, \dots, n$ i wszelkich dopuszczalnych wartości y_1, $y_2, \dots, y_n$ nie nastąpiło przekroczenie zakresu. Mnożnik 2^m wprowadzimy dla zmniejszenia błędów zaokrągleń.

Obecnie nadamy wzorom (4-19) postać dogodną do obliczeń z programowanym przecinkiem.

Dokonamy podstawień

$$\left.\begin{aligned} y_{ij} &= 2^{p_i} Y_{ij},\\ r_{ij} &= 2^{p_i} R_{ij},\\ k_{ij-1} &= 2^{p_i} K_{ij-1},\\ q_{ij} &= 2^{p_i} Q_{ij}, \end{aligned}\right\} \tag{4-20}$$

oraz

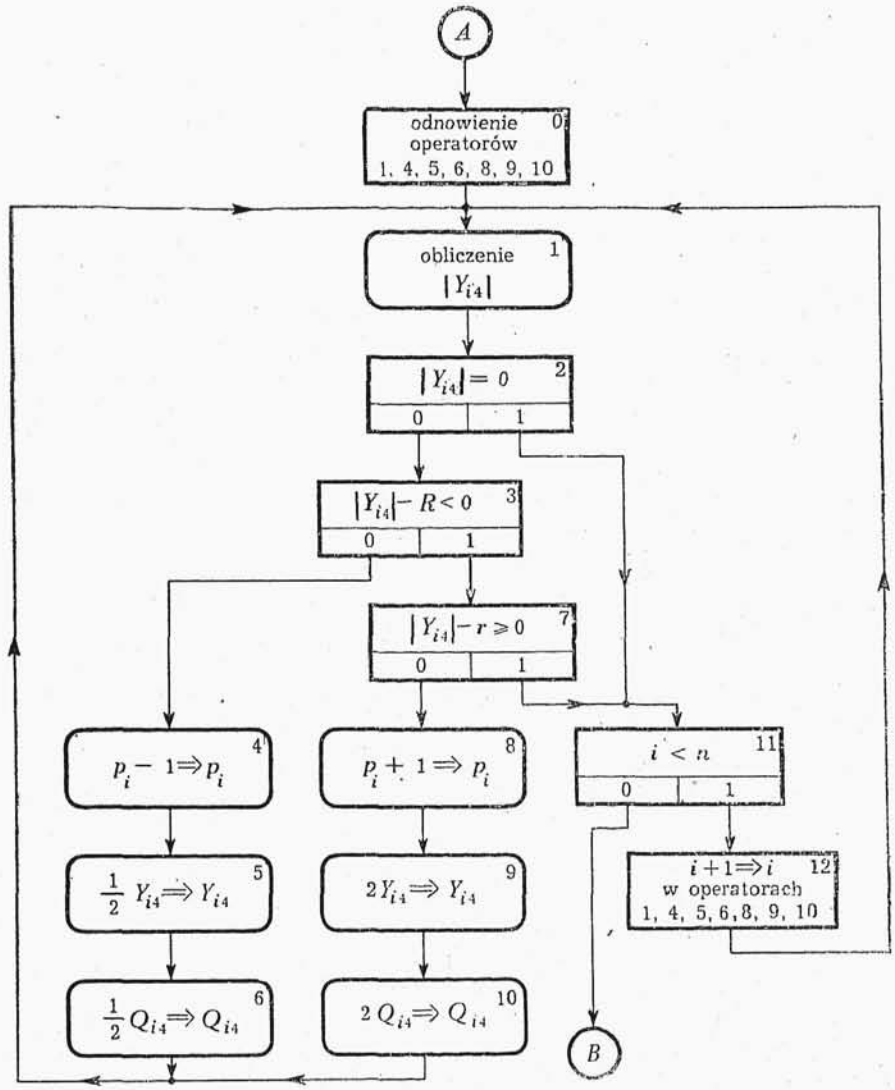
$$h = 2^{p_h} H, \qquad |H|<1, \tag{4-21}$$

oraz

$$f_i(y_{1j}, y_{2j}, \dots, y_{nj}) = 2^{l_{ij}} F_{ij}, \qquad |F_{ij}|<1, \tag{4-22}$$

gdzie $j = 1,2,3,4$.

Podstawiając wyrażenie (4-20), (4-21) i (4-22) do wzorów (4-19) po przekształceniu otrzymamy



Rys. 4-15. Schemat blokowy przenormowania wyników po jednym kroku metody Rungego-Kutty-Gilla z programowanym przecinkiem

$$\left.\begin{aligned} K_{i0} &= 2^{m+p_h+l_{i0}-p_i} HF_{i0}, \\ R_{i1} &= \tfrac{1}{2}(K_{i0} - Q_{i0}), \\ Y_{i1} &= Y_{i0} + 2^{-m} R_{i1}, \\ Q_{i1} &= Q_{i0} + 3R_{i1} - \tfrac{1}{2} K_{i0}, \end{aligned}\right\} \quad \text{dla } i = 1,2, \ldots, n, \qquad \text{(4-23a)}$$

$$\left.\begin{aligned} K_{i1} &= 2^{m+p_h+l_{i1}-p_i} HF_{i1}, \\ R_{i2} &= (1 - \sqrt{\tfrac{1}{2}})(K_{i1} - Q_{i1}), \\ Y_{i2} &= Y_{i1} + 2^{-m} R_{i2}, \\ Q_{i2} &= Q_{i1} + 3R_{i2} - (1 - \sqrt{\tfrac{1}{2}}) K_{i1}, \end{aligned}\right\} \quad \text{dla } i = 1,2, \ldots, n, \qquad \text{(4-23b)}$$

$$\left.\begin{aligned} K_{i2} &= 2^{m+p_h+l_{i2}-p_i} HF_{i2}, \\ R_{i3} &= \sqrt{\tfrac{1}{2}}(K_{i2} - Q_{i2}) + (K_{i2} - Q_{i2}), \\ Y_{i3} &= Y_{i2} + 2^{-m} R_{i3}, \\ Q_{i3} &= Q_{i2} + 3R_{i3} - \sqrt{\tfrac{1}{2}} K_{i2} - K_{i2}, \end{aligned}\right\} \quad \text{dla } i = 1,2, \ldots, n, \tag{4-23c}$$

$$\left.\begin{aligned} K_{i3} &= 2^{m+p_h+l_{i3}-p_i} HF_{i3}, \\ R_{i4} &= \tfrac{1}{3}(\tfrac{1}{2} K_{i3} - Q_{i3}), \\ Y_{i4} &= Y_{i3} + 2^{-m} R_{i4}, \\ Q_{i4} &= Q_{i3} + 3R_{i4} - \tfrac{1}{2} K_{i3}, \end{aligned}\right\} \quad \text{dla } i = 1,2, \ldots, n. \tag{4-23d}$$

Przy przejściu od s-tego kroku całkowania do $(s+1)$-ego sprawdzimy, czy y_{i4} obliczone w s-tym kroku spełniają nierówności postaci (4-13). W przypadku niespełnienia nierówności (4-13) dobieramy tak skale (czyli wyznaczamy nowe wartości p_i), aby przeskalowane wielkości spełniły już nierówności (4-13).

Wartości dla p_h, H, m oraz p_i^0 (początkowe wartości p_i) dobieramy tak, aby przy wykonywaniu jednego kroku całkowania nie nastąpiło przekroczenie zakresu, oraz po to, aby uniknąć normalizacji liczb przed rozpoczęciem obliczeń.

Uwaga: Jeśli $p_i = p_i^0$, to zamiast nierówności (4-13) wystarczy sprawdzić nierówność

$$|Y_{ij}| < R. \tag{4-24}$$

Powyższa uwaga nie daje zasadniczych korzyści praktycznych, ponieważ wymaga dodatkowej rozbudowy programu bez znaczniejszego wzrostu prędkości liczenia.

Przy przejściu od kroku s-tego do $(s+1)$-ego pamiętamy następujące $3n$ liczb, $Y_{14}, Y_{24}, \ldots, Y_{n4}, Q_{14}, Q_{24}, \ldots, Q_{n4}, p_1, p_2, \ldots, p_n$. Dla przechowywania wielkości Y_{i4}, Q_{i4} będziemy korzystali z komórek długich, dla przechowywania wielkości p_i będziemy korzystali z komórek krótkich.

Na rysunku 4-15 przedstawiony jest schemat blokowy czwartej części programu z programowanym przecinkiem dla metody Rungego-Kutta-Gilla.

4.5. ORGANIZACJA BIBLIOTEKI PODPROGRAMÓW

Przez bibliotekę podprogramów rozumiemy zbiór programów, z których każdy przeznaczony jest do wykonywania pewnych obliczeń bądź pewnych czynności organizacyjnych. Programy te muszą być zbudowane tak, aby dawały się łączyć w sposób możliwie prosty z programem głównym. Ponadto programy te muszą być zapisane w adresach względnych.

Na to aby w łatwy sposób korzystać z biblioteki podprogramów, wprowadzimy specjalną klasyfikację działową. Ponadto do każdego programu dołączymy kartę z charakterystycznymi danymi podprogramu.

Wyżej omówione karty umożliwiają posługiwanie się podprogramem bez szczegółowej jego znajomości.

Tablica 4-12

Karta klasyfikacyjna podprogramu

1. Symbol klasyfikacji działowej: . . . . . .
2. Numer inwentarzowy: . . . . . .
3. Zastosowanie: . . . . . .
. . . . . .
4. Postać liczb: arytmetyka . . . . . .
liczby . . . . . .
przecinek . . . . . .
5. Opis formuły numerycznej lub wykonywanych czynności: . . . . . .
. . . . . .
. . . . . .
6. Podprogram liniowy — cykliczny — iteracyjny(1)
modyfikowany (nazwa modyfikatora) — ustalony(1)
odnawiający się — nie odnawiający się(1)
7. Wywołanie podprogramu za pomocą rozkazu . . . . . .
. . . . . .
. . . . . .
8. Wykorzystane podprogramy: . . . . . .
. . . . . .
9. Długość programu . . . . . komórek krótkich
10. Liczba wykonywanych rozkazów: . . . . . .
11. Średni czas wykonywania: . . . . . .
. . . . . .
12. Rozmieszczenie danych wejściowych . . . . . .
. . . . . .
. . . . . .
13. Rozmieszczenie wyników . . . . . .
. . . . . .
. . . . . .
14. Komórki robocze: . . . . . .
. . . . . .
. . . . . .
15. Użyte parametry z pamięci stałej: . . . . . .
. . . . . .
16. Własne parametry logiczne: . . . . . .
. . . . . .
. . . . . .
17. Podprogram wykonał(a) . . . . . dn. . . . . .
18. Podprogram sprawdził(a) na maszynie . . . . . dn. . . . . .
19. Uwagi: . . . . . .
. . . . . .
. . . . . .

(1) Niepotrzebne skreślić.

Podamy jeszcze projekt klasyfikacji działowej podprogramów:

A. Wprowadzenie. Wyprowadzenie. Zmiana systemu pozycyjnego.

B. Kontrola pracy maszyny liczącej.

C. Demonstracja pracy maszyny liczącej.
D. Interpretacja kodów zewnętrznych. Kodowanie automatyczne.
F. Logika matematyczna.
G. Porządkowanie. Wybieranie. Zliczanie.
I. Specjalne działania arytmetyczne (zmiennoprzecinkowe, ze zwiększoną dokładnością).
J. Obliczenie wartości funkcji. Układanie tablic. Sumowanie szeregów.
K. Interpolacja. Ekstrapolacja. Aproksymacja.
M. Teoria liczb.
N. Algebra liniowa.
O. Równania algebraiczne nieliniowe, równania przestępne i układy.
Q. Przybliżone różniczkowanie. Przybliżone całkowanie.
R. Równania różnicowe.
S. Równania różniczkowe zwyczajne.
T. Równania różniczkowe cząstkowe.
U. Równania całkowe.
V. Inne zagadnienia analizy matematycznej.
X. Statystyka matematyczna.
W tablicy 4-12 przedstawiona jest karta klasyfikacyjna podprogramu.

5. ORGANIZACJA PRACY NA MASZYNIE TYPOWEJ

[5.1. Programy wprowadzające, 5.2. Programy wyprowadzające, 5.3. Uruchomienie maszyny, 5.4. Szukanie błędów w programach]

5.1. PROGRAMY WPROWADZAJĄCE

5.1.0. Programy i materiał liczbowy wprowadzamy do maszyny za pomocą specjalnych programów wprowadzających, które z symboli wydziurkowanych na taśmie perforowanej i odczytanych przez maszynę (tabl. 2-5) formują słowa (17 albo 34 bitowe) i umieszczają je pod odpowiednimi adresami w pamięci.

Mała maszyna cyfrowa nakłada dość trudne w praktyce do realizacji warunki na programy wprowadzające. Szczególnie ważne jest, aby program przeznaczony do wprowadzania rozkazów i pseudorozkazów (punkt 3.0) zapisanych w kodzie wewnętrznym w adresach względnych (punkt 3.1), dalej zwanym podstawowym programem wprowadzającym, działał stosunkowo szybko i był możliwie krótki.

Obecnie omówimy trzy programy wprowadzające, a mianowicie:

C. Demonstracja pracy maszyny liczącej.
D. Interpretacja kodów zewnętrznych. Kodowanie automatyczne.
F. Logika matematyczna.
G. Porządkowanie. Wybieranie. Zliczanie.
I. Specjalne działania arytmetyczne (zmiennoprzecinkowe, ze zwiększoną dokładnością).
J. Obliczenie wartości funkcji. Układanie tablic. Sumowanie szeregów.
K. Interpolacja. Ekstrapolacja. Aproksymacja.
M. Teoria liczb.
N. Algebra liniowa.
O. Równania algebraiczne nieliniowe, równania przestępne i układy.
Q. Przybliżone różniczkowanie. Przybliżone całkowanie.
R. Równania różnicowe.
S. Równania różniczkowe zwyczajne.
T. Równania różniczkowe cząstkowe.
U. Równania całkowe.
V. Inne zagadnienia analizy matematycznej.
X. Statystyka matematyczna.
W tablicy 4-12 przedstawiona jest karta klasyfikacyjna podprogramu.

5. ORGANIZACJA PRACY NA MASZYNIE TYPOWEJ

[5.1. Programy wprowadzające, 5.2. Programy wyprowadzające, 5.3. Uruchomienie maszyny, 5.4. Szukanie błędów w programach]

5.1. PROGRAMY WPROWADZAJĄCE

5.1.0. Programy i materiał liczbowy wprowadzamy do maszyny za pomocą specjalnych programów wprowadzających, które z symboli wydziurkowanych na taśmie perforowanej i odczytanych przez maszynę (tabl. 2-5) formują słowa (17 albo 34 bitowe) i umieszczają je pod odpowiednimi adresami w pamięci.

Mała maszyna cyfrowa nakłada dość trudne w praktyce do realizacji warunki na programy wprowadzające. Szczególnie ważne jest, aby program przeznaczony do wprowadzania rozkazów i pseudorozkazów (punkt 3.0) zapisanych w kodzie wewnętrznym w adresach względnych (punkt 3.1), dalej zwanym podstawowym programem wprowadzającym, działał stosunkowo szybko i był możliwie krótki.

Obecnie omówimy trzy programy wprowadzające, a mianowicie:

1) podstawowy program wprowadzający, przeznaczony do wprowadzania słów krótkich (rozkazów lub pseudorozkazów) i umieszczania tych słów krótkich w odpowiednich miejscach w pamięci;

2) stałoprzecinkowy program wprowadzająco-przeliczający dla dziesięciocyfrowych liczb dziesiętnych (ostatnia cyfra parzysta) co do modułu mniejszych od jedności; program ten przelicza liczby z systemu dziesiętnego na binarny i umieszcza wyniki w odpowiednich miejscach pamięci;

3) zmiennoprzecinkowy program wprowadzająco-przeliczający dla ośmiocyfrowych zmiennoprzecinkowych liczb dziesiętnych; program ten przelicza liczby ze zmiennoprzecinkowego systemu dziesiętnego na zmiennoprzecinkowy system binarny i umieszcza wynik w odpowiednich miejscach pamięci.

Podstawowy program wprowadzający znajduje się na stałe w maszynie (jest on nagrany na ścieżce nr zero), pozostałe programy wprowadzające w zależności od potrzeb wprowadzamy do maszyny (oczywiście za pomocą podstawowego programu wprowadzającego).

Przy korzystaniu z każdego z wyżej wymienionych programów wprowadzających nastawiamy warunek wejścia na pierwszej pozycji wejścia, za pomocą przełącznika Y0 (Załącznik 2).

5.1.1. Podstawowy program wprowadzający. Jak już wspominaliśmy, program ten służy do wprowadzania słów krótkich (rozkazów lub pseudorozkazów). Program ten rozróżnia słowa dwóch długości: słowa, które nie podlegają przeadresowaniu przy wprowadzaniu, kodowane w sześciu rządkach taśmy perforowanej i słowa, które podlegają przeadresowaniu w trakcie wprowadzania, kodowane w siedmiu rządkach taśmy perforowanej. Podstawowy program wprowadzający dopuszcza korzystanie w jednej wprowadzonej sekwencji rozkazów i pseudorozkazów z co najwyżej dwunastu modyfikatorów (stałych przeadresowania), oznaczonych odpowiednio literami X, N, H, L, M, S, G, B, D, F, J, K. Jako zasadę przy posługiwaniu się programem przyjęto używać w przypadku wprowadzania podprogramów, zawierających stałą przeadresowania, modyfikatora K.

Podstawowy program wprowadzający korzysta z 14 miejsc roboczych na ścieżce nr 1.

Do sterowania podstawowym programem wprowadzającym służą specjalne rozkazy, tzw. rozkazy taśmowe, wprowadzone do maszyny przez ten program, umieszczone pod adresem 0101, a następnie wykonane przez podstawowy program wprowadzający. W tablicy 5-1 są podane wszystkie rozkazy taśmowe, jakich używamy przy wprowadzaniu programu do maszyny.

Perforowanie taśmy. Przywykliśmy do pisania liczb począwszy od cyfry najbardziej znaczącej, a skończywszy na cyfrze najmniej znaczącej. Okazało się, że znaczne uproszczenia podstawowego programu wprowadzającego można uzyskać wprowadzając liczby w kierunku przeciwnym do tego, w jakim przywykliśmy pisać liczby. Tę pozorną trudność możemy rozwiązać w sposób bardzo prosty, a mianowicie dziurkujemy kolejne cyfry liczby na taśmie w kolejności od najbardziej znaczącej do najmniej znaczącej, wprowadzamy zaś taśmę w kierunku przeciwnym do dziurkowania. Musimy przy tym pamiętać, że zaczynamy wówczas wprowadzanie od ostatniego roz-

Tablica 5-1

Rozkazy taśmowe

Lp.	Kod	Czynności wykonywane
1	M1 *n*	przeczytanie jednego słowa, a następnie przekazanie sterowania rozkazowi, którego adres jest zapisany pod adresem *n*
2	M2 *n*	przeczytanie jednego słowa, a następnie przekazanie sterowania rozkazowi zapisanemu pod adresem *n*
3	M5 *n*	przeczytanie jednego słowa, a następnie „stop" z pobraniem do rejestru dyspozytora rozkazu zapisanego pod adresem *n*
4	S4 *n*	przesłanie następnie odczytanego słowa krótkiego przez program wprowadzający pod adres *n*
5	G4 *n*	przeczytanie jednego słowa, a następnie przewinięcie taśmy perforowanej o jeden rządek

Tablica 5-2

Przykład rozkazów taśmowych końca czytania

Kod rozkazu	Kolejność symboli dziurkowanych	Kod zewnętrzny symboli dziurkowanych
	0	10 000
	0	10 000
	0	10 000
00 0000	0	10 000
	0	10 000
	0	10 000
	M	01 000
	2	10 010
M2 1777	1	10 001
	7	10 111
	7	10 111
	7	10 111

kazu programu, a kończymy wprowadzanie wprowadzając pierwszy rozkaz programu. Dziurkując taśmę korzystamy z programów zapisanych na formularzach EM (rys. 3-1), dziurkujemy rozkaz po rozkazie w kolejności wzrastających adresów.

Przystępując do dziurkowania, dziurkujemy najpierw rozkazy dotyczące czynności maszyny, po wprowadzeniu programu. W tablicy 5-2 podany jest przykład rozkazów przekazujących sterowanie po zakończeniu czytania rozkazowi stojącemu pod adresem 1777'. Następnie dziurkujemy kolejne rozkazy (albo pseudorozkazy) w kolejności ustalonej na arkuszu programowym EM. Jeśli w części adresowej występuje stała przeadresowania, np. K, to dziurkujemy ją również za pomocą odpowiedniego klawisza dziurkarki. Pomiędzy kolejnymi ósemkami słów krótkich dziurkujemy odstępy wejścia, które ułatwią nam kontrolę taśmy i ewentualne poprawki na taśmie. Na końcu dziurkujemy grupę rozkazów taśmowych początku czytania i wartości stałych przeadresowania dla wydziurkowanego programu. W tablicy 5-3 podaliśmy przykład rozkazów

taśmowych dotyczących początku czytania, przy założeniu, że ostatni rozkaz podprogramu umieszczamy pod adresem 0533′, a stała przeadresowania K (modyfikator) ma wartość 0237′.

Korekcja taśmy. Jeśli jakiś rozkaz w podprogramie został źle wydziurkowany, ale przy tym żadna cyfra nie została opuszczona ani też nie zostało wydziurkowane więcej cyfr, to między końcem czytania a pierwszym rozkazem podprogramu umieszczamy grupę rozkazów: S4-adres błędnie wydziurkowanego rozkazu, a następnie właściwą postać tego rozkazu. Aby uniknąć zbędnego klejenia taśmy, należy zostawić duży odstęp między rozkazami końca czytania a pierwszym rozkazem programu i w tym miejscu wydziurkujemy dodatkową grupę rozkazów korekcji taśmy.

Tablica 5-3

Przykład rozkazów taśmowych początku czytania

Kod rozkazu	Kolejność symboli dziurkowanych	Kod zewnętrzny symboli dziurkowanych
	S	01 001
	4	10 100
S4 0533	0	10 000
	5	10 101
	3	10 011
	3	10 011
	0	10 000
	0	10 000
00 0237	0	10 000
(modyfikator	2	10 010
K)	3	10 011
	7	10 111
	S	01 001
	4	10 100
S4 0116	0	10 000
	1	10 001
	1	10 001
	6	10 110

Ponieważ program wprowadzamy w kierunku przeciwnym do kierunku dziurkowania, więc kolejność czytania grup rozkazów początku czytania i końca czytania jest taka jak w tablicach 5-4 i 5-5.

Jak już wspominaliśmy, podstawowy program wprowadzający dysponuje 12 modyfikatorami, których wartości zapisujemy odpowiednio w komórkach o adresach: X = = (0102), N = (0104), H = (0105), L = (0106), M = (0107), S = (0110), G = (0111), B = (0112), D = (0113), F = (1014), J = (0115) i K = (0116). Pozostałe dwie komórki robocze 0100 i 0101 będziemy oznaczali odpowiednio literami r i t. Komórkę 0100 będziemy nazywali relatywizatorem programu wprowadzającego, a komórkę

Tablica 5-4

Przykład początku czytania

Taśma	Kod zewnętrzny	Symbole na klawiaturze dziurkarki	
● • ● ●	10 110	6	
● • ●	10 001	1	
● • ●	10 001	1	S4 0116
● •	10 000	0	
● • ●	10 100	4	
• ● • ●	01 001	S	
● • ● ● ●	10 111	7	
● • ● ●	10 011	3	
● • ●	10 010	2	00 0237
● •	10 000	0	modyfikator K
● •	10 000	0	
● •	10 000	0	
● • ● ●	10 011	3	
● • ● ●	10 011	3	
● • ● ●	10 101	5	S4 0533
● •	10 000	0	
● • ●	10 100	4	
• ● • ●	01 001	S	

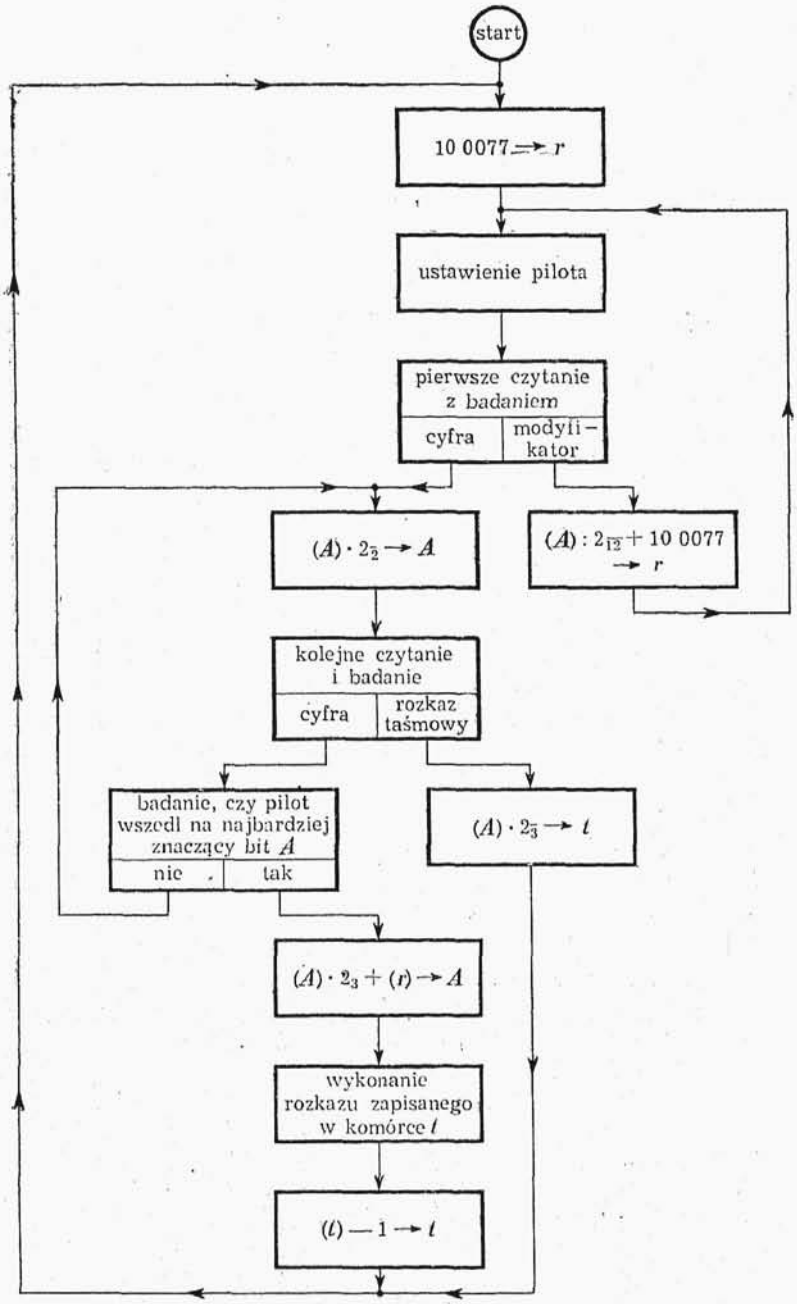
Tablica 5-5

Przykład końca czytania

Taśma	Kod zewnętrzny	Symbole na klawiaturze dziurkarki	
● • ● ● ●	10 111	7	
● • ● ● ●	10 111	7	
● • ● ● ●	10 111	7	M2 1777
● • ●	10 001	1	
● • ●	10 010	2	
• ● •	01 000	M	
● •	10 000	0	
● •	10 000	0	
● •	10 000	0	00 0000
● •	10 000	0	
● •	10 000	0	
● •	10 000	0	

0101, w której umieszczony jest rozkaz taśmowy, będziemy nazywali transferatorem programu wprowadzającego.

Na rysunku 5-1 podany jest schemat blokowy podstawowego programu wprowadzającego, w tabl. 5-6 zaś podane są kolejne rozkazy tego programu. Podstawowy program wprowadzający jest programem trójwariantowym, pierwszy wariant — czytanie rozkazu (pseudorozkazu) niezmodyfikowanego, drugi wariant — czytanie rozkazu (pseudorozkazu) modyfikowanego, trzeci wariant — czytania rozkazu taśmowego.



Rys. 5-1. Schemat blokowy podstawowego programu wprowadzającego

Tablica 5-6

Podstawowy program wprowadzający

Kolejny adres	Kolejny rozkaz		Objaśnienia
0000	12	0027	przesłanie rozkazu 10 0077 do relatywizatora
1	14	0100	
2	12	0076(1)	przesłanie i ustawienie w akumulatorze pilota
3	22	0040	
4	24	2424	czytanie jednego rządka taśmy
5	03	0022	skok w przypadku odczytania modyfikatora
6	23	0002	przesunięcie zawartości akumulatora o dwie pozycje w lewo
7	24	0000	kolejne czytanie jednego rządku taśmy
0010	03	0025	skok w przypadku odczytania rozkazu taśmowego
1	23	0000	sprawdzenie położenia pilota, skok, gdy pilot nie znajduje
2	06	0006	się na bicie α_0
3	23	0003	ustawienie w akumulatorze odczytanego słowa krótkiego
4	00	0100	wykonanie rozkazu zapisanego w relatywizatorze
5	00	0101	wykonanie rozkazu zapisanego w transformatorze
6	12	0101	
7	11	0076(1)	przeadresowanie zawartości transformatora
0020	14	0101	
1	02	0000	skok do początku programu w przypadku słowa krótkiego
2	22	0014	modyfikowanego, sformowanie właściwej postaci modyfika-
3	10	0027	tora
4	02	0001	skok do rozkazu 0001
5	23	0003	ustawienie w akumulatorze odczytanego rozkazu taśmowego
6	02	0020	skok do rozkazu 0020
7	10	0077	parametr

(1) Pod adresem 0076 zapisany jest parametr 2^{-16}, w postaci pseudorozkazu 00 0001.

Tablica 5-7

Wariant I: Kolejne zawartości licznika rozkazów przesyłane do rejestru dyspozytora przy czytaniu rozkazu (pseudorozkazu) nie modyfikowanego

0000	0006	0012	0011
0001	0007		0012
0002	0010	0006	
0003	0011	0007	0013
0004	0012	0010	0014
0005		0011	0015
0006		0012	0016
0007	0006		0017
0010	0007	0006	0020
0011	0010	0007	0021
0012	0011	0010	

Dla łatwiejszego zrozumienia działania podstawowego programu wprowadzającego podaliśmy w tabl. 5-7, 5-8 i 5-9 kolejne zawartości licznika rozkazów przesyłane do rejestru dyspozytora, dla wariantów pierwszego, drugiego i trzeciego.

Tablica 5-8

Wariant II: Kolejne zawartości licznika rozkazów przesyłane do rejestru dyspozytora przy czytaniu rozkazu (pseudorozkazu) modyfikowanego

0000	0002	0006	0012	0010
0001	0003	0007		0011
0002	0004	0010	0006	0012
0003	0005	0011	0007	0013
0004	0006	0012	0010	0014
0005	0007		0011	0015
0022	0010	0006	0012	0016
0023	0011	0007		0017
0024	0012	0010	0006	0020
0001		0011	0007	0021

Tablica 5-9

Wariant III: Kolejne zawartości licznika rozkazów przesyłane do rejestru dyspozytora przy czytaniu rozkazu taśmowego

0000	0012	0010	0006
0001		0011	0007
0002	0006	0012	0010
0003	0007		0025
0004	0010	0006	0026
0005	0011	0007	0020
0006	0012	0010	0021
0007		0011	
0010	0006	0012	
0011	0007		

5.1.2. Stałoprzecinkowy program wprowadzająco-przeliczający. W odróżnieniu od podstawowego programu wprowadzającego, program wprowadzająco-przeliczający wprowadza do maszyny pojedyncze liczby, dlatego też wprowadzanie liczby za pomocą tego podprogramu jest wolniejsze od wprowadzania liczb zakodowanych w postaci par pseudorozkazów za pomocą podstawowego programu wprowadzającego. Dla zapisania jednej liczby dziesiętnej dziesięciocyfrowej (ostatnia cyfra parzysta), co do modułu mniejszej od jedności, potrzebujemy aż 16 rządków taśmy perforowanej z podziałem na następujące grupy:

1 rządek + 4 rządki + 1 rządek + 10 rządków.

Pierwszy rządek przeznaczony jest na zapis symbolu końca przeliczenia, w tym celu perforujemy na tej pozycji przed pierwszą liczbą cyfrę 0, a przed następnymi liczbami

literę D. Dalsze cztery rządki przeznaczone są na adres (zapisany w kodzie ósemkowym), pod który ma być przesłana odczytana liczba. Następnie jeden rządek przeznaczony jest na znak liczby przeliczonej. Pozostałe 10 rządków służy do zapisania kolejnych dziesięciu cyfr liczby przeliczonej. Liczby dziesiątkowe przeznaczone do perforowania zapisujemy na specjalnych arkuszach programowych D.

Oznaczmy kolejne cyfry rozwinięcia l w postaci dziesiętnej przez d_i; możemy liczbę l zapisać w postaci:

$$l = \sum_{i=1}^{10} d_i 10^{-i}. \tag{5-1}$$

Dla przeliczenia wyrażeniu (5-1) nadamy postać Hornera:

$$(\ldots(d_{10} \cdot 10^{-1} + d_9) \cdot 10^{-1} + d_8)10^{-1} + \ldots + d_1)10^{-1}. \tag{5-2}$$

Stałoprzecinkowy program wprowadzająco-przeliczający, podobnie jak podstawowy program wprowadzający, wprowadza liczby w kierunku od najmniej znaczącej cyfry do najbardziej znaczącej cyfry (w kierunku przeciwnym do dziurkowania). Liczba 10^{-1} jest liczbą dwójkowo niewymierną i dlatego bezpośrednie korzystanie z formuły (5-2) prowadziłoby do dużych błędów. Dla uniknięcia tej trudności, zamiast mnożyć kolejne cyfry rozwinięcia d_i przez 10^{-1}, będziemy je dzielić przez 10, dzielenie wykonujemy z zaokrągleniem według reszty.

Poświęćmy trochę miejsca dla omówienia metody zaokrąglenia przy dzieleniu. Jeśli dowolną liczbę dodatnią w rozwinięciu dziesiętnym k-cyfrową mamy podzielić przez jakąś inną liczbę dodatnią w rozwinięciu dziesiętnym i chcemy w wyniku otrzymać najlepsze k-cyfrowe przybliżenie wyniku, to za najmniej znaczącą cyfrę dzielnej zapisujemy połowę dzielnika i w ten sposób otrzymaną liczbę dzielimy przez nasz dzielnik. Omówimy jeszcze powyższą metodę na przykładach.

Przykład 5-1. Niech $k = 2$, dzielna będzie równa 2,2, zaś dzielnik 3. Wykonujemy dzielenie

```
0,733...
2,2        :3
21
 10
  9
 10
  :
```

Najlepszym dwucyfrowym przybliżeniem wyniku jest liczba 0,73. Jeśli teraz zastosujemy metodę dzielenia z zaokrągleniem, to zamiast dzielić liczbę 2,2 będziemy dzielić liczbę 2,215 przerywając dzielenie po otrzymaniu drugiej cyfry wyniku:

```
0,73
2,215    :3
21
 11
  9
```

Przykład 5-2. Niech $k = 2$, dzielna będzie równa 1,7, dzielnik zaś będzie równy 2,4. Wykonujemy dzielenie:

```
0,708...
1,7     :2,4
168
 200
 192
   8
   :
```

Najlepszym dwucyfrowym wynikiem będzie liczba 0,71. Jeśli teraz zastosujemy metodę dzielenia z zaokrągleniem, to zamiast dzielić liczbę 1,7 będziemy dzielić liczbę 1,712 przerywając dzielenie po otrzymaniu drugiej cyfry wyniku:

```
0,71
1,712   :2,4
168
 32
 24
```

Jak widać z powyższych przykładów, metoda dzielenia z zaokrągleniem pozwala na podanie k-cyfrowego przybliżenia wyniku z zaokrągleniem bez obliczenia $k + 1$ cyfry wyniku. Uzasadnienie takiego postępowania jest niesłychanie proste; przeprowadzimy je dla przypadku dzielenia dwu liczb dodatnich mniejszych od jedności l_1, l_2. Niech obie te liczby mają k-cyfrowe rozwinięcia przy podstawie g, wówczas możemy je przedstawić w postaci.

$$l_1 = \sum_{i=1}^{k} a_i g^{-i}, \qquad l_2 = \sum_{i=1}^{k} b_i g^{-i}. \tag{5-3a}$$

Jeśli chcemy otrzymać k-cyfrowe przybliżenie wyniku dzielenia liczb l_1 i l_2, musimy do $k + 1$ cyfry wyniku dodać $\frac{1}{2}g$

$$\frac{\sum_{i=1}^{k} a_i g^{-i}}{\sum_{i=1}^{k} b_i g^{-i}} + \frac{g}{2} \cdot g^{-k-1}. \tag{5-3b}$$

Poniższe wyrażenie równoznaczne jest z wyrażeniem (5-3b)

$$\frac{\sum_{i=1}^{k} a_i g^{-i}}{\sum_{i=1}^{k} b_i g^{-i}} + \frac{1}{2} g^{-k}. \tag{5-3c}$$

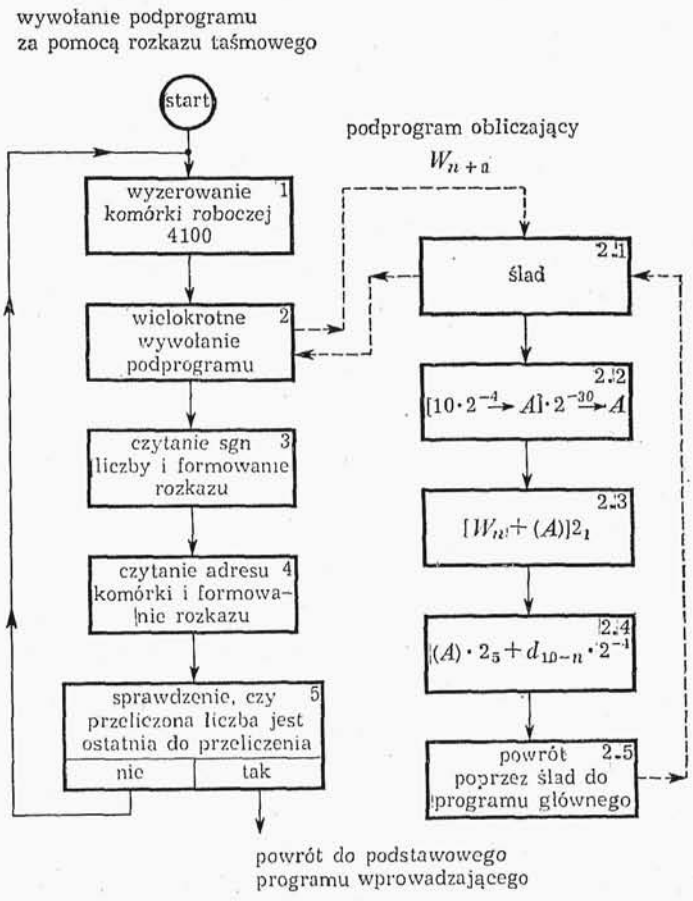
Po sprowadzeniu wyrażenia (5-3c) do wspólnego mianownika otrzymamy zgodnie

Tablica 5-10

Stałoprzecinkowy program wprowadzająco-przeliczający

Operator lub predykat	Kolejny adres	Kolejny rozkaz		Objaśnienia
1	$k+0000$	12	0077	położenie $W_0 = 0$
	1	14	4100	
2	2	04	$k+0036$	pierwsze wywołanie podprogramu oblicz. W_{n+1}
	3	04	$k+0036$	drugie wywołanie, jak wyżej
	4	04	$k+0036$	trzecie wywołanie, jak wyżej
	5	04	$k+0036$	czwarte wywołanie, jak wyżej
	6	04	$k+0036$	piąte wywołanie, jak wyżej
	7	04	$k+0036$	szóste wywołanie, jak wyżej
	$k+0010$	04	$k+0036$	siódme wywołanie, jak wyżej
	1	04	$k+0036$	ósme wywołanie, jak wyżej
	2	04	$k+0036$	dziewiąte wywołanie, jak wyżej
	3	04	$k+0036$	dziesiąte wywołanie, jak wyżej
3	4	12	$k+0047$	sformowanie znaku liczby przeliczonej
	5	24	0000	poprzez sformowanie rozkazu przesyła-
	6	14	$k+0031$	nia
4	7	24	0000	
	$k+0020$	23	0002	
	1	24	0000	
	2	23	0002	czytanie adresu, pod który mamy przesłać
	3	24	0000	wynik
	4	23	0002	
	5	24	0000	
	6	22	0003	
	7	11	0007	sformowanie rozkazu przesyłającego wy-
	$k+0030$	14	$k+0032$	nik przeliczenia pod zadany adres
	1	77	4100	miejsce na rozkaz przesyłania formujący znak liczby
	2	14	7777	miejsce na rozkaz przesyłania wyniku pod zadany adres
5	3	24	0000	sprawdzenie, czy przeliczona liczba jest ostatnią z liczb przeznaczonych do prze-
	4	03	$k+0000$	liczenia, a jeśli tak, to powrót do podsta-
	5	02	0000	wowego programu wprowadzającego
2.1	6	77	7777	miejsce na łącznik
2.2	7	12	0065	
	$k+0040$	22	0036	formowanie wyrażenia
2.3	1	10	4100	$W_n \cdot 2^{-4} + d_{10-n} \cdot 2^{-4} + 10 \cdot 2^{-38}$
	2	23	0001	
2.4	3	24	0000	
2.5	4	17	0065	obliczenie W_{n+1}
	5	14	4100	$W_{n+1} \rightarrow W_n$
2.6	6	01	$k+0036$	powrót do programu głównego parametr
	7	20	4000	

Podprogram realizujący algorytm $W_{n+1} = (W_n \cdot 2^{-4} + d_{10-n} \cdot 2^{-4} + 10 \cdot 2^{-38}) : (2^{-4} \cdot 10)$



Rys. 5-2. Schemat blokowy stałoprzecinkowego programu wprowadzająco-przeliczającego

z poprzednio podaną metodą dzielenia z zaokrągleniem:

$$\frac{\sum_{i=1}^{k} a_i g^{-i} + \frac{1}{2}\sum_{i=1}^{k} b_i g^{-k-i}}{\sum_{i=1}^{k} b_i g^{-i}}. \tag{5-3d}$$

Wzór (5-2) możemy przedstawić jako następującą formułę rekurencyjną:

$$W_0 = 0, \quad W_{n+1} = (W_n^* + d_{10-n}) : 10, \tag{5-4a}$$

gdzie $n = 1,2,\ldots,9$.

Ponieważ w maszynie nie dysponujemy stałoprzecinkową liczbą 10, tylko liczbą $2^{-4} \cdot 10$, musimy jeszcze pomnożyć licznik i mianownik wzoru (5-4a) przez 2^{-4} oraz powiększyć licznik o połowę mianownika pomnożonego przez 2^{-33} (zaokrąglenie dzielenia), ostatecznie otrzymamy wzór w postaci:

$$W_0 = 0, \quad W_{n+1} = (W_n \cdot 2^{-4} + d_{10-n} \cdot 2^{-4} + 10 \cdot 2^{-38}) : 2^{-4} \cdot 10, \tag{5-4b}$$

gdzie $n = 1,2,\ldots,9$.

Musimy jeszcze oszacować błąd maksymalny (definicja błędu maksymalnego podana jest przez J. Łukaszewicza i M. Warmusa — Bibliografia [12]), jaki popełniamy przeliczając liczby stałoprzecinkowe dziesiętne na liczby stałoprzecinkowe binarne. Na powstanie błędu przy obliczeniu W_{n+1} składają się dwa czynniki: błąd, z jakim obliczyliśmy W_n, oraz błąd zaokrąglenia przy obliczeniu W_{n+1}. Odejmując stronami od wyrażenia obarczonego błędami wyrażenia dokładne otrzymamy następujący związek między błędem maksymalnym w $(n+1)$-kroku Δ_{n+1} a błędem maksymalnym w n-kroku Δ_n i błędem zaokrąglenia $\pm 2^{-34}$, przy założeniu, że $\Delta_0 = 0$:

$$\Delta_{n+1} = \frac{1}{10}\Delta_n + 2^{-34}. \tag{5-5a}$$

Korzystając z wzoru (5-5a) możemy z łatwością oszacować błąd maksymalny Δ wyniku przeliczenia przy użyciu wzoru (5-4b):

$$\Delta = \Delta_{10} < \frac{10}{9} \cdot 2^{-34}. \tag{5-5b}$$

Jak widać z powyższego oszacowania, błąd przeliczenia jest niewielki i przeliczenie liczb dziesięciocyfrowych z ostatnią cyfrą parzystą (tzw. dziewięć i pół cyfr znaczących) w systemie dziesiętnym na system binarny jest uzasadnione, ponieważ nie prowadzi do większych strat dokładności.

Schemat blokowy stałoprzecinkowego programu przeliczająco-wprowadzającego jest przedstawiony na rys. 5-2. W tablicy 5-10 podane są kolejne rozkazy programu.

5.1.3. Zmiennoprzecinkowy program wprowadzająco-przeliczający. Podobnie jak stałoprzecinkowy program wprowadzająco-przeliczający, program ten działa wolno, ponadto wymaga on programu działań zmiennego przecinka, dokładniej podprogramu sformowania liczby zmiennoprzecinkowej.

Zmiennoprzecinkowy program wprowadzająco-przeliczający służy do wprowadzania do maszyny liczby postaci:

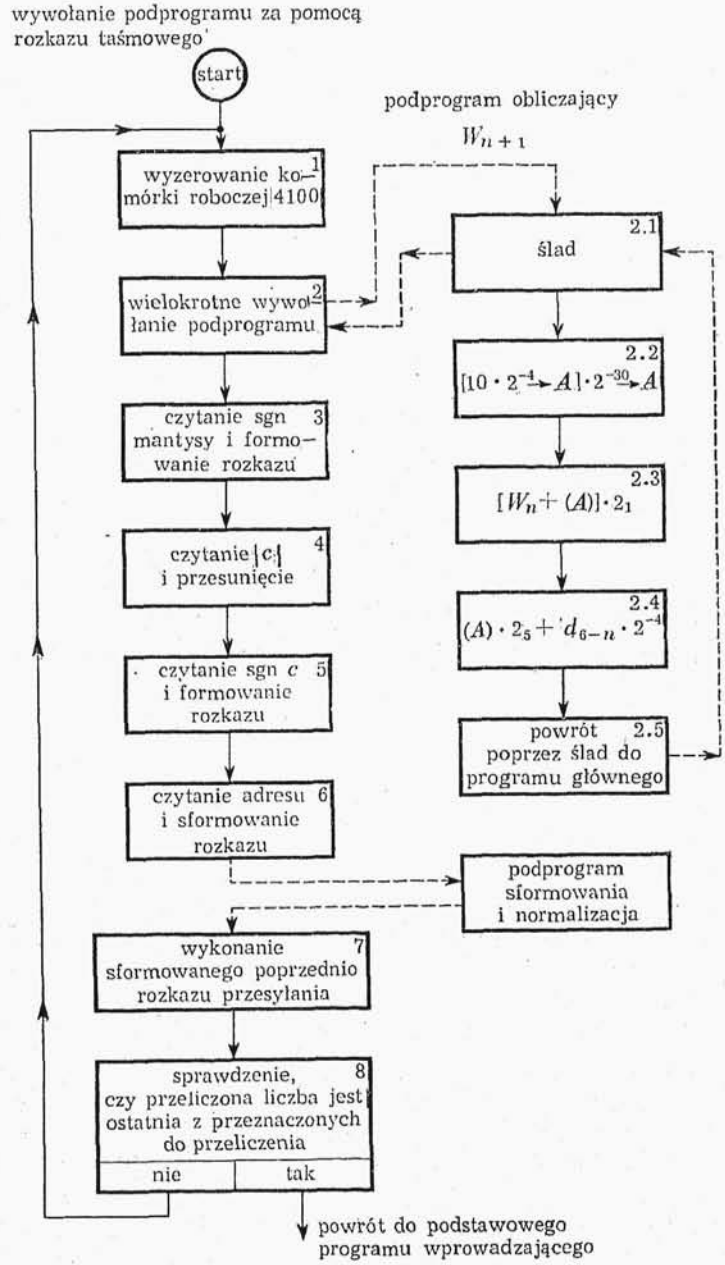
$$10^c \cdot m, \tag{5-6}$$

gdzie $-9 \leqslant c \leqslant 9$, mantysa zaś jest liczbą dziesiętną ośmiocyfrową co do modułu mniejszą od jedności. Dla zapisania jednej liczby postaci (5-6) potrzebujemy 16 rządków na taśmie perforowanej z podziałem na następujące grupy:

1 rządek + 4 rządki + 1 rządek + 1 rządek + 1 rządek + 8 rządków.

Podobnie jak w stałoprzecinkowym programie wprowadzająco-przeliczającym pierwszy rządek przeznaczony jest na zapis symbolu końca przeliczania, dalsze cztery rządki przeznaczone są na adres (zapisany w kodzie ósemkowym), pod który ma być przesłana odczytana liczba. Następnie jeden rządek przeznaczony jest na znak cechy i jeden na moduł cechy, dalej jeden znak przeznaczony jest na znak mantysy i pozostałe osiem rządków na moduł mantysy.

Algorytm przeliczenia mantysy, z którego korzysta zmiennoprzecinkowy program wprowadzający, jest analogiczny od algorytmu przedstawionego w punkcie 5.1.2. Cecha dziesiętna liczby postaci (5-6) jest przeliczana metodą słownika; szczegółową budowę takiego słownika omówimy w punkcie 6.1. Schemat blokowy programu jest



Rys. 5-3. Schemat blokowy zmiennoprzecinkowego programu wprowadzająco-przeliczającego

przedstawiony na rys. 5-3. Zmiennoprzecinkowy program wprowadzająco-przeliczający jest wywoływany rozkazami taśmowymi, a po zakończeniu przeliczenia sekwencji liczb postaci (5-6), przekazuje sterowanie podstawowemu programowi wprowadzającemu.

5.1.4. Program ustawiający standartowe podprogramy. Duże ułatwienie przy składaniu i przeadresowywaniu programów daje umieszczenie podprogramów funkcji elementarnych w stałych miejscach pamięci. Wadą tego rodzaju rozwiązania jest nieekonomiczne wykorzystanie pamięci, wynikające z tego, że nie zawsze wszystkie podprogramy funkcji elementarnych są wykorzystane przy obliczeniach. Pewnym kompromisem jest wywołanie podprogramów przez tzw. słownik, w którym adresy wywoływania poszczególnych podprogramów są ustalone raz na zawsze. Zakładając, że przy obliczeniach dysponujemy co najwyżej 26 podprogramami, możemy przyjąć, że słownik mieści się na ścieżce nr 37. Na każdy podprogram w słowniku wykorzystujemy dwa słowa krótkie.

Dla podprogramu nr 0 (np. $\sqrt{x}$) wykorzystujemy adresy 3714, 3715, dla podprogramu nr 1 (np $\sqrt[3]{x}$) wykorzystujemy adresy 3716, 3717 itd. Dla podprogramu nr 25 wykorzystujemy adresy 3776 i 3777. Pod pierwszym z adresów przeznaczonych w słowniku na wywołanie podprogramu zostawimy miejsce na ślad, pod drugim rozkaz 02 do pierwszego rozkazu podprogramu. Powrót po wykonaniu podprogramu odbywa się przez ślad zostawiony przez program główny w słowniku.

Przy założeniu, że programy składają się z parzystej ilości słów krótkich oraz że pierwszy rozkaz podprogramu ma adres parzysty, można za pomocą prostego programu ustawić podprogramy w pamięci jeden za drugim, tak że wszystkie miejsca pamięci są wykorzystane. Dla wywołania programu „ustawiającego" umieszczamy na końcu podprogramu (złożonego wyłącznie z rozkazów i pseudorozkazów) grupę rozkazów taśmowych:

1) 00 Numer podprogramu (od 0000' do 0030'),
2) M2 0127,
3) 00 Ilość słów krótkich (od 0000' do 3777'),
4) M2 0117.

Tablica 5-11

Program ustawiający podprogramy

	00	0000						
	M5	0117						
0117	14	0103	0130	14	0132	0140	02	0000
0120	12	0144	1	12	0103	1	11	0144
1	11	0103	2	14	7777	2	03	0000
2	10	0147	3	12	0144	3	05	0144
3	14	0116	4	14	0101	4	14	3713
4	10	0140	5	12	0103	5	11	7777
5	14	0103	6	10	0145	6	14	3777
6	02	0000	7	14	0144	7	24	0001
7	10	0146					S4	0147

Podprogramy zamknięte wprowadzamy kolejno jeden za drugim (kolejność dowolna).

Kolejne rozkazy programu ustawiającego, łącznie z rozkazami taśmowymi podane są w tabl. 5-11.

Ponadto program ustawiający umożliwia programiście autokontrolę, czy przypadkiem rozkazy programu głównego nie są wprowadzane na miejsca, na które poprzednio wprowadziliśmy podprogramy. W tym celu na końcu programu głównego należy umieścić grupę rozkazów taśmowych:

1) S4 g,
2) 14 $g-1$,
3) M2 0141,

gdzie g oznacza najwyższy adres zajęty przez program główny. W przypadku pomyłki w podziale pamięci program główny nie zostanie wprowadzony do maszyny.

5.2. PROGRAMY WYPROWADZAJĄCE

Wyprowadzanie wyników obliczeń oraz programów zapisanych w pamięci dokonujemy za pomocą programów wyprowadzających. Przy normalnej eksploatacji maszyny musimy dysponować co najmniej czterema programami wyprowadzającymi:

1) programem do wyprowadzania zawartości kolejnych miejsc pamięci w postaci rozkazów i pseudorozkazów,
2) programem wyprowadzająco-przeliczającym dla liczb całkowitych,
3) stałoprzecinkowym programem wyprowadzająco-przeliczającym,
4) zmiennoprzecinkowym programem wyprowadzająco-przeliczającym.

W odróżnieniu od programów wprowadzających, programy wyprowadzające mają prostą strukturę. Ograniczymy się do omówienia tylko dwóch programów wyprowadzających, a mianowicie stałoprzecinkowego programu wyprowadzająco-przeliczającego (bardzo uproszczonego) i programu dla wyprowadzania zawartości kolejnych miejsc pamięci w postaci rozkazów i pseudorozkazów.

Stałoprzecinkowy program wyprowadzająco-przeliczający łącznie z podstawowym programem wprowadzającym i parametrami jest nagrany na stałe na ścieżce nr zero. Program ten służy do drukowania liczb stałoprzecinkowych co do modułu mniejszych od jedności w układzie dziesiętnym, ponadto służy do drukowania odstępów potrójnych i napisów trzyznakowych.

Stałoprzecinkowy program wyprowadzająco-przeliczający zastosowany do liczby x ($|x|<1$), drukuje symbol $+0$, lub -0, w zależności od znaku liczby x oraz 12 cyfr dziesiętnych liczby x. Cyfry te uzyskuje się przez wielokrotne mnożenie x przez $2^{-4} \cdot 10$, drukowanie cyfry, która znalazła się na miejscu charakterystycznym akumulatora oraz arytmetyczne przesuwanie wyniku o 4 miejsca w lewo (czyli mnożenie wyniku przez 2^4). Po wydrukowaniu liczby program drukuje potrójny odstęp. Odstęp taki można również drukować bez drukowania jakiejkolwiek liczby.

Uwaga: Napis -0, tworzymy przez dodanie do napisu $+0$, liczby ósemkowej

Podprogramy zamknięte wprowadzamy kolejno jeden za drugim (kolejność dowolna).

Kolejne rozkazy programu ustawiającego, łącznie z rozkazami taśmowymi podane są w tabl. 5-11.

Ponadto program ustawiający umożliwia programiście autokontrolę, czy przypadkiem rozkazy programu głównego nie są wprowadzane na miejsca, na które poprzednio wprowadziliśmy podprogramy. W tym celu na końcu programu głównego należy umieścić grupę rozkazów taśmowych:

1) S4 g,
2) 14 $g-1$,
3) M2 0141,

gdzie g oznacza najwyższy adres zajęty przez program główny. W przypadku pomyłki w podziale pamięci program główny nie zostanie wprowadzony do maszyny.

5.2. PROGRAMY WYPROWADZAJĄCE

Wyprowadzanie wyników obliczeń oraz programów zapisanych w pamięci dokonujemy za pomocą programów wyprowadzających. Przy normalnej eksploatacji maszyny musimy dysponować co najmniej czterema programami wyprowadzającymi:

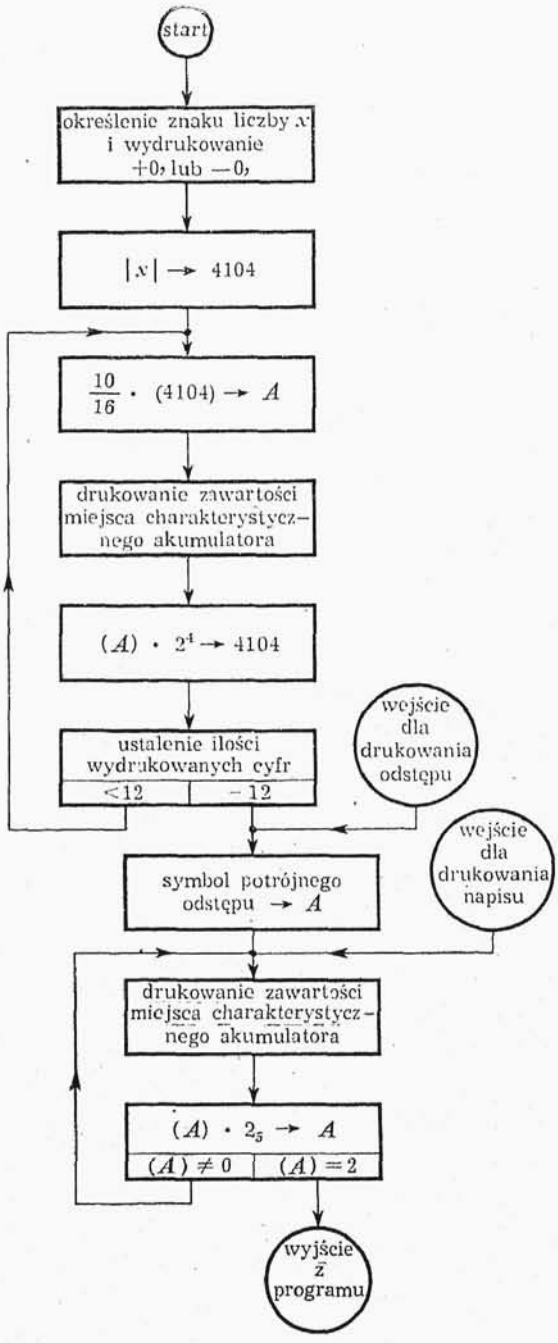
1) programem do wyprowadzania zawartości kolejnych miejsc pamięci w postaci rozkazów i pseudorozkazów,
2) programem wyprowadzająco-przeliczającym dla liczb całkowitych,
3) stałoprzecinkowym programem wyprowadzająco-przeliczającym,
4) zmiennoprzecinkowym programem wyprowadzająco-przeliczającym.

W odróżnieniu od programów wprowadzających, programy wyprowadzające mają prostą strukturę. Ograniczymy się do omówienia tylko dwóch programów wyprowadzających, a mianowicie stałoprzecinkowego programu wyprowadzająco-przeliczającego (bardzo uproszczonego) i programu dla wyprowadzania zawartości kolejnych miejsc pamięci w postaci rozkazów i pseudorozkazów.

Stałoprzecinkowy program wyprowadzająco-przeliczający łącznie z podstawowym programem wprowadzającym i parametrami jest nagrany na stałe na ścieżce nr zero. Program ten służy do drukowania liczb stałoprzecinkowych co do modułu mniejszych od jedności w układzie dziesiętnym, ponadto służy do drukowania odstępów potrójnych i napisów trzyznakowych.

Stałoprzecinkowy program wyprowadzająco-przeliczający zastosowany do liczby x ($|x|<1$), drukuje symbol $+0$, lub -0, w zależności od znaku liczby x oraz 12 cyfr dziesiętnych liczby x. Cyfry te uzyskuje się przez wielokrotne mnożenie x przez $2^{-4} \cdot 10$, drukowanie cyfry, która znalazła się na miejscu charakterystycznym akumulatora oraz arytmetyczne przesuwanie wyniku o 4 miejsca w lewo (czyli mnożenie wyniku przez 2^4). Po wydrukowaniu liczby program drukuje potrójny odstęp. Odstęp taki można również drukować bez drukowania jakiejkolwiek liczby.

Uwaga: Napis -0, tworzymy przez dodanie do napisu $+0$, liczby ósemkowej



Rys. 5-4. Schemat blokowy stałoprzecinkowego programu wprowadzająco-przeliczającego

01 0000, gdyż minus w kodzie wewnętrznym ma postać ósemkową 13 0000, zaś plus ma postać 12 0000.

Wywołanie stałoprzecinkowego podprogramu wyprowadzająco-przeliczającego dla przeliczania i drukowania liczby x (gdzie $|x|<1$) wygląda następująco:

```
a+0000   14   4104,
     1   12   4070,
     2   14   4102,
     3   04   0101.
```

Uwaga: Zakładamy tu, że liczba przeznaczona do drukowania znajduje się w akumulatorze; w przypadku gdy liczba ta znajduje się pod jakimś adresem różnym od 4104, do wywołania musimy dołączyć jeszcze jeden rozkaz. W przypadku, gdy liczba przeznaczona do drukowania znajduje się już pod adresem 4104, wywołanie skraca się o jeden rozkaz.

Parametr występujący w wywołaniu podprogramu dla drukowania liczb ósemkowych składa się z dwóch słów krótkich, pierwsze z nich (o adresie krótkim parzystym) określa *wejście* do programu, drugie (o adresie krótkim nieparzystym) jest skalą logiczną dla pętli drukującej.

Wywołanie podprogramu dla drukowania odstępu wygląda następująco:

```
d+0000   12   0072
     1   14   0102
     2   04   0101
```

Wywołanie podprogramu dla drukowania dowolnego napisu ma postać:

```
e+0000   12   0073
     1   14   0102
     2   12   < napis >
     3   04   0101
```

W przypadku gdy napis jest postaci „+0," wywołanie wygląda następująco:

```
e+0000   12   0073
     1   14   0102
     2   12   0064
     3   04   0101
```

```
START

0,25 EXP  +0,000000000000   =  +0,250000000000

0,25 EXP  +0,001999999978   =  +0,250500500318

0,25 EXP  +0,003999999957   =  +0,251002002623

0,25 EXP  +0,005999999935   =  +0,251504508894

                                          STOP
```

Rys. 5-5. Przykład drukowania wyników i napisów za pomocą stałoprzecinkowego programu przeliczająco-wyprowadzającego

Tablica 5-12

Stałoprzecinkowy program wyprowadzająco-przeliczający

Kolejny adres	Kolejny rozkaz		Uwagi
0032	12	4104	początek programu drukowania liczb
3	20	0041	
4	22	0004	
5	10	0064	
6	25	0000	
7	23	0006	
0040	30	0074	
1	03	0036	
2	12	4104	
3	26	0000	
4	14	4104	
5	27	0065	
6	16	4104	
7	25	0000	
0050	20	0004	
1	14	4104.	
2	12	0103	
3	23	0001	
4	14	0103	
5	03	0046	
6	12	0004	początek podprogramu drukowania odstępu
7	25	0000	początek podprogramu drukowania napisu
0060	23	0006	(1)
1	31	0074	
2	03	0057	
3	02	0101	
4	12	0025	+ 0,(2)
5	12	0000	
6	01	0000	
7	00	0002	
0070	02	0032	parametr dla drukowania liczb
1	17	7740	
2	02	0056	parametr dla drukowania odstępu
3	02	0057	parametr dla drukowania napisu
4	37	7777	parametr

(1) Pod adresem 0004 znajduje się parametr 24 2424 (patrz tabl. 5-6) będący potrójnym symbolem odstępu (patrz tabl. 2-6).
(2) Patrz tabl. 2-6

Na rysunku 5-4 podany jest schemat blokowy stałoprzecinkowego programu wyprowadzająco-przeliczającego.

Na rysunku 5-5 pokazana jest postać wyników obliczeń, wyprowadzonych na zewnątrz za pomocą omawianego wyżej podprogramu.

Podamy jeszcze kilka danych dotyczących tego programu — długość 38 słów krótkich, liczba wykonanych rozkazów 130 dla liczby, 13 dla trzyznakowego napisu. Kolejne rozkazy naszego programu podano w tabl. 5-12.

Tablica 5-13

Program drukowania kolejnych miejsc pamięci w kodzie rozkazowym

Kolejny adres	Kolejny rozkaz		Objaśnienia
$k + 0000$	12	<16 0000>	nastawienie dalekopisu na drukowanie cyfr i znaków
1	25	0000	
2	12	<22 0000>	wykonanie dodatkowego „wysuwu“
3	25	0000	
4	12	<21 0000>	ustawienie nowego wiersza
5	25	0000	
6	12	<22 0000>	
7	25	0000	
$k + 0010$	12	0120	drukowanie adresu
1	10	$k + 4054$	
2	04	$k + 0043$	
3	12	<24 0000>	drukowanie podwójnego odstępu
4	25	0000	
5	25	0000	
6	00	0120	drukowanie części operacyjnej
7	30	<37 0000>	
$k + 0020$	22	0006	
1	10	<00 0040>	
2	04	$k + 0043$	
3	12	<24 0000>	drukowanie pojedynczego odstępu
4	25	0000	
5	00	0120	drukowanie części adresowej
6	10	$k + 4054$	
7	04	$k + 0043$	
$k + 0030$	12	0120	przeadresowanie
1	10	0076	
2	14	0120	
3	11	0121	
4	03	$k + 0036$	
5	05	0120	stop
6	12	0120	sprawdzenie końca „bloczka“ złożonego z ośmiu słów
7	11	<00 0007>	
$k + 0040$	30	<00 0007>	
1	03	$k + 0004$	
2	02	$k + 0002$	skok do drukowania dodatkowego „wysuwu“
3	77	7777	podprogram
4	30	$k + 4052$	
5	06	$k + 0051$	
6	23	0003	
7	25	0005	
$k + 0050$	02	$k + 0044$	
1	01	$k + 0043$	
2	20	0000	5 zer 13 jedynek 16 zer
3	00	7777	0...0 1...1 0...0 parametr
4	20	0000	17 zer 16 zer
5	00	0000	0...0 1 0...0 parametr

W tablicy 5-13 podany jest program przeznaczony do drukowania zawartości kolejnych krótkich miejsc pamięci począwszy od n-tego, a skończywszy na m-tym (gdzie $n \leqslant m$), w postaci rozkazów i pseudorozkazów. Program ten wywołujemy w następujący sposób:

1) umieszczamy pod adresami 0120 i 0121 parametry

$$\begin{array}{rrl} 0120 & 12 & n, \\ 1 & 12 & m+1, \end{array}$$

gdzie n i m pierwszy i ostatni adres krótkich słów przeznaczonych do drukowania;

2) wykonujemy rozkaz

$$02 \quad k+0000.$$

Program podany w tabl. 5-13, drukuje kolejne rozkazy i pseudorozkazy, począwszy od zawartości adresu n, a skończywszy na zawartości adresu m.

Tablica 5-14

Parametry rozkazowe

Adres parametru	Postać parametru w kodzie rozkazowym
0066	01 0000
0021	02 0000
0065	12 0000
0011	23 0000
0007	24 0000
0036	25 0000
0043	26 0000

Tablica 5-15

Parametry arytmetyczne

Adres parametru	Postać parametru w kodzie rozkazowym	Wartość arytmetyczna parametru
0077	00 0000	0 (zero krótkie)
0076	00 0001	2^{-16}
0074	37 7777	-2^{-16}
0067	00 0002	2^{-15}
0075	17 7777	$1-2^{-16}$
4076	—	2^{-33}
4074	—	$1-2^{-33}$
0066	01 0000	2^{-4}
0021	02 0000	2^{-3}
0065	12 0000	$10 \cdot 2^{-4}$

W tablicach 5.14, 5.15 podane są parametry zapisane na stałe na ścieżce zerowej; z parametrów tych będziemy wielokrotnie dalej korzystali.

5.3. URUCHOMIENIE MASZYNY

W punkcie tym nie będziemy zajmowali się bliżej stroną techniczną uruchomienia, sprawy te omawia szczegółowo instrukcja techniczna obsługi danej maszyny. Natomiast zakładając, że maszyna jest włączona do sieci oraz że wszystkie napięcia przyjmują prawidłowo wartości, omówimy czynności wykonywane przez matematyka dla rozpoczęcia obliczeń.

1. Za pomocą przycisku W15 gasimy neonówkę S2 (zegar), w przypadku gdy neonówka S2 nie zgasła lub zapaliła się ponownie, nie można prowadzić obliczeń, gdyż nastąpiło uszkodzenie maszyny (patrz Załącznik 2, pulpit sterowania).

2. Zerujemy wszystkie rejestry maszyny przez przyciśnięcie przycisku W4 (patrz jak wyżej).

3. Ustawiamy przełącznik W3 (I-II takt pracy) w położeniu górnym „I takt pracy" (patrz jak wyżej).

4. Ustawiamy przełącznik W1 (praca ciągła — praca krokowa) w położeniu górnym „praca ciągła" (patrz jak wyżej).

5. Zakładamy taśmę perforowaną z testem lub programem, który chcemy wykonać, pod głowicę czytnika, zamykamy głowicę czytnika, włączamy oświetlenie czytnika za pomocą przełącznika znajdującego się pod czytnikiem. Wówczas czytnik samoczynnie przesuwa taśmę zatrzymując się na rządku pierwszym przeznaczonym do czytania.

6. Przyciskamy przycisk W2 (start) (patrz jak wyżej). Wówczas maszyna zaczyna wykonywać rozkazy umieszczone pod kolejnymi adresami począwszy od adresu 0000. Jak już wiemy, pod adresem 0000 znajduje się początek programu wprowadzającego, który spowoduje odczytanie grupy rozkazów taśmowych początku czytania (punkt 5.1.1, podstawowy program wprowadzający), perforowanych na taśmie, a następnie zacznie wprowadzać kolejne rozkazy i pseudorozkazy programu. Jak mówiliśmy już w punktach 5.1.2 i 5.1.3, wywołania programów wprowadzająco-przeliczających dokonujemy za pomocą rozkazów taśmowych, oczywiście po uprzednim wprowadzeniu tych programów do maszyny.

Uwaga 1: W przypadku gdy chcemy uruchomić maszynę, która zatrzymała się samoczynnie po wykonaniu pewnego etapu obliczeń, w zależności od budowy wykonywanego programu nastawimy przełącznik W3 w którymś z położeń: „I takt pracy" lub „II takt pracy" i naciskamy przycisk „start".

Uwaga 2: W przypadku gdy w czasie obliczeń zapali się neonówka S2 (zegar), należy zatrzymać maszynę i sprawdzić, czy neonówka S2 daje się zgasić przy użyciu przycisku W15. Gdy neonówka nie daje się zgasić, należy przystąpić do kontroli technicznej, w przypadku zaś gdy neonówka S2 daje się zgasić, należy sprawdzić maszynę testem (programem kontrolnym), po czym należy powtórzyć obliczenia.

5.4. SZUKANIE BŁĘDÓW W PROGRAMACH

Po wprowadzeniu programu do maszyny należy przeprowadzić szczegółową kontrolę programu, następnie zaś wykonać próbne obliczenia np. dla takich wartości argumentów, dla których znamy wyniki obliczeń. W przypadku gdy próbne obliczenia wypadną dobrze, przystępujemy do właściwych obliczeń. Sprawą próbnych obliczeń nie będziemy się zajmowali bliżej ze względu na trudność spowodowaną wielką różnorodnością metod numerycznych stosowanych w obliczeniach automatycznych. Natomiast sprawę formalnej kontroli programu, to co Rosjanie nazywają „otładka programmy", omówimy dosyć dokładnie.

Zakładając, że schemat blokowy programu jest ułożony prawidłowo, kontrolujemy, czy nie powstały błędy w czasie kodowania, dziurkowania lub wprowadzania programu do maszyny. Najprostszą metodę takiej kontroli zastosowano przy eksploatacji angielskiej maszyny cyfrowej EDSAC, polega ona na drukowaniu części operacyjnej kolejno wykonywanych rozkazów programu, przy czym kontrola ta była jednocześnie próbnym obliczeniem. W przypadku wykonywania rozkazu skokowego program kontroli działania uruchamianego programu drukował wielokrotny odstęp między kodami operacji. Metodę tę można zmodyfikować o tyle, że w przypadku gdy program główny wywołuje podprogram, program kontroli działania uruchamianego programu sygnalizuje ten fakt, po czym nie kontrolując podprogramu (zakładając, że jest on poprawny) dalej kontroluje program główny. Program dla tego typu kontroli programów jest prostym programem interpretacyjnym (punkt 6.1), pobierającym kolejne rozkazy programu kontrolowanego do wykonania z drukowaniem w części operacyjnej wykonywanego rozkazu w przypadku, gdy jest to rozkaz skokowy; program interpretacyjny poza kodem operacji drukuje wielokrotny odstęp, w przypadku zaś wywołania podprogramu drukuje również część adresową rozkazu wywołującego, po czym przechodzi do następnego rozkazu programu głównego. Omówiony powyżej program kontroli jest prosty i dosyć uniwersalny, wymaga natomiast dużo czasu dla samej kontroli i starannego przygotowania próbnych obliczeń, łącznie z obliczeniem wartości funkcji obliczanych przez podprogramy.

Poza omówioną wyżej metodą, bywają używane metody dające dużo dokładniejszą kontrolę, metody te mają jednak tę wadę, że wymagają dużego nakładu pracy i trwają stosunkowo długo, znacznie dłużej niż metoda stosowana na maszynie EDSAC. Ogólna charakterystyka tych metod jest następująca:

1) bardziej uniwersalna, polegająca na układaniu przez program kontrolowany schematu blokowego kontrolowanego programu,

2) indywidualna, polegająca na kontroli programu i porównywaniu go z zadanym schematem blokowym.

W praktyce najwygodniej jest po prostu uruchamiać program przez wykonywanie próbnych obliczeń, a następnie wyprowadzenie za pomocą programu podanego w tabl. 5-13 program z maszyny (lub części programu, zawierające liczniki cykli i ślady). Następnie kontrolujemy w wyprowadzonym programie stany liczników, na podstawie których szukamy błędów.

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

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

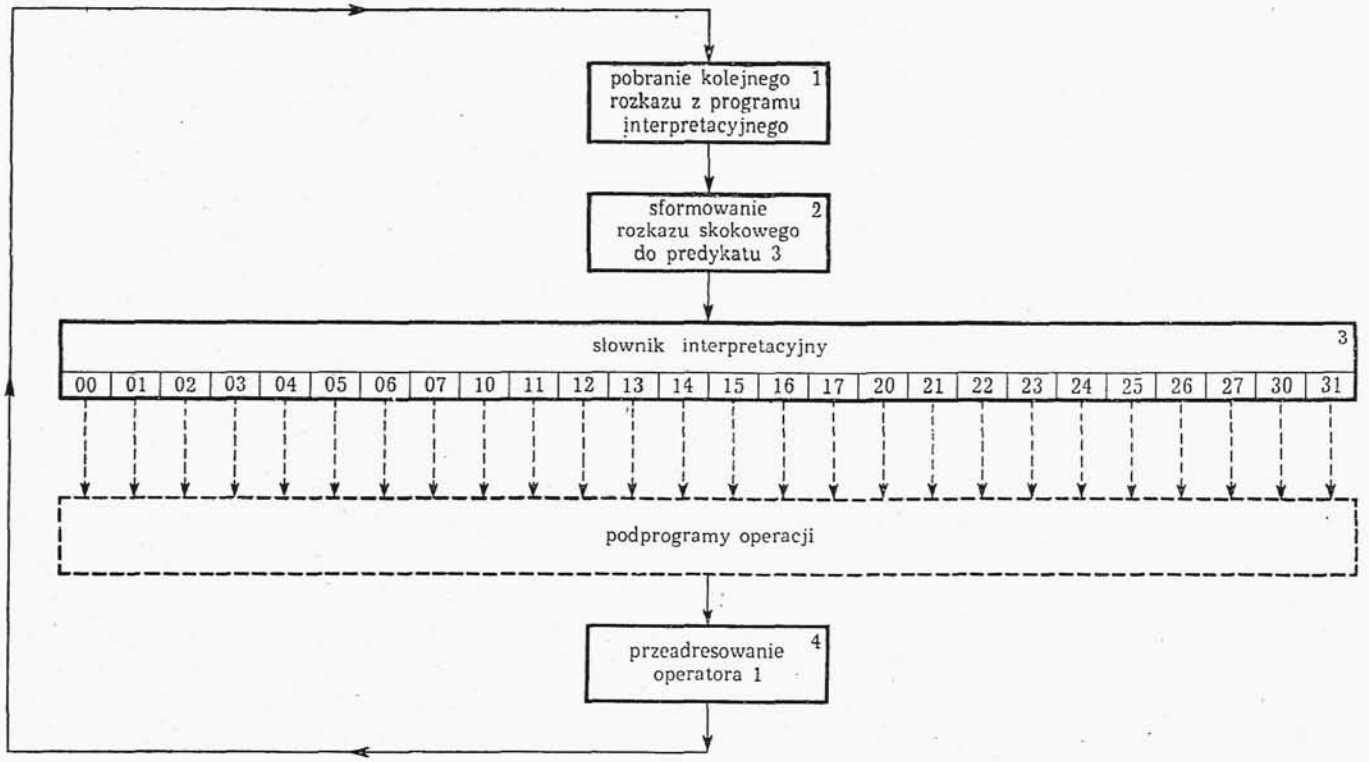
dzielenie

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

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:

$s + 0000$	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	$p_{17} + 0000$,

$s + 0020$ 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 w komórce roboczej
1	10	<12 0000>	
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 w komórce roboczej
1	10	<11 0000>	
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

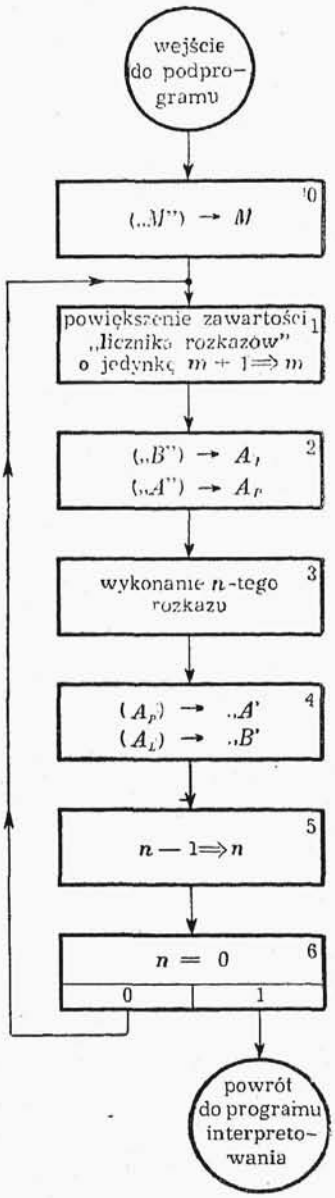
„26" N — obliczenie modułu liczby zespolonej z, znajdującej się w pseudoakumulatorze. Podprogram korzysta z jednej komórki roboczej: kom. rob. 1 oraz z podprogramu obliczenia pierwiastka kwadratowego

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.

$d + 0010$		
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),

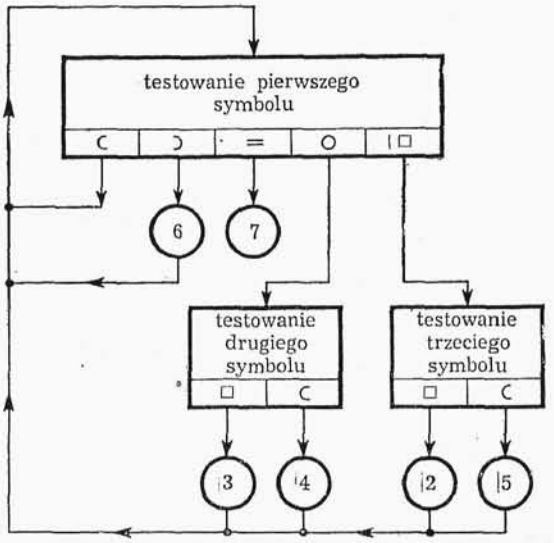
jeśli ma znak wykorzystania, to programuj

$s+0$	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

$s+0$	12	$<a>$
1	14	P,
2	12	$<b>$,
3	04	„·“.

3. Następnie stosujemy dyrektywę 3.

$s+4$	12	$<c>$,
5	04	„+“.

4. Następnie stosujemy powtórnie dyrektywę 3.

$s+6$	12	$<d>$,
7	04	„·“.

5. Następnie stosujemy po raz trzeci dyrektywę 3.

$s+10$	12	$<e>$,
1	04	„+“.

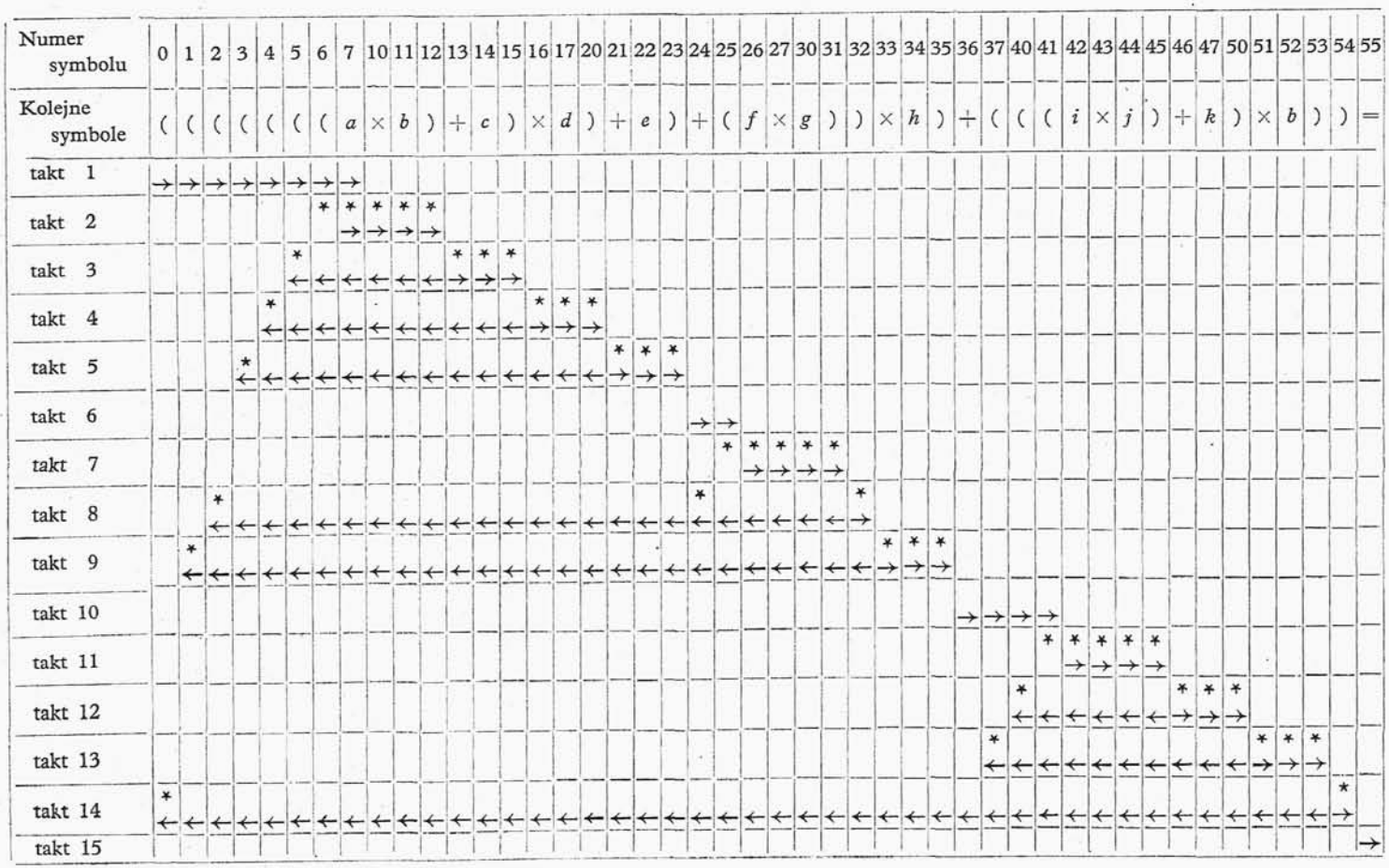
6. Z kolei stosujemy dyrektywę 4

$s+12$	14	R.

7. Następnie stosujemy dyrektywę 2

$s+13$	12	$<f>$,
4	14	P,
5	12	$<g>$,
6	06	„·“.

Tablica 6-21



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	<β>	
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>	+ 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>	$\alpha_6\alpha_7\alpha_8\alpha_9$ [$\alpha_{10}\alpha_{11}\alpha_{12}$] $\alpha_{13}\alpha_{14}\alpha_{15}\alpha_{16}$ ←
	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	t+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	
	6	02	a+0114	wywołanie podprogramu 3 → (H)
	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	$r + 0000$.

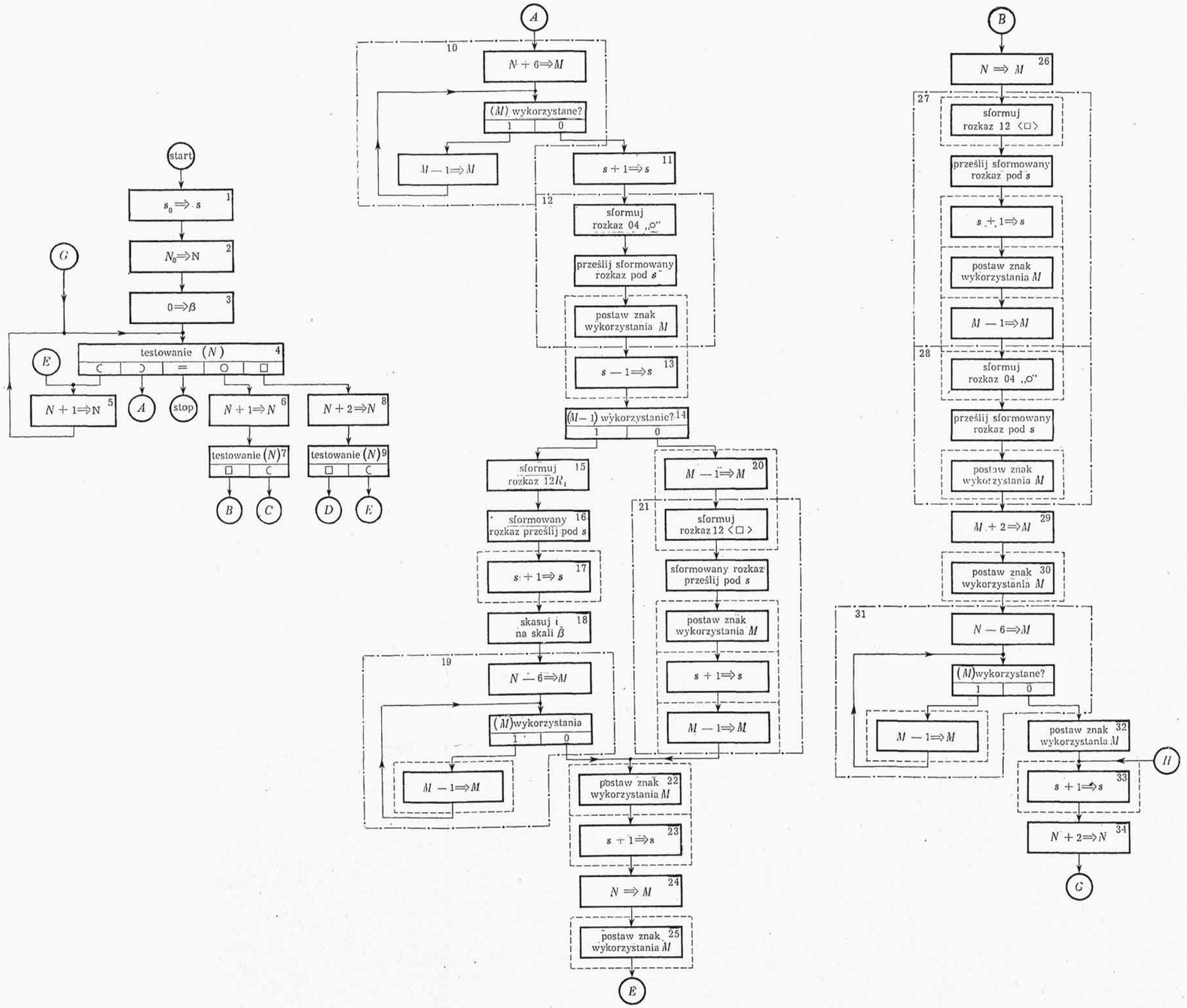
4. Szukanie w kierunku malejącym adresów pierwszego symbolu, w którym nie ma znaku wykorzystania, począwszy od N-6

$k + 0000$		
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:

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

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),

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

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

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),

przy czym będziemy zakładali, że zarówno układy produkcyjne, jak i rozdzielcze działają natychmiastowo. Rzeczywistość zaś będziemy modelowali poprzez łączenie układów produkcyjnych i rozdzielczych z odpowiednimi układami opóźniającymi.

Wejścia i wyjścia do trzech pierwszych rodzajów układów mogą być zarówno wejściami i wyjściami rzeczowymi (np. wejścia surowcowe czy energetyczne), jak i wejściami i wyjściami informacyjnymi.

Uwaga: W modelach ekonomicznych przyjmuje się, że kredyt jest informacją.

Układ planujący jest to układ, który ma wyjścia tylko informacyjne; wyjścia te sterują układami rozdzielczymi, które z kolei wykonują decyzje podziału surowców, energii, półfabrykatów i siły roboczej między odpowiednie układy produkcyjne. W zależności od zmian wartości „wektor funkcji" opisującej działania każdego z układów rozdzielczych, realizujemy różne polityki gospodarcze.

Układy produkcyjne są opisane dwoma rodzajami funkcji:

1. Produkcja jest opisywana za pomocą ważonego minimum[1]), ze stanów wejść.

2. Współczynniki zużycia wyposażenia produkcyjnego są opisywane za pomocą funkcji ciągłych względem czasu wykorzystania. W najprostszym przypadku współczynniki te są iloczynami ilości produkcji przez pewne stałe zużycia.

Układy opóźniające są opisywane przez funkcje identycznościowe, przesunięte w czasie o stałą opóźnienia. Układ planujący jest najbardziej złożonym układem, poza trywialnym przypadkiem, kiedy wyjścia tego układu przyjmują stałe wartości, bez względu na stan wejść.

Przy modelowaniu układów dynamicznych należy traktować je jako układy względnie odosobnione. Wpływ świata zewnętrznego na model uwzględniamy bądź poprzez przyjęcie pewnych wartości średnich zaburzeń powodowanych przez wejścia niesterowane na układ, bądź przez przyłożenie zaburzeń losowych z hipotecznie przyjętym rozkładem (punkt 6.3).

Omówienie szczegółowo konkretnego przykładu zajęłoby zbyt dużo miejsca, dlatego też zainteresowanych czytelników odsyłamy do pracy H. Greniewskiego (Bibliografia [5]). Zaprogramowanie przedstawionych wyżej funkcji, po zapoznaniu się z poprzednimi rozdziałami, czytelnik z łatwością sam przeprowadzi. Ze swej strony omówimy kilka spraw związanych z samym procesem układania programów modelujących układy dynamiczne.

Przy bardziej złożonych układach dynamicznych dość trudno narysować szczegółowy schemat blokowy układu, dla ułożenia programu zaś konieczna jest bardzo szczegółowa znajomość wszystkich połączeń. Trudność tę można ominąć przez zastąpienie schematu blokowego zero-jedynkową macierzą połączeń (Bibliografia [5]). Przyporządkowania tego dokonujemy w sposób następujący: niech układ dynamiczny składa się z n układów względnie odosobnionych $V_1, V_2, \ldots, V_n$. Świat zewnętrzny będziemy traktowali jako dodatkowy układ względnie odosobniony. Następnie wypiszemy kolejno poziomo i pio-

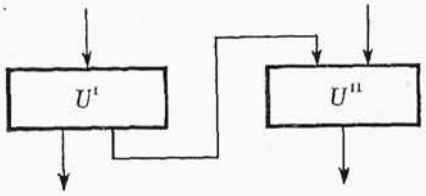
---

(1) Przez ważone minimum n liczb $x_1, \ldots, x_n$ z wektorem wag $a_1, \ldots, a_n$ rozumiemy funkcję,

$$P(x_1, \ldots, x_n) = \min (a_1 x_1, a_2 x_2, \ldots, a_n x_n).$$

W ekonomii składowe wektora wag $a_1, \ldots, a_n$ noszą nazwę współczynników technologicznych.

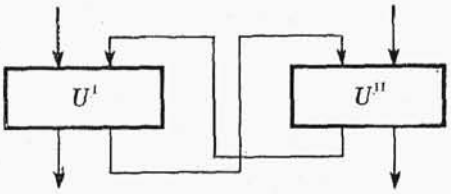
nowo wszystkie układy względnie odosobnione. Poziomo wypisane w wierszu układy będziemy traktowali jako wejścia. Pionowo wypisane w kolumnie układy będziemy traktowali jako wyjścia. Jeśli wyjście z i-tego układu wchodzi na wejście do j-tego układu to



wejście

wyjście: z \ na	zewnątrz	U'	U''
zewnątrz	0	1	1
U'	1	0	1
U''	1	0	0

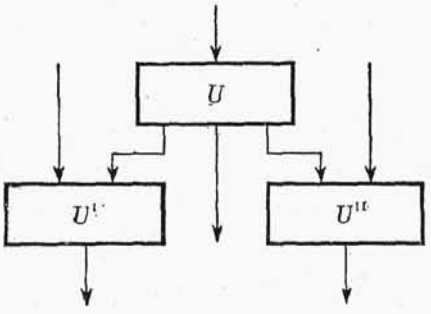
Rys. 6-5.



wejście

wyjście: z \ na	zewnątrz	U'	U''
zewnątrz	0	1	1
U'	1	0	1
U''	1	1	0

Rys. 6-6.



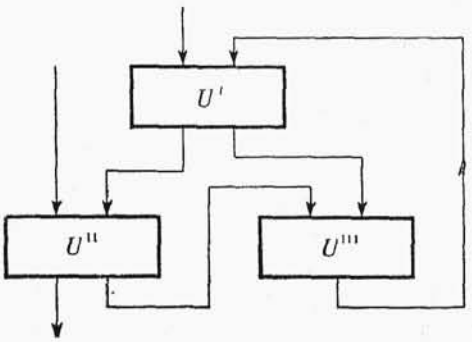
wejście

wyjście: z \ na	zewnątrz	U	U'	U''
zewnątrz	0	1	1	1
U	1	0	1	1
U'	1	0	0	0
U''	1	0	0	0

Rys. 6-7.

na przecięciu i-tego wiersza i j-tej kolumny stawiamy jedynkę, jeśli zaś wyjście z i-tego układu nie wchodzi na wejście j-tego układu to stawiamy zero. Na rysunkach 6-5÷6-8 podane są przykłady układów i odpowiadających im matryc.

W przypadku gdy układ względnie odosobniony wchodzący w skład naszego układu działa jako predykat, można zbudować macierz, jak na przykładzie podanym na rys.

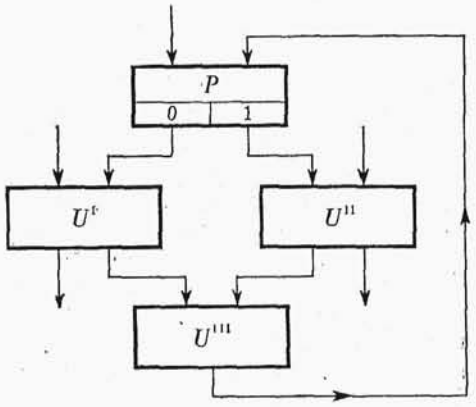


wejście

wyjście: z \ na	zewnątrz	U'	U''	U'''
zewnątrz	0	1	1	0
U'	0	0	1	1
U''	1	0	0	1
U'''	0	1	0	0

Rys. 6-8.

6-9, gdzie przez P oznaczyliśmy układ względnie odosobniony, będący predykatem, a przez p oznaczyliśmy zmienną logiczną odpowiadającą spełnieniu warunków określonych przez predykat P, przez $\bar{p}$ oznaczamy funkcję $1-p$.



wejście

wyjście: z \ na	zewnątrz	P	U'	U''	U'''
zewnątrz	0	1	1	1	0
P	0	0	$\bar{p}$	p	0
U'	1	0	0	0	1
U''	1	0	0	0	1
U'''	0	1	0	0	0

Rys. 6-9.

Tak zbudowaną matrycę możemy traktować jako skalę logiczną dla wywołania podprogramów, przy założeniu, że układy względnie odosobnione ustawiliśmy w kolejności zgodnej z zależnościami między wzorami opisującymi poszczególne układy względnie odosobnione.

Pozostaje jeszcze do omówienia cel, w jakim budujemy modele układów dynamicznych. Cel ten może być trojaki:

1. Badanie zachowania się stabilności układu w zależności od charakteru sterowania.
2. Badanie wpływu poszczególnych wyjść informacyjnych na działania sterowania i ewentualna eliminacja tych wyjść informacyjnych, których wpływ na działanie sterowania można uznać za mały.
3. Badanie wpływu centralnego układu sterowania na działanie różnych grup układów względnie odosobnionych naszego układu dynamicznego i ewentualne opracowa-

nie lokalnego sterowania dla pewnych autonomicznych grup układów względnie odosobnionych.

O ile badania omówione w punkcie 1 są stosunkowo proste i w zasadzie dysponujemy odpowiednimi kryteriami dla tych badań, o tyle dla badań określonych w punktach 2 i 3 nie ma dotychczas jakichś ogólnych kryteriów i wymagają one za każdym razem indywidualnego podejścia.

Wśród wielu techników panuje przekonanie, że właściwymi urządzeniami do badania układów dynamicznych są maszyny analogowe, nie zaś maszyny cyfrowe. Wydaje się, że w tym miejscu należałoby skorygować ten pogląd. W przypadku małych układów dynamicznych stosowanie maszyn analogowych daje dobre rezultaty, w przypadku układów bardziej złożonych maszyny analogowe nie dają zadowalających rezultatów, ze względu na małą (ograniczoną) dokładność obliczeń. Ponadto przy projektowaniu urządzeń analogowych do badania bardziej złożonych układów dynamicznych trzeba wykonać obliczenia projektowe na UPMC. Tego rodzaju postępowanie mija się z celem.

6.6. MASZYNA CYFROWA A CYBERNETYKA

Cybernetyka, nauka o sterowaniu i łączności w maszynach, organizmach żywych i społeczeństwach, w znacznej swej części zajmuje się budowaniem modeli (w rozumieniu definicji podanej w punkcie 6.0) różnorakich zjawisk zachodzących bądź w organizmach żywych, bądź w społeczeństwach. Modele częściowe lub całościowe organizmów żywych pozwalają nieraz poznać hipotetyczny mechanizm zjawiska. Ponadto badanie modeli pozwala niejednokrotnie zaplanować doświadczenie, które to z kolei pozwoli rozstrzygnąć, jak dane reakcje zrealizowała przyroda.

UPMC stały się obecnie podstawowym instrumentem badań cybernetycznych. Cyfrowe modelowanie wielu procesów badanych przez cybernetykę jest dziś metodą niemal klasyczną. Jako pierwsi zastosowali do badań cybernetycznych UPMC amerykańscy matematycy B.G. Farley i W.A. Clark, którzy prowadzili badania nad układami samoorganizującymi się. Użycie UPMC jako instrumentu modelowania z reguły pozwala poważnie zredukować koszt i czas badań. Znacznie łatwiej ułożyć jest program dla UPMC niż budować skomplikowany analog.

Dla bliższego wyjaśnienia zastosowania UPMC do badań cybernetycznych omówimy program modelujący urządzenie zwane homeostatem Ashbiego (homeostaza, tj. zdolność przywracania przez organizm chwiejnej równowagi).

Prostym modelem matematycznym zjawiska homeostazy jest układ czterech równań jednorodnych algebraicznych liniowych:

$$\sum_{i=1}^{4} a_{ij}x_i = 0, \quad i = 1, \ldots, 4, \tag{6-13}$$

którego macierz spełnia następujące warunki:

1. Wyznacznik macierzy $||a_{ij}||$ jest zerem:

$$|a_{ij}| = 0.$$

2. Wyrazy głównej przekątnej macierzy są co do wartości bezwzględnej znacznie większe od pozostałych wyrazów macierzy.

3. Minory główne macierzy są różne od zera i ich bezwzględna wartość jest bliska bezwzględnej wartości wyrazów głównej przekątnej macierzy $||a_{ij}||$.

Przykładem spełniającym warunki $1 \div 3$ jest macierz

$$\begin{Vmatrix} 1 & 0 & 0 & \frac{1}{\sqrt[4]{2}} \\ \frac{1}{\sqrt[4]{2}} & 1 & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{64} & \frac{1}{8} & 1 & -\frac{1}{16} \\ 0 & \frac{1}{\sqrt{2}} & 0 & 1 \end{Vmatrix} \qquad (6\text{-}14)$$

Równanie i-te z układu (6-13) odpowiada wpływowi pozostałych x_j (gdzie $i \neq j$) na x_i. Jeśli teraz ustalimy którąś z wartości x_i, to przez skreślenie i-tego równania w układzie (6-13) otrzymamy układ 3 równań niejednorodnych, którego rozwiązania są jednoznacznie wyznaczone przez wartość x_i. Podobnie jeśli ustalimy dwie wartości x_{i_1} i x_{i_2}, to przez skreślenie i_1 oraz i_2 równania w układzie (6-13), otrzymamy układ dwóch równań niejednorodnych, którego rozwiązania są jednoznacznie wyznaczone przez wartości x_{i_1} i x_{i_2}. Dalej, jeśli ustalimy trzy wartości: x_{i_1}, x_{i_2}, x_{i_3}, to przez skreślenie i_1, i_2, i_3 równania z układu (6-13) otrzymamy jedno równanie niejednorodne, którego rozwiązanie jest jednoznacznie wyznaczone przez wartości x_{i_1}, x_{i_2}, x_{i_3}.

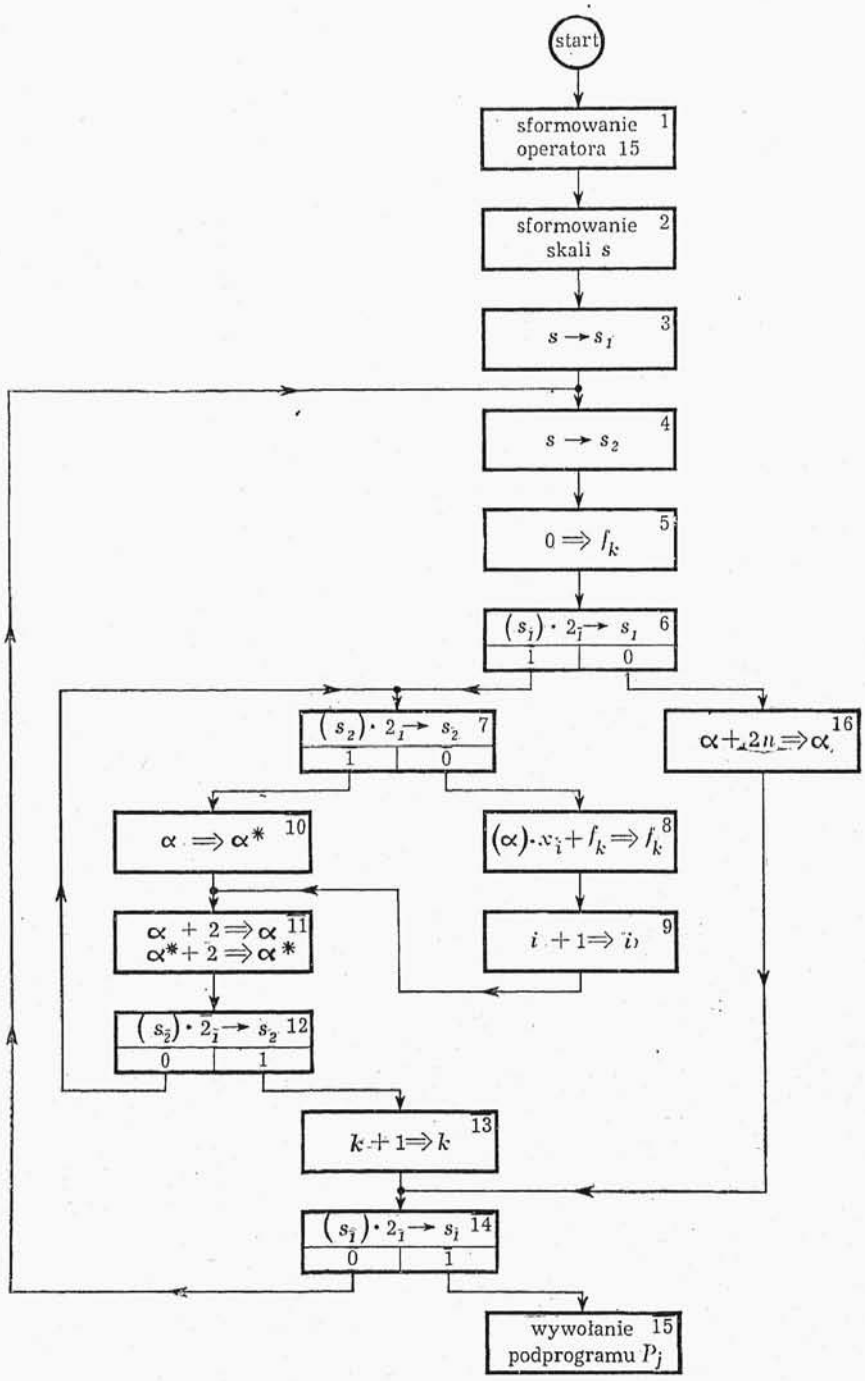
W przypadku ustalenia jednej albo dwóch zmiennych w układzie (6-13) możemy, w wyniku skreślenia odpowiednich równań z układu (6-13), układ równań algebraicznych liniowych rozwiązywać metodą iteracji prostej Gaussa (Bibliografia [12]). Kolejne przybliżenia rozwiązań układu równań będą odpowiadały ustalaniu się nowego stanu równowagi homeostatu.

Program modelujący powyższy proces będzie się składał z dwóch części:

1) podprogramu ustawiającego odpowiednią macierz i obliczającego wyrazy wolne dla układu równań i wybierającego odpowiedni podprogram dla rozwiązania sformowanego uprzednio układu równań algebraicznych liniowych.

2) podprogramów dla iteracyjnego rozwiązywania układów dwóch i trzech równań algebraicznych liniowych z drukowaniem wyniku po każdym kroku iteracyjnym i podprogramu dla rozwiązania jednego równania algebraicznego liniowego z równoczesnym drukowaniem wyników.

Obecnie omówimy szczegółowo część 1 programu modelującego. Na rysunku 6-10 przedstawiony jest schemat blokowy programu, w tablicy 6-25 podane są zakodowane operatory i predykaty z rys. 6-10. W programie przyjęto następujące oznaczenia: $\alpha = a_{11}$, α^* — adres pierwszego wyrazu sformowanej macierzy, i — indeks ($i = 1,2,3,4,$) ustalonego (lub ustalonych) iksów, j—ilość ustalonych zmiennych ($i = 1, 2, 3$) kodowana jako $j \cdot 2^{-16}$. (Przedstawiony fragment programu opracowała Z. Jankowska).



Rys. 6-10. Schemat blokowy części programu modelującego zjawisko homeostazy

Tablica 6-25

Część programu modelującego zjawisko homeostazy

Operator lub predykat	Kolejny adres	Kolejny rozkaz		Uwagi
1	$k+0000$	12	$<j>$	
	1	11	$<2^{-16}>$	
	2	26	0000	
	3	03	$k+0007$	
	4	12	$<04\ P_1>$	
	5	14	$k+0103$	
	6	02	$k+0017$	
	7	11	$<2^{-16}>$	
	$k+0010$	26	0000	
	1	03	$k+0015$	
	2	12	$<04\ P_2>$	
	3	14	$k+0103$	
	4	02	$k+0017$	
	5	12	$<04\ P_3>$	
	6	14	$k+0103$	
2	7	12	t_1	
	$k+0020$	10	$<i_1>$	
	1	10	$<i_2>$	
	2	10	$<i_3>$	
3	3	14	s	
	4	14	s_1	
4	5	12	s	
	6	14	s_2	
	7	12	<zero>	
	$k+0030$	14	f_k	
6	1	12	s_1	
	2	23	0001	
	3	14	s_1	
	4	06	$k+0104$	
7	5	12	s_2	
	6	23	0001	
	7	14	s_2	
	$k+0040$	03	$k+0054$	
8	1	12	α	
	2	14	4100	
	3	12	$<x_1>$	
	4	04	„·“	
	5	12	f_k	
	6	04	„+“	
	7	14	f_k	
9	$k+0050$	12	$k+0043$	
	1	10	$<2^{-15}>$	
	2	14	$k+0043$	
	3	02	$k+0056$	
10	4	12	α	

Tablica 6-25 (d. c).

Operator lub predykat	Kolejny adres	Kolejny rozkaz		Uwagi
	$k+0055$	14	α^*	
11	6	12	$k+0041$	
	7	10	$<2^{-15}>$	
	$k+0060$	14	$k+0041$	
	1	14	$k+0054$	
	2	12	$k+0055$	
	3	10	$<2^{-15}>$	
	4	14	$k+0055$	
12	5	10	s_2	
	6	23	0001	
	7	14	s_2	
	$k+0070$	06	$k+0035$	
13	1	12	$k+0045$	
	2	10	$<2^{-15}>$	
	3	14	$k+0045$	
	4	10	<02 0000>	
	5	14	$k+0030$	
	6	14	$k+0047$	
14	7	12	s_1	
	$k+0100$	23	0001	
	1	14	s_1	
	2	06	$k+0025$	
15	3	04	P_j	
16	4	12	$k+0041$	
	5	10	$<00\ 2n>$	
	6	14	$k+0041$	
	7	14	$k+0054$	
	$k+0110$	02	$k+0077$	

6.7. UWAGI KOŃCOWE

Na zakończenie rozdziału należy jeszcze powiedzieć parę słów o zastosowaniu modeli cyfrowych do sterowania z predykcją(1) procesami dynamicznymi.

Dla danego układu dynamicznego, który chcemy sterować z predykcją cyfrową, czas dzielimy na odcinki o stałej długości (rys. 6-11).

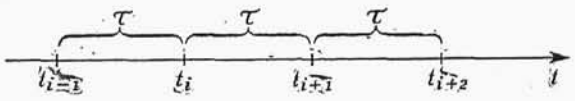
Długość τ dobieramy tak, aby spełniony był związek

$$\tau = \tau_1 + \tau_2 + \tau_3, \tag{6-15}$$

gdzie τ_1 oznacza czas potrzebny do przekazania informacji o stanie układu w poprzednim okresie czasu τ (od chwili t_{i-1} do chwili t_i), τ_2 — czas potrzebny na opracowanie otrzy-

(1) Sterowanie z predykcją jest to sterowanie z uwzględnieniem „hipotezy o stanie układu w chwili bieżącej".

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:

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

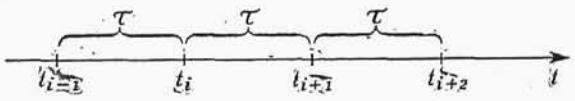
7. KODY ZEWNĘTRZNE

7.0. UWAGI WSTĘPNE

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);

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

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:

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

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 $\overline{\text{ZER}}$ M1 $\overline{+126}$ ADD 273 735 735 $\overline{\text{INA}}$ +001 M1 $\overline{\text{STO}}$	))	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)

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

b) część b (10 komórek)

$$
\begin{aligned}
&< b_{\alpha} > = 1 \cdot \alpha,\\
&< b_{s} > = 1 \cdot s,
\end{aligned}
$$

c) część d (10 komórek)

$$
\begin{aligned}
&< d_{k} > = 1 \cdot k,\\
&< d_{m} > = 1 \cdot m,\\
&< d_{j} > = 1 \cdot j,
\end{aligned}
$$

d) część e (10 komórek)

$$
\begin{aligned}
&< e_{s} > = 1 \cdot s,\\
&< e_{r} > = 1 \cdot r,
\end{aligned}
$$

2. Lista P (parametry)

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

3. Lista C (stałe i zmienne)

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

4. Lista K (schemat logiczny)

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

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

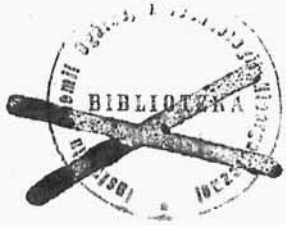
$$[\ \underset{m}{[}\ \underset{k}{[}\ a_{mk} \Longrightarrow d_k;\ \ 0 \Longrightarrow a_{mk}];\ 1 \Longrightarrow a_{mm};\ d_m + 1 \Longrightarrow c;$$

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

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

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

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



BIBLIOGRAFIA

[1] Акушский И. Я., *О некоторых общих вопросах программирования, Вопросы теории математических машин,* Сборник 1, Москва 1958.

[2] Bell W. D., *A management guide to electronic computer,* New York-Toronto-London 1957.

[3] Бондаренко В. Н., Плотников И. Т., Полозов П. П., *Программирование задач для машины „Урал",* 1957, Изд. Артиллерийской инж. академии им. Дзержинского.

[4] Booth K. H. V., *Programing for automatic digital calculator,* „London Butterworths Scentific Publications", 1958.

[5] Greniewski H., *Cybernetyka a modele ekonomiczne,* „Nauka Polski" VI, 1958.

[6] Greniewski H., Elementy cybernetyki sposobem niematematycznym wyłożone, Warszawa 1959, PWN.

[7] Ершов А. П., *Программирующая программа для быстродействующей электронной счетной машины,* Москва, 1958, Идз. АН СССР.

[8] Kitow A., *Elektroniczne maszyny cyfrowe* (tłumaczenie z rosyjskiego) Warszawa 1959, Wydawnictwo MON.

[9] Китов А. И., Криницкий Н. А., *Электронные цифровые машины и программирование,* 1959, Физматгиз.

[10] Линский В. С. *О выборе рационального количества адресов цифровой вычислительной машины, Вопросы теории Математических Машин,* Сборник 1, Москва 1958.

[11] Ляпунов А. А., *О логических схемах программ.* „Проблемы кибернетики", сборник 1, Москва 1958.

[12] Łukaszewicz J. i. Warmus M., *Metody numeryczne i graficzne,* Warszawa 1956, PWN.

[13] Cracen D. D Mc., *Digital computer programming,* New York 1957 John Wiley and Sons.

[14] Пашковский С., *Внешний код для цифровых вычислительных машин,* „Zastosowanie Matematyki" 1960, str. 381—392.

[15] Pawlak Z. i Wakulicz A., *Use of expansions with negative basis in the arithometer of a digital computer,* Bull. Ac. Pol. Sc. III, Vol. 5, No 3, 1957.

[16] Подловченко Р. И., *Об основных понятиях программирования,* „Проблемы кибернетики", Сборник 1, Москва 1958.

[17] Richards R. K., *Arithmetic operation in Digital Computer,* Toronto-New York-London 1955

[18] под ред. Шура-Бура М. Р., *Система стандартных подпрограмм,* Москва 1958, Государственное Издательство Физико-математической Литературы.

[19] Wilkes M. V., Wheeler D. J., Gill S., *The preparation of programs for electronic digital computer,* 2 ed., 1957, USA Massachusetts.

SKOROWIDZ

ZAŁĄCZNIKI

IBJ

ARKUSZ PROGRAMOWY EM

019

Dn. 23. 04. 59

(Program odracowała J. Rogińska-Empacherowa)

PROGRAM DZIAŁAŃ ZMIENNEGO PRZECINKA

$k+000$	0	77	7777	
	1	14	4102	
	2	12	4102	
	3	06	0043	k
	4	12	4100	
	5	06	0045	k
	6	04	0146	k
	7	11	0105	

dodawanie

$k+004$	0	21	0001	
	1	10	4100	
	2	02	0024	k
	3	12	4100	
	4	01	0000	k
	5	12	4102	
	6	14	4100	
	7	01	0000	k

$k+001$	0	03	0026	k
	1	10	0022	k
	2	14	0017	k
	3	12	0104	
	4	10	0076	
	5	14	0104	
	6	12	4102	
	7	21	7777	

$k+005$	0	77	7777	
	1	14	4102	
	2	12	0016	k
	3	10	0066	
	4	14	0016	k
	5	14	0037	k
	6	12	4102	
	7	04	0000	k

odejmowanie prawe

$k+002$	0	14	4102	
	1	12	4100	
	2	21	0001	
	3	10	4102	
	4	04	0162	k
	5	01	0000	k
	6	26	0000	
	7	10	0022	k

$k+006$	0	12	0016	k
	1	11	0066	
	2	14	0016	k
	3	14	0037	k
	4	12	4100	
	5	01	0050	k
	6	77	7777	
	7	22	0042	

odejmowanie lewe

$k+003$	0	14	0035	k
	1	12	0105	
	2	10	0076	
	3	14	0104	
	4	12	4100	
	5	21	7777	
	6	14	4100	
	7	12	4102	

$k+007$	0	10	4100	
	1	14	4102	
	2	23	0042	
	3	14	4100	
	4	12	0066	k
	5	14	0050	k
	6	02	0052	k
	7	77	7777	

mnożenie

Załącznik 1 (d. c.)

ARKUSZ PROGRAMOWY EM

IBJ | 019

Dn. 23. 04. 59 str. (2)

k+010	0	14	4102		
	1	12	4102		
	2	06	0114	k	
	3	12	4100		
	4	06	0113	k	
	5	04	0146	k	
	6	10	0105		
	7	14	0104		

k+014	0	10	4100		
	1	23	0042		
	2	14	4100		
	3	23	0042		
	4	04	0116	k	
	5	01	0136	k	
	6	77	7777		podprogram rozformowania
	7	21	0013		

k+011	0	27	4100		
	1	16	4102		
	2	04	0162	k	
	3	01	0077	k	
	4	14	4100		
	5	01	0077	k	
	6	77	7777		dzielenie
	7	14	4102		prawe

k+015	0	14	0104		
	1	23	0021		
	2	14	4100		
	3	12	4102		
	4	21	0013		
	5	14	0105		
	6	23	0021		
	7	14	4102		

k+012	0	12	4102		
	1	03	0123	k	
	2	05	0220	k	
	3	12	4100		
	4	06	0135	k	
	5	04	0146	k	
	6	11	0105		
	7	10	0076		

k+016	0	12	0104		
	1	01	0146	k	
	2	77	7777		podprogram normalizacji
	3	14	4100		
	4	12	4100		
	5	06	0214	k	
	6	10	4100		
	7	37	0176	k	

k+013	0	14	0104		
	1	12	4100		
	2	21	0001		
	3	17	4102		
	4	04	0162	k	
	5	01	0116	k	
	6	77	7777		dzielenie
	7	22	0042		lewe

k+017	0	14	4100		
	1	12	0104		
	2	11	0076		
	3	14	0104		
	4	12	4100		
	5	02	0166	k	
	6	77	7777		
	7	12	0104		

Załącznik 1 (d. c.)

ARKUSZ PROGRAMOWY EM

IBJ | 019

Dn. 23. 04. 59 str. ③

$k+020$	0	11	0215	k	0			
	1	03	0203	k	1			
	2	05	0220	k	2			
	3	10	0216	k	3			
	4	03	0212	k	4			
	5	12	4100		5			
	6	22	0021		6			
	7	10	0104		7			

$k+021$	0	20	0013		0			
	1	02	0213	k	1			
	2	12	0077		2			
	3	14	4100		3			
	4	01	0162	k	4			
	5	00	0040		5			
	6	00	0100		6			
	7	36	0000		7			

$k+022$	0	12	0217	k	0			
	1	25	0000		1			
	2	06	0220	k	2			
	3	05	0000		3			
	4				4			
	5				5			
	6				6			
	7				7			

	0				0			
	1				1			
	2				2			
	3				3			
	4				4			
	5				5			
	6				6			
	7				7			

Załącznik 2

OPIS PULPITU STEROWANIA MASZYNY TYPOWEJ

Pulpit sterowania służy do informowania obsługi UPMC o aktualnej pracy maszyny oraz umożliwia ingerowanie obsługi na prace UPMC. Przedstawiony niżej pulpit sterowania maszyny przykładowej umożliwia bardzo złożone manipulacje ręczne. Na przedstawionym pulpicie znajdują się neonówki, przyciski i przełączniki, których działanie omówimy kolejno.

Neonówki:

S0 — zapalanie się tej neonówki sygnalizuje, że $W = 1$,
S1 — zapalenie się tej neonówki sygnalizuje, że $N = 1$,
S2 — zapalenie się tej neonówki sygnalizuje, że przyczyną zatrzymania się maszyny było uszkodzenie zegara maszyny,
S3 — zapalenie się tej neonówki sygnalizuje zatrzymanie się maszyny w wyniku powstałego błędu w pracy, w przypadku gdy przełącznik $W10$ jest włączony,
S4 — zapalenie się tej neonówki sygnalizuje zatrzymanie się maszyny w wyniku rozkazu „stop",
N0, N1, ... , N16 — neonówki te pokazują nam w czasie pracy maszyny kolejne stany rejestru dyspozytora,
ND — zapalenie się tej neonówki sygnalizuje pojawienie się informacji do drukowania w przypadku gdy:
1) albo silnik dalekopisu nie jest włączony,
2) albo gdy poprzednie drukowanie nie zostało jeszcze wykonane,
C0, C1, ... , C4 — neonówki te pokazują nam zawartość miejsca charakterystycznego akumulatora, przeznaczoną do drukowania,

Przełączniki

RZ0, RZ1, ... , RZ16 — przełączniki dwupołożeniowe służące do ręcznego ustawienia słowa krótkiego, które następnie za pomocą przełączników W9 i W14 wprowadzamy do któregoś z rejestrów maszyny,
RD0, RD1, ... , RD16 — przełączniki dwupołożeniowe służące do ręcznego sterowania maszyny przy odpowiednim położeniu przełączników W11,
RS0, RS1, ... , RS4 — przełączniki dwupołożeniowe służące do wybrania ścieżki na bębnie, z której za pomocą przełączników RF0 i RF1 wybieramy 8 kolejnych 34-bitowych komórek, których zawartość oglądamy na wizjerze pamięci,
RF0, RF1 — przełączniki dwupołożeniowe służące do wybrania 8 kolejnych 34-bitowych komórek ze ścieżki wybranej za pomocą przełączników RS0 — RS4,
W1 — Przełącznik dwupołożeniowy służący do ustawienia maszyny do pracy ciągłej lub skokowej,
W2 — przycisk startu służący do wpuszczania impulsu do pętli operacji: w zależności od położenia przełącznika W1 maszyna po przyciśnięciu przycisku W2 zacznie wykonywać kolejne rozkazy programu albo wykona tylko jeden kolejny rozkaz programu,
W3 — przełącznik dwupołożeniowy; przy jednym z jego położeń naciśnięcie przycisku W2 powoduje powtarzanie poprzednio wykonanego rozkazu, w zależności od położenia przełącznika W3 maszyna uruchamia albo pierwszy, albo drugi takt pracy,
W4 — przycisk służący do zerowania wszystkich rejestrów maszyny,
W5 — przełącznik dwupołożeniowy służący do oglądania zawartości licznika rozkazów,
W6 — przełącznik dwupołożeniowy służący do oglądania rejestrów 34-pozycyjnych,
W7 — przycisk służący do zerowania wybranego za pomocą przełącznika W8 rejestru,
W8 — przełącznik czteropołożeniowy, służący do wybierania rejestru przeznaczonego do zerowania,
W9 — przełącznik pięciopołożeniowy, służący do wybierania rejestru, do którego chcemy wprowadzić słowo krótkie nastawiono za pomocą przełączników RZ0-RZ16 i W14, w jedynym z położeń swoich przełącznik umożliwia oglądanie na wizjerze pamięci zawartości komórek pamięci wybranych przełącznikami RS0 — RS4 i RF0 — RF1(1),

W10 — przełącznik dwupołożeniowy, w jednym ze swoich położeń powoduje zatrzymanie maszyny w przypadku przekroczenia zakresu (sygnał $N = 1$ stop),

W11 — przełącznik dwupołożeniowy, w jednym z położeń tego przełącznika maszyna ustawiona jest do normalnej pracy, w drugim z położeń — sterowanie zamiast z rejestru dyspozytora odbywa się bezpośrednio z pulpitu sterowania z przełączników RD0, RD1, ..., RD16,

W12 — przycisk służący do gaszenia neonówki S3,

W13 — przycisk służący do zerowania rejestru nadmiaru N, a tym samym do gaszenia neonówki S1,

W14 — przycisk służący do przesyłania słowa krótkiego ustawionego na przełącznikach RZ0, RZ1,..., RZ16, do rejestru wybranego za pomocą przełącznika W9,

W15 — przycisk służący do gaszenia neonówki S2,

W16 — przełącznik dwupołożeniowy włączający drukowanie,

Z0, Z1, ..., Z4 — przełączniki dwupołożeniowe służące do generowania warunku $W=1$ dla rozkazu „drukuj"; jeśli któryś z tych przełączników jest w położeniu „1", to w przypadku pojawienia się na odpowiadającej mu pozycji miejsca charakterystycznego akumulatora jedynki i rozkazu „drukuj" zostaje generowany sygnał $W = 1$,

ZB0, ZB1, ..., ZB4 — przełączniki dwupołożeniowe służące do blokowania poszczególnych pozycji miejsca charakterystycznego, tzn. niezależnie od stanu tej pozycji miejsca charakterystycznego dalekopis otrzymuje na zablokowanej pozycji sygnał „zero".

Y0, Y1, ..., Y4 — przełączniki dwupołożeniowe służące do generowania sygnału $W = 1$ dla rozkazu „czytaj"; jeśli któryś z tych przełączników jest w położeniu „1", to w przypadku pojawienia się na odpowiadającej im pozycji taśmy perforowanej sygnału „jedynka" i rozkazu „czytaj" zostaje generowany sygnał $W = 1$,

YB0, YB1, ..., YB2 — przełączniki dwupołożeniowe służące do blokowania poszczególnych pozycji taśmy perforowanej, tzn. niezależnie od stanu tej pozycji taśmy perforowanej na pozycję zablokowaną miejsca charakterystycznego wchodzi zawsze przy rozkazie czytaj — sygnał „zero".

Uwaga: blokada wyjścia (lub wejścia) nie wpływa na generowanie warunku za pomocą przełączników Z (lub Y).

---

(1) Dla obejrzenia zawartości któregoś z rejestrów maszyny należy:
1) zatrzymać maszynę,
2) przełącznik W11 ustawić w położeniu — sterowanie ręczne,
3) wybrać rejestr za pomocą przełącznika W9, W5 albo W9, W6.

LISTA ROZKAZÓW

Nr rozkazu	Kod ósemkowy rozkazu	Zapis symboliczny rozkazu	Warunek generacji sygnału $W = 1$	Warunek generacji sygnału $N = 1$	Nazwa i szczegółowy opis rozkazu
1	2	3	4	5	6
0	00	$(n) \rightarrow D$	—	—	Pobranie rozkazu: pobierz rozkaz zawarty w komórce o adresie krótkim n — wykonaj go. Uwaga: Po czym powróć do poprzedniej sekwencji rozkazów, o ile rozkaz wykonywany nie był rozkazem skokowym.
1	01	$(n) \rightarrow L$	—	—	Skok z podstawieniem: do licznika rozkazów L prześlij część adresową słowa odczytanego pod adres n.
2	02	$n \rightarrow L$	—	—	Skok: do licznika rozkazów L prześlij część adresową wykonywanego rozkazu (tzn. skocz do n).
3	03	$n \rightarrow L$ $W = 1$	—	—	Pierwszy skok warunkowy: jeżeli zawartość rejestru $W = 1$, to część adresową wykonywanego rozkazu prześlij do licznika rozkazów L; jeżeli zaś zawartość rejestru $W = 0$, to nie zmieniaj zawartości licznika rozkazów. Uwaga: Rozkaz pierwszego skoku warunkowego zeruje rejestr W.
4	04	$L \rightarrow n$ $n+1 \rightarrow L$	—	—	Skok ze śladem: zawartość licznika rozkazów L prześlij pod adres n, do licznika rozkazów L prześlij zaś $n+1$.
5	05	stop n	—	—	Stop z pobraniem: pobierz rozkaz według wskazań części adresowej do rejestru dyspozytora D, po czym zatrzymaj maszynę. Po ponowym uruchomieniu w zależności od manipulacji ręcznej albo zostaje wykonany rozkaz znajdujący się w rejestrze D, po którym maszyna pobierze kolejny rozkaz według wskazań licznika rozkazów L, albo maszyna nie pobierze kolejnego rozkazu według wskazań licznika rozkazów L (patrz pulpit sterowania).
6	06	$n \rightarrow L$ $W = 0$	—	—	Drugi skok warunkowy: jeśli zawartość rejestru $W = 0$, to część adresową wykonywanego rozkazu prześlij do licznika rozkazów L; jeśli zaś zawartość rejestru $W = 1$, to nie zmieniaj zawartość licznika rozkazów L, Uwaga: Rozkaz drugiego skoku warunkowego zeruje rejestr W.

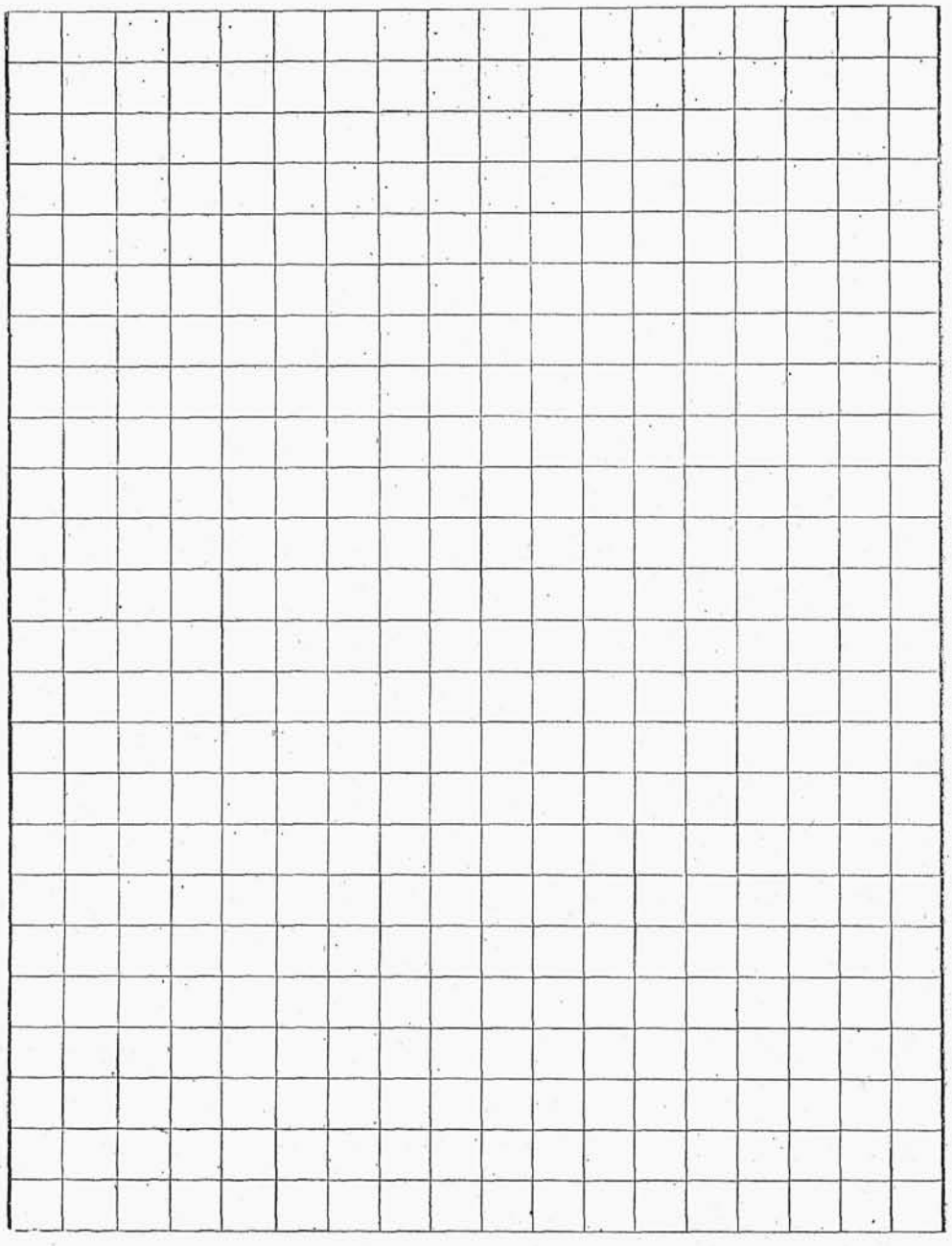
8	10	$(A)+(n)\rightarrow A$	$(A)<0$	$(A)\geqslant 1$ albo $(A)<-1$	Dodawanie: 1) gdy n-adres długi, do 34 bardziej znaczących pozycji akumulatora dodaj zawartość adresu n; 2) gdy n-adres krótki, do 17 najbardziej znaczących pozycji akumulatora dodaj zawartość adresu n. Jeśli wynik jest liczbą ujemną, to generuj sygnał $W=1$. Jeśli zaś nastąpiło przekroczenie zakresu, to generuj sygnał $N=1$.
9	11	$(A)-(n)\rightarrow A$	$(A)<0$	$(A)\geqslant 1$ albo $(A)<-1$	Odejmowanie: 1) gdy n-adres długi, od 34 bardziej znaczących pozycji akumulatora odejmij zawartość adresu n; 2) gdy n-adres krótki, od 17 najbardziej znaczących pozycji akumulatora odejmij zawartość adresu n. Jeśli wynik jest liczbą ujemną, to generuj sygnał $W=1$. Jeśli nastąpiło przekroczenie zakresu, to generuj sygnał $N=1$.
10	12	$(n)\rightarrow A$	$(A)\neq 0$	—	Plus przesłanie: 1) gdy n-adres długi, prześlij zawartość adresu n do 34 bardziej znaczących pozycji akumulatora, 2) gdy n-adres krótki — prześlij zawartość adresu n do 17 najbardziej znaczących pozycji akumulatora. Jeśli liczba przesłana do akumulatora jest różna od zera, to generuj sygnał $W=1$.
11	13	$-(n)\rightarrow A$	$(A)\neq 0$	$(A)=1$	Minus przesyłanie: 1) gdy n-adres długi, prześlij uzupełnienie zawartości adresu n do 34 bardziej znaczących pozycji akumulatora, 2) gdy n-adres krótki, prześlij uzupełnienie zawartości adresu n do 17 najbardziej znaczących pozycji akumulatora. Jeśli liczba przesłana do akumulatora jest różna od zera, to generuj sygnał $W=1$. Jeśli nastąpiło przekroczenie, to generuj $N=1$
12	14	$(A)\rightarrow n$	$(A)<0$	—	Przesyłanie do pamięci: 1) gdy n-adres długi, prześlij zawartość 34 bardziej znaczących pozycji akumulatora do komórki pamięci o adresie n, 2) gdy n adres krótki, prześlij 17 najbardziej znaczących pozycji akumulatora do komórki pamięci o adresie n. Jeśli zawartość akumulatora jest ujemna, to generuj sygnał $W=1$. Uwaga: W wyniku operacji przesyłanie do pamięci zawartości akumulatora nie ulega zmianie.
13	15	$z(A)\rightarrow A$	$(A)<0$	$(A)\geqslant 1$ albo $(A)<-1$	Zaokrąglanie: W zależności od znaku zawartości akumulatora dodaj lub odejmij $z=2^{-34}$. Jeśli zawartość akumulatora ujemna, to generuj sygnał $W=1$. Jeśli w wyniku zaokrąglania nastąpiło przekroczenie zakresu, to generuj sygnał $N=1$.

Nr rozkazu	Kod ósemkowy rozkazu	Zapis symboliczny rozkazu	Warunek generacji sygnału $W = 1$	Warunek generacji sygnału $N = 1$	Nazwa i szczegółowy opis rozkazu
1	2	3	4	5	6
14	16	$(M) \cdot (n) \rightarrow A$	$(A) < 0$	$(A) = 1$	Mnożenie: pomnóż zawartość adresu n przez zawartość rejestru mnożnej M; i wynik umieść w akumulatorze. Jeśli wynik jest liczbą ujemną, to generuj sygnał $W = 1$. Uwaga: w wyniku operacji mnożenia zawartość rejestru M nie ulega zmianie.
15	17	$(A):(n) \rightarrow A_L$ reszta $\rightarrow A_p$	$(A) < 0$	$\|(A)\| \geqslant \|(n)\|$	Dzielenie: zawartość akumulatora podziel przez zawartość adresu n, umieszczając iloraz na 34 bardziej znaczących pozycjach akumulatora, zaś resztę umieszczając na 34 mniej znaczących pozycjach akumulatora. Jeśli wynik jest liczbą ujemną, to generuj sygnał $W = 1$. Jeśli $\|(A)\| \geqslant \|(n)\|$, to generuj sygnał $N = 1$, przy czym dzielenie nie zostaje wykonane. Uwaga: w przypadku gdy maszyna nie wykonuje dzielenia, przechodzi ona do następnego taktu pracy.
16	20	$(A) \cdot 2^n \rightarrow A$	$(A) < 0$	$(A) \geqslant 1$ albo $(A) < -1$	Mnożenie przez 2^n: zawartość podwójnie długiego akumulatora pomnóż przez 2^n. Liczba n nie może być większa od 63 (adres mod 64); jeśli wynik jest liczbą ujemną, to generuj sygnał $W = 1$. Jeżeli nastąpiło przekroczenie zakresu, to generuj sygnał $N = 1$.
17	21	$(A) : 2^n \rightarrow A$	$(A) < 0$	—	Dzielenie przez 2^n: zawartość podwójnie długiego akumulatora pomnóż przez 2^{-n}. Liczba n nie może być większa od 63 (adres mod 64). Jeśli wynik jest liczbą ujemną, to generuj sygnał $W = 1$.
18	22	$(A) : 2_n \rightarrow A$	$(A) < 0$	—	Przesuwanie w prawo: zawartość podwójnie długiego akumulatora przesuń o n miejsc binarnych cyklicznie w prawo. Liczba n nie może być większa od 63 (adres mod 64). Jeśli wynik jest liczbą ujemną, to generuj sygnał $W = 1$.
19	23	$(A) \cdot 2_n \rightarrow A$	$(A) < 0$	—	Przesuwanie w lewo: zawartość podwójnie długiego akumulatora przesuń o n miejsc binarnych cyklicznie w lewo. Liczba n nie może być większa od 63 (adres mod 64). Jeśli wynik jest liczbą ujemną, to generuj sygnał $W = 1$.

20	24	czytaj	$Y = 1$	—	Czytanie: odczytaj kolejny rządek taśmy perforowanej według wskazań przełączników blokada wejścia (patrz pulpit sterowania) i wynik zsumuj logicznie z zawartością miejsca charakterystycznego akumulatora przesuwając jednocześnie cyklicznie poprzednią zawartość podwójnie długiego akumulatora o pięć pozycji binarnych w prawo. Jeśli na którymś z uwarunkowanych miejsc za pomocą przełączników warunek wyjścia Y, (patrz pulpit sterowania) pojawi się jedynka, to generuj sygnał $W = 1$.
21	25	drukuj	$Z = 1$	—	Drukowanie: wydrukuj znak odpowiadający zawartości miejsca charakterystycznego akumulatora według wskazań przełączników blokada wyjścia (patrz pulpit sterowania). Jeśli na którymś z uwarunkowanych miejsc za pomocą przełączników warunek wyjścia Z, (patrz pulpit sterowania) pojawi się jedynka, to generuj sygnał $W = 1$.
22	26	$\|(A)\| \rightarrow A$	$(A) \neq 0$	$(A) = 1$	Tworzenie wartości bezwzględnej: zastąp zawartość akumulatora bezwzględną wartością akumulatora. Jeśli liczba zawarta w akumulatorze jest różna od zera, to generuj sygnał $W = 1$. Jeśli nastąpiło przekroczenie zakresu, to generuj sygnał $N = 1$.
23	27	$(n) \rightarrow M$	$(M) < 0$	—	Przesyłanie do M: 1) gdy n-adres długi prześlij zawartość adresu n do rejestru mnożnej M; 2) gdy n adres krótki, prześlij zawartość adresu n do 17 bardziej znaczących pozycji rejestru mnożnej M. Jeśli liczba przesyłana do rejestru mnożnej M jest ujemna, to generuj sygnał $W = 1$.
24	30	$(A) \cap (n) \rightarrow A$	$(A) \neq 0$	—	Koniunkcja: weź koniunkcję zawartości akumulatora i zawartości adresu n. Uwaga: Jeśli adres n jest adresem krótkim, to maszyna traktuje to jakby na 17 mniej znaczących pozycjach zawartość adresu długiego były dane zera. Jeśli wynik jest liczbą różną od zera, to generuj sygnał $W = 1$.
31	37	$(L) \rightarrow n$ $n + 1 \rightarrow L$ $N = 1$	—	—	Skok ze śladem przy nadmiarze: Jeśli zawartość rejestru $N = 1$, to zawartość licznika rozkazów L prześlij pod adres n, do licznika rozkazów L prześlij zaś $n+1$; jeśli zaś zawartość rejestru $N = 0$, to nie przesyłaj zawartości licznika rozkazów L pod adresem n oraz nie zmieniaj zawartości licznika rozkazów. Uwaga: Rozkaz skoku ze śladem przy nadmiarze zeruje rejestr N.

Uwaga: „—" w rubryce generacji sygnału oznacza, że wykonywany rozkaz nie wpływa na stan rejestru (W lub N).

Załącznik 4

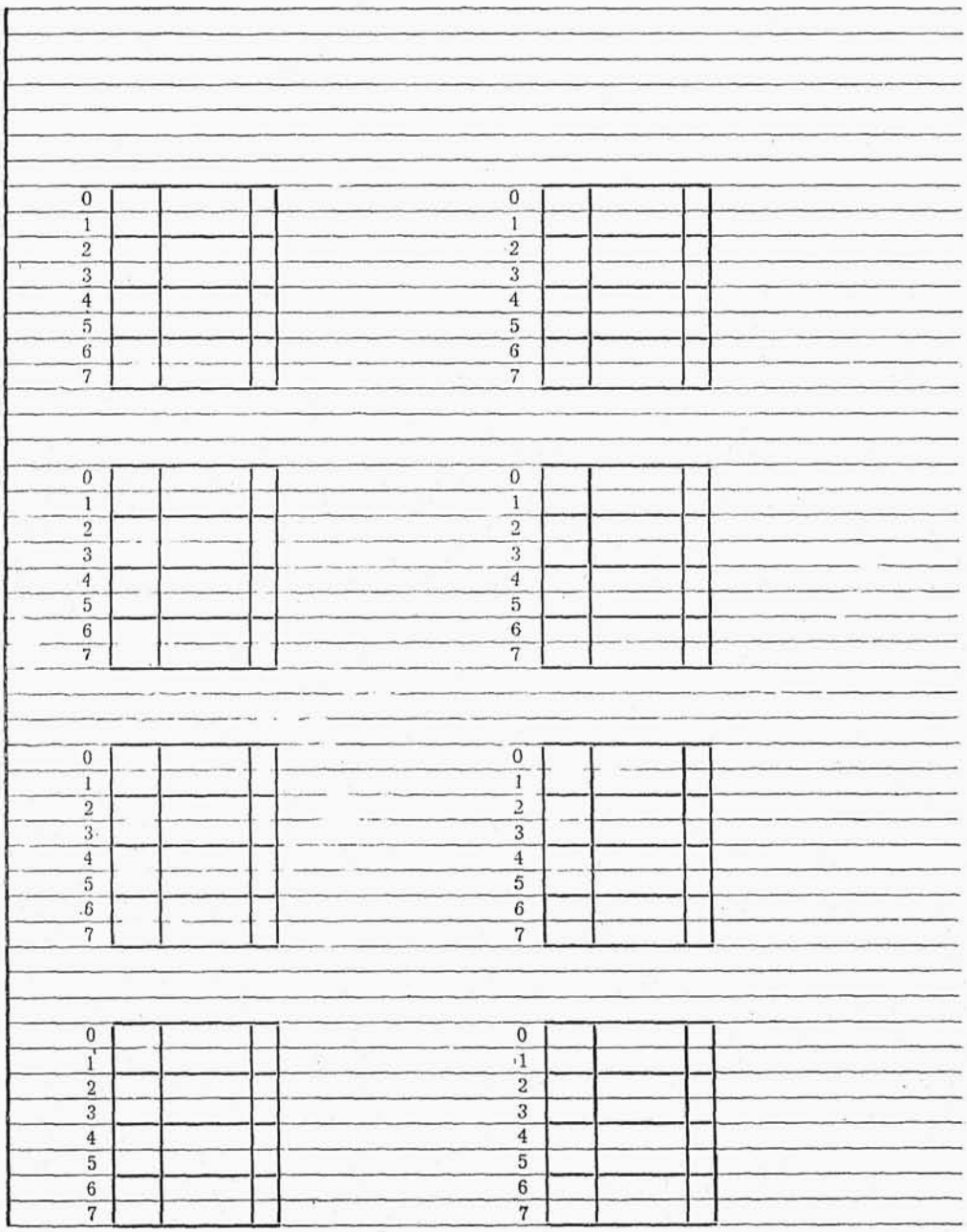


Załącznik 4

ARKUSZ PROGRAMOWY EM

Dn. .....

str.



Państwowe
Wydawnictwo Naukowe

*

Wydanie I. Nakład 2000+200 egz. Ark. wyd. 16,5; ark. druk. 12,25+1 wkł. Papier ilustracyjny kl. V, 70 g, 70×100. Oddano do składania 26 XI 1960 r. Podpisano do druku 10 X 1961 r. Druk ukończono w październiku 1961 r. Zam. nr. 4314/12.60 E-3 Cena zł 36,—

*

Zakł. Graf. im. M. Kasprzaka
w Poznaniu