C

PROGRAMOWANIE EMC

Układanie algorytmów postępowania dla EMC nosi nazwę programowania[1]. Jeśli układamy program jako sekwencję rozkazów EMC, to takie programowanie nazywamy programowaniem w kodzie wewnętrznym EMC. Jeśli natomiast układamy program, pisząc go jako ciąg wyrażeń jakiegoś języka sformalizowanego (autokodu), przy czym istnieje specjalny program, tzw. translator (*compiler*), który tłumaczy wyrażenia tego języka sformalizowanego na sekwencję rozkazów EMC, to takie programowanie nazywamy programowaniem w języku zewnętrznym lub w autokodzie. Na ogół proces programowania składa się z pięciu podstawowych etapów:

a) rysowanie schematu blokowego (*flow diagram*) programu, będącego graficznym obrazem dróg sterowania programu i kryteriów wyboru tych dróg;

b) napisanie na podstawie wcześniej narysowanego schematu programu jako sekwencji wyrażeń wybranego autokodu lub kodu wewnętrznego EMC (*coding*);

c) tłumaczenie programu, napisanego w autokodzie na kod wewnętrzny EMC (*compiling*) (dotyczy to jedynie programów napisanych w autokodzie);

d) sprawdzanie programu na EMC ze sztucznymi danymi lub uproszczonymi danymi rzeczywistymi (*testing*);

---

[1] Oczywiście, proces programowania musi być poprzedzony sformułowaniem zadania i wyborem metody rozwiązania.

e) próbna eksploatacja i ulepszanie programu z danymi rzeczywistymi (*debuging*).

W naszych rozważaniach ograniczymy się do omówienia dwu pierwszych etapów: a) i b). W dalszym ciągu będziemy rozróżniali trzy rodzaje programów: programy użytkowe (*program*), programy typowe (*routine*) i podprogramy (*subroutine*).

Celem schematów blokowych jest przedstawienie graficzne zasadniczych czynności programu. Warto podkreślić, że schematy blokowe mają dwojaką przydatność:

a) ułatwiają kodowanie problemu,

b) umożliwiają łatwe zorientowanie się w strukturze programu.

Program napisany w autokodzie jest wprawdzie dużo bardziej czytelny od programu napisanego w kodzie EMC, ale bez pomocy schematu blokowego jest trudny do zrozumienia dla osoby postronnej. Schematy blokowe wymagają pewnych konwencji dotyczących zasady podziału programu na prostsze części. Na ogół przyjmuje się następujące zasady podziału:

a. Elementy podziału muszą dawać możliwość ich niezależnego napisania, przy założeniu, że operują one nazwami lub adresami danych wspólnymi w obrębie całego programu.

b. Podział musi uwzględniać specyfikę obliczeń automatycznych, tzn. musi odróżniać obliczanie wartości wyrażeń arytmetycznych od operacji logicznych na danych, operacji przenoszenia informacji z jednego obszaru pamięci (operacyjnej) do drugiego (czy też do albo z pamięci zewnętrznej, czy urządzeń wejścia do urządzeń wyjścia) czy wreszcie od pomocniczych operacji organizacyjnych programu.

c. Podział musi uwzględniać wszystkie dopuszczalne w programie kolejności wykonywania sekwencji rozkazów lub wyrażeń autokodowych oraz kryteria wyboru tych kolejności.

Wszystkie czynności, które mogą wchodzić w skład programu, podzielimy na dwie klasy: testy i operatory. Testy są to czynności decydujące o wyborze drogi w programie, operatory zaś to wszystkie inne czynności. Zarówno testom, jak i operatorom odpowiadają sekwencje rozkazów EMC. Wprowadzimy z kolei dwa pomocnicze skróty.

1. Będziemy mówili krótko: wejście testu (operatora), zamiast mówić: pierwsze wyrażenie autokodowe albo: pierwszy rozkaz testu (operatora).

2. Będziemy mówili krótko: wyjście testu (operatora), zamiast mówić: wyrażenie autokodowe albo: rozkaz przekazania sterowania z testu (operatora) na zewnątrz.

Na to aby testy i operatory spełniały zasady a÷c, powinny być spełnione następujące warunki:

a. Warunek uporządkowania testu. Sterowanie z zewnątrz do testu może być przekazane jedynie przez wejście testu. Test musi mieć co najmniej dwa wyjścia.

b. Warunek uporządkowania operatora. Sterowanie z zewnątrz do operatora może być przekazane jedynie przez wejście operatora. Operator może mieć tylko jedno wyjście.

c. Warunek prostoty operatora. Operator realizuje tylko jeden rodzaj czynności:

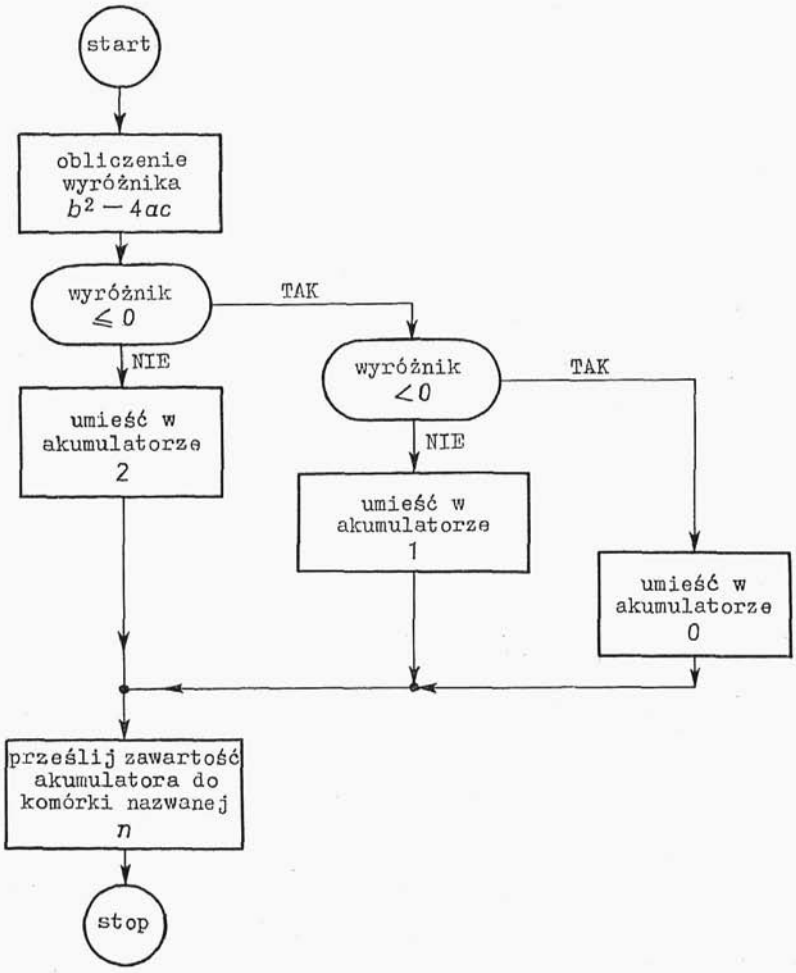
Graficznie przedstawiamy testy jak na rys. 38a, operatory zaś jak na rys. 38b. Każdy program zbudowany jest z pewnej ilości operatorów i testów odpowiednio między sobą połączonych. Dla zmniejszenia ilości połączeń rysowanych pomiędzy poszczególnymi operatorami i testami wprowadzamy specjalne oznaczenia początków sekwencji czynności tzw. etykiet (patrz rys. 38c) i oznaczenia końców sekwencji (patrz rys. 38d) wskazujących, od jakiej etykiety począwszy zaczyna się następna czynność programu.

Omówimy z kolei podstawowe typy sieci czynności, które występują praktycznie w każdym programie.

Sieć liniowa. Programy o sieciach liniowych mają najprostszą z możliwych struktur logicznych. W programach tych występują jedynie operatory, np. s_1, s_2, ..., s_n, przy czym wyjście pierwszego operatora łączy się z wejściem drugiego operatora, wyjście drugiego łączy się z wejściem trzeciego operatora itd. Każdy z operatorów jest wykonywany tylko jeden raz. Zaletą programów o sieciach liniowych jest prędkość ich wykonywania przez EMC, wadą zaś ilość miejsca, jaką zajmują one w pamięci. Wyobraźmy sobie program o sieci liniowej złożony z 1000 rozkazów; załóżmy dalej, że EMC pracuje z szybkością 100 000 operacji na sekundę, wówczas cały program zostałby wykonany przez EMC w ciągu 10 ms. Na to więc, aby zapewnić pracę EMC na podstawie programu z sieci liniowej przez 1 minutę, potrzebna jest pamięć operacyjna o pojemności 6 000 000 rozkazów, a takich pamięci operacyjnych jeszcze się nie pro-

dukuje. Programy o sieciach liniowych są więc rozwiązaniem, którego należy raczej unikać przy programowaniu.

Sieci z rozwidleniami. Problemy, w których w różnych przedziałach rozwiązania są dane za pomocą różnych formuł, rozwiązujemy za pomocą programu o sieci z rozwidleniami. Program taki zawiera



Rys. 50. Przykład schematu blokowego programu o sieci z rozwidleniem

test (lub testy) sprawdzający warunki, sekwencje operatorów zwane wariantami i wreszcie operator (lub sekwencje operatorów) wykonujący końcowe obliczenia. Sekwencje operatorów i testów nazywamy wariantem, który może być lub nie być wykonywany przy jednorazowym użyciu programu. Rozpatrzymy obecnie elementarny przykład, prowadzący do programu o sieci z rozwidleniami.

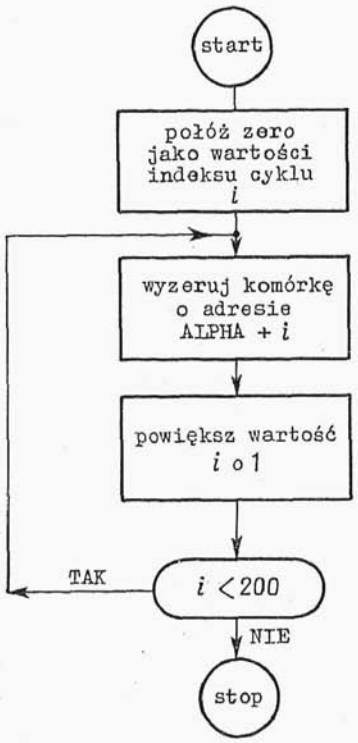
Przypuśćmy, że dla pewnych celów potrzebny nam jest program obliczający ilość pierwiastków rzeczywistych algebraicznego równania kwadratowego

$$ax^2+bx+c=0.$$

Oznaczmy ilość pierwiastków rzeczywistych równania kwadratowego przez n (oczywiście, n należy do zbioru $\{0, 1, 2\}$). Program będzie zawierał jeden operator arytmetyczny do obliczania wyróżnika równania kwadratowego b^2-4ac, test badający, czy wyróżnik jest mniejszy lub równy 0, z którego program rozwidla się w dwa warianty. W wariancie odpowiadającym wartości wyróżnika większej od 0 będzie znajdował się jeden operator umieszczania liczby 2 w akumulatorze. W wariancie odpowiadającym wartości wyróżnika mniejszej lub równej 0 będzie znajdował się test badający, czy wyróżnik jest mniejszy od 0, z którego program ponownie rozwidli się w dwa warianty. W wariancie odpowiadającym wartości wyróżnika równej 0 będzie znajdował się jeden operator umieszczający liczbę 1 w akumulatorze. W wariancie odpowiadającym wartości wyróżnika mniejszym od 0 będzie znajdował się jeden operator umieszczający liczbę 0 w akumulatorze. W końcowej części programu, po ponownym połączeniu się wariantów, będzie znajdował się jeden operator przesłania zawartości akumulatora do komórki pamięci operacyjnej nazwanej n. Na rysunku 50 pokazany jest schemat blokowy omawianego przykładu.

Sieci z cyklem. Jeśli chcielibyśmy wyzerować zawartość 200 kolejnych komórek pamięci operacyjnej począwszy od adresu nazwanego ALPHA za pomocą programu o sieci liniowej, to program taki składałby się z 200 rozkazów. Mimo szybkości działania zajmowałby on zbyt dużo miejsca w pamięci operacyjnej. Dużo oszczędniej można uzyskać ten sam wynik używając programu o sieci z cyklem. Program taki składałby się z operatora ustawiającego wartość początkową tzw. indeksu cyklu, ope-

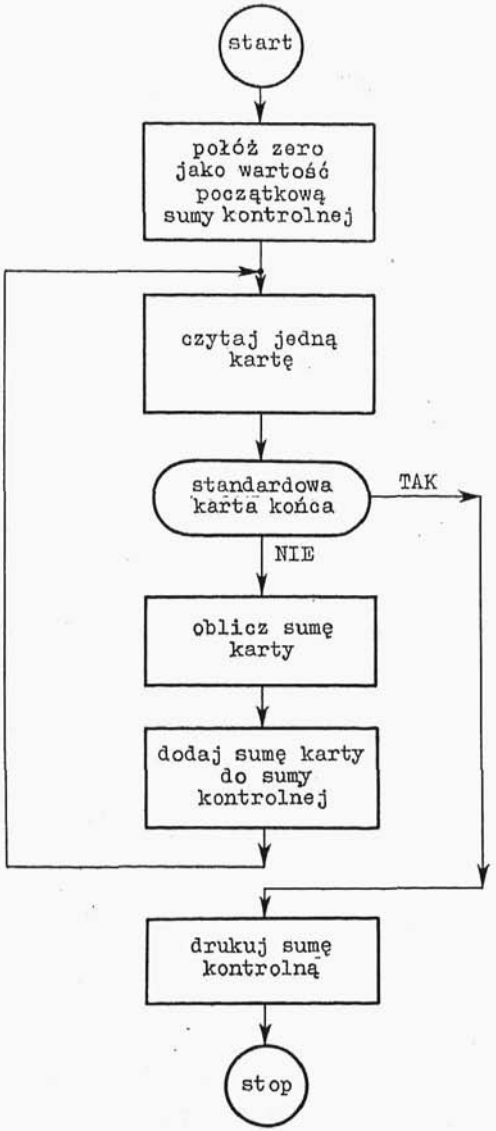
ratora zerowania kolejnej komórki pamięci, operatora zwiększającego indeks cyklu o 1, testu badającego czy wartość indeksu jest mniejsza od 200. Jeśli indeks jest mniejszy od 200, to nastąpi powtórne wykonanie dwu ostatnich operatorów. W przypadku gdy indeks równa się 200, następuje przejście do dalszych czynności. Na rysunku 51 pokazany jest schemat blokowy omawianego przykładu.



Rys. 51. Przykład schematu blokowego programu o sieci z cyklem

Podany przykład mimo swej prostoty dobrze ilustruje zasadę działania programu o sieci z cyklem. Dla cyklu istotne jest podanie krotności. Na to aby dany algorytm dawał się sprowadzić do programu o sieci z cyklem, konieczne jest istnienie tzw. postaci rekurencyjnej algorytmu.

Sieci z iteracją. Istotną cechą programu o sieci z cyklem była z góry określona krotność powtarzania fragmentu programu. Iteracja, w odróżnieniu od cyklu, nie ma z góry określonej krotności powtarzania. W programach przetwarzania danych krotność powtarzania jest określona ilością przetwarzanych dokumentów, przy czym ilość dokumentów wchodzących w skład danego zbioru nie może być na ogół wcześniej określona. Dlatego też za ostatnim dokumentem należącym do zbioru umieszcza się: w przypadku zbioru na kartach perforowanych tzw. standardową kartę końca, w przypadku zbiorów przechowywanych na taśmach magnetycznych tzw. etykietę końca zbioru.



Rys. 52. Przykład schematu blokowego programu o sieci z iteracją

Na rysunku 52 pokazany jest przykład schematu blokowego programu o sieci z iteracją. Program ten służy do obliczania sumy zawartości określonych grup kolumn kart perforowanych, tzw. sumy kontrolnej. W przypadku napotkania standardowej karty końca następuje przerwanie iteracji i wydrukowanie obliczonej sumy kontrolnej.

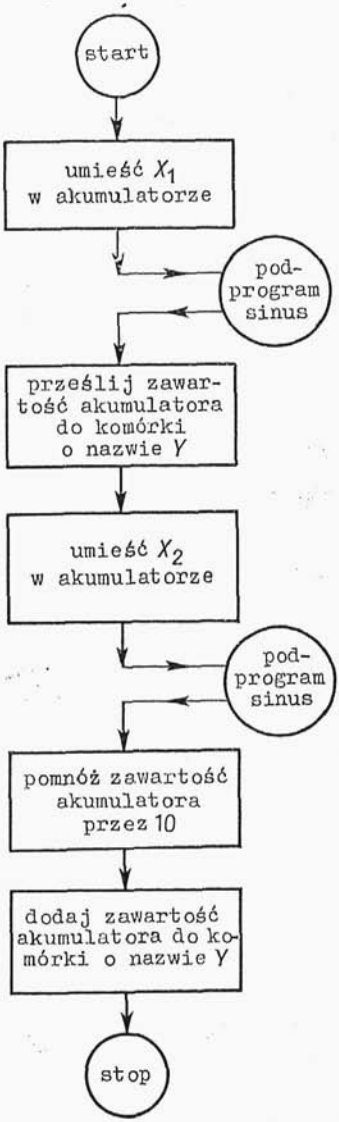
Warto podkreślić, że tzw. iteracje numeryczne prowadzą również do programów o sieci z iteracją.

Podprogramy. Przez podprogramy rozumiemy zwykle zespół operatorów i testów przeznaczony do wykonywania zamkniętego fragmentu obliczeń arytmetycznych, przetwarzania lub czynności organizacyjnych (gdzie przez zamknięty fragment rozumiemy wykonywanie określonego algorytmu dla zadanych każdorazowo wartości parametrów). Na początku tego zespołu operatorów i testów znajduje się operator przechowania śladu czy adresu, pod którym znajduje się pierwszy rozkaz programu (wywołującego dany podprogram), który ma być wykonany bezpośrednio po zakończeniu wykonywania podprogramu. Na końcu podprogramu znajduje się rozkaz powodujący przekazanie sterowania począwszy od adresu (śladu).

Na rysunku 53 pokazany jest schemat blokowy programu korzystającego z podprogramu obliczającego wartość funkcji sinus dla zadanego argumentu umieszczonego w akumulatorze i umieszczającego wynik również w akumulatorze.

Opis danych. Informacje będące podmiotem działania programu nazywamy danymi. Przy prowadzeniu obliczeń numerycznych mamy do czynienia zwykle z trzema rodzajami danych: tak zwanymi zmiennymi całkowitoliczbowymi, zmiennymi rzeczywistymi oraz zmiennymi logicznymi (zero-jedynkowymi). Dlatego też w autokodach dla celów obliczeń naukowych i technicznych sprawa opisu danych potraktowana jest marginesowo. W przetwarzaniu danych mamy odmienną sytuację. Występują tam poza danymi numerycznymi również dane alfanumeryczne (złożone z kodów — binarnych znaków alfanumerycznych) lub czysto alfabetyczne. Poza tym dane te mogą mieć różne długości, tzn. składać się z różnej ilości znaków. Dotyczy to również danych numerycznych. Obok różnych ilości znaków, dane numeryczne mogą mieć punkt dziesiętny w różnych położeniach.

Jeśli w całym procesie obliczeniowym (przetwarzania) punkt dziesiętny znajduje się stale w tym samym położeniu (np. między siódmą a ósmą cyfrą liczby), to dane takie nazywamy stałoprzecinkowymi (*fixed point*). Szczególnym przypadkiem są dane całkowitoliczbowe, dla których punkt dziesiętny znajduje się zawsze za najmniej znaczącą cyfrą.



Jeśli w całym procesie obliczeniowym (przetwarzania) jest określona stale jedynie ilość cyfr danej, natomiast położenie punktu dziesiętnego określa specjalna pomocnicza dana, tzw. wykładnik (*exponent*), to dane takie nazywamy zmiennoprzecinkowymi (*floating point*). Przy obliczeniach numerycznych dane zmiennoprzecinkowe są nazywane zmiennymi lub liczbami rzeczywistymi.

Uzupełniając powyższe rozważania należy podkreślić, że obok omawianych typów danych, w procesach przetwarzania danych mamy często do czynienia z liczbami stałoprzecinkowymi dodatnimi o różnych ilościach znaków.

Jak widać z powyższego, problem opisu danych w programach przetwarzania odgrywa poważną rolę. Autokody, przeznaczone do programowania zadań prze-

Rys. 53. Przykład schematu blokowego programu korzystającego z podprogramu sinusa do obliczania wartości wyrażenia $y = \sin x_1 + 10 \sin x_2$

twarzania danych, mają zwykle specjalne wyrażenia przeznaczone do opisywania danych, tworzących w przypadku języka Cobol[1] specjalny rozdział programu, tzw. *Data Division*. Natomiast wyrażenia proceduralne programu napisanego w autokodzie do przetwarzania danych, w czasie tłumaczenia przez EMC (przy użyciu specjalnego programu, tzw. translatora) na kod EMC, są zastępowane odpowiednimi sekwencjami rozkazów opisanymi przez te wyrażenia proceduralne i opis danych, na których mają być wykazywane działania.

---

[1] Język Cobol (skrót od słów *Common Business Oriented Language*) został opracowany w r. 1959 w USA, przez specjalny komitet, w którego skład wchodzili przedstawiciele producentów i użytkowników EMC. W wyniku zdobytych doświadczeń eksploatacyjnych w następnych latach Cobol został nieco zmodyfikowany. Podstawowe informacje na temat zastosowania Cobolu do opisu danych może czytelnik znaleźć w pracy [48].