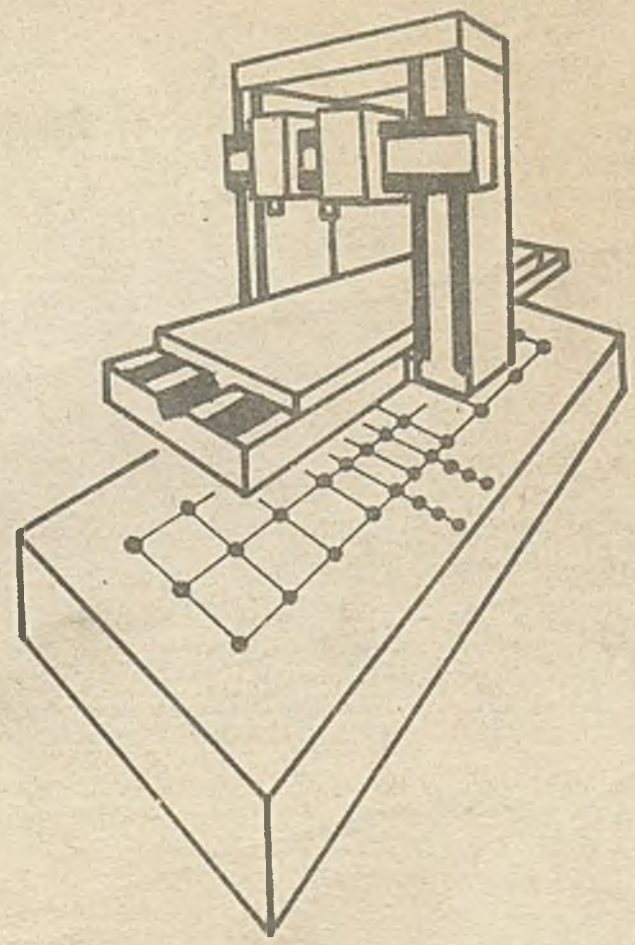
biuletyn informacyjny

3 '79

MERA IMM OBIEKTOWE SYSTEMY KOMPUTEROWE



Zjednoczenie Przemysłu Automatyki i Aparatury Pomiarowej „MERA"
Instytut Maszyn Matematycznych „MERA IMM" Branżowy Ośrodek INTE

Na okładce: Przykład obrabiarki sterowanej numerycznie. Rysunek zaczerpnięto z CAD-Berichte. The CAD Support Project Status, July 1978, s.75

Druk okładki IMM zam. 64/80 nakł. 820

D W U M I E S I Ę C Z N I K

Wydaje:

CENTRUM NAUKOWO-PRODUKCYJNE TECHNIK KOMPUTEROWYCH I POMIARÓW

I N S T Y T U T M A S Z Y N M A T E M A T Y C Z N Y C H

Branżowy Ośrodek Informacji Naukowej Technicznej i Ekonomicznej

Adres redakcji: ul. Krzywickiego 34, 02-078 Warszawa
tel. 28-37-29 lub 21-84-41 w.244

COPS

SYSTEM PRODUKCJI TRANSLATORÓW

Za pomocą Systemu COPS (COMPILER PRODUCTION SYSTEM) można:

i automatycznie generować translatory języków programowania,

ii usprawnić i obniżyć koszty wytwarzania tzw. obiektowych systemów minikomputerowych,

iii wspomagać proces projektowania nowych języków programowania oraz natychmiast weryfikować kolejne wersje takich języków,

iv dogłębnie poznać technologię implementacji języków programowania, a w szczególności - zastosowania gramatyk atrybutowych.

System pracuje na maszynach IBM/360-370 oraz JS pod kontrolą systemu operacyjnego OS.

W skład Systemu COPS wchodzą:

i środki opisowe (metajęzyki), za pomocą których zapisuje się formalną definicję języka źródłowego,

ii zestaw programów (metatranslator), który - dysponując formalną definicją języka źródłowego - tworzy translator dla tego języka.

Zaletą stosowania systemu jest istotne zmniejszenie pracochłonności wytwarzania translatora (od 50% - w zależności od oprogramowania dostępnego w systemie operacyjnym maszyny docelowej). Przyczynia się to tego m.in. ustalona technologia oraz dobre udokumentowanie pracy nad translatorem.

COPS nie jest odpowiednim narzędziem do produkcji assemblerów i mikroassemblerów, ze względu na ich złożoną strukturę leksykalną.

Systemu nie można stosować do produkcji translatorów rezydentnych (poza maszynami JS pod kontrolą OS) oraz translatorów pracujących w reżimie konwersacyjnym.

Kod roboczy produkowany przez translator wygenerowany automatycznie przez COPS nie różni się pod względem czasu wykonania czy zajętości pamięci od kodu produkowanego przez translator wytworzony tradycyjnie. Natomiast czas translacji oraz wymagania pamięciowe są w przypadku translatora otrzymanego automatycznie większe. Na przykład, translator krzyżowy (R32 → MERA400) języka COBOL wyprodukowany za pomocą systemu COPS wymaga 256 K pamięci operacyjnej dla programów średniej wielkości. Zaawansowane są jednak prace nad nową, udoskonaloną pod tym względem wersją Systemu.

Bardziej szczegółowe informacje o Systemie COPS i możliwościach jego zastosowań można znaleźć w tekście "System Produkcji Translatorów", który oferujemy.

JAN BOROWIEC

INSTYTUT MASZYN MATEMATYCZNYCH

UL. KRZYWICKIEGO 34

02-078 WARSZAWA

Wasz System interesuje mnie i proszę o:

☐ opis systemu COPS pn. "SYSTEM PRODUKCJI TRANSLATORÓW" /str.150/

☐ skrócony opis systemu pn. "SYSTEM PRODUKCJI TRANSLATORÓW - WPROWADZENIE" /str.35/

☐ informacje na temat możliwości zainstalowania systemu w moim ośrodku /krótka charakterystyka bazy sprzętowej: .....
.....
...../

☐ ocenę przydatności Systemu do realizacji moich zadań w zakresie oprogramowania m.c. /krótka charakterystyka: .....
.....
.....
...../

Nazwisko i imię:.....

Adres: .....
.....
.....

Biuletyn Informacyjny OBIEKTOWE SYSTEMY KOMPUTEROWE

Rok XVII Nr 3 1979

Spis treści

mgr inż. Zbigniew Poznański – Symulacja komputerowa w biologii i medycynie ..... s. 3

mgr inż. Zbigniew Poznański – Simula 67 – uniwersalny język programowania ..... s. 9

mgr inż. Halina Gutowska – Problemy przygotowywania programów obróbki części oraz właściwej eksploatacji systemu ... s. 27

mgr inż. Jan Wrona – Próba przedstawienia wieloprocesorowej wewnętrznej organizacji wyspecjalizowanego minikomputera sterującego systemami wytwarzania ..... s. 39

dr inż. Tomasz Jamson, doc.dr hab.inż. Zygmunt Zawisławski – Niektóre zagadnienia prędkości wydruku optoelektronicznych drukarek kserograficznych .... s. 51

z cyklu: Komputeryzacja projektowania inżynierskiego

mgr inż. Jerzy Mocała – Automatyzacja procesu projektowania. Program KATLG dla mc MERA 400 ..... s. 57

Ankieta ..... s. 67

Contents

Z.Poznański – Computer simulation in biology and medicine .. .. p. 3

Z.Poznański – Simula 67 – general purpose programming language ..... p. 9

H.Gutowska – Problems of preparing part treatment programs and appropriate APT system exploitation ..... p. 27

J.Wrona – Multiprocessor internal organization of the special purpose minicomputer for production control ..... p. 39

T.Jamson, Z.Zawisławski – Some printing frequency problems of optoelectronic xerographic printers ..... p. 51

From: Computerization of engineering design

J.Mocała – Design process automation. KATLG program for MERA 400 computer ..... p. 57

Содержание

З. Познаньски – Компьютерная имитация в биологии и медицине . . . . . с. 3

З. Познаньски – Simula 67 – универсальный язык программирования . . . . . с. 9

Х. Гутовска – Проблемы подготовки программ обработки деталей и соответствующей эксплуатации системы APT. с.27

Й. Врона – Попытка представления многопроцессорной внутренней организации специализированной мини-ЭВМ, управляющей производственными системами . . . с.39

Т. Янсон, З. Завиславски – Некоторые вопросы скорости печати оптоэлектронных ксерографических печатающих устройств . . с.51

Из цикла:

Компьютеризация инженерного проектирования

Й. Моцала – Автоматизация процесса проектирования. Программа KATLG для ЭВМ MERA 400 . .с.57

mgr inż. Zbigniew POZNAŃSKI
Instytut Technologii Elektronowej

Symulacja komputerowa w biologii i medycynie

Metody analityczne badania złożonych procesów można z powodzeniem stosować tylko w bardzo prostych przypadkach, rzadko spotykanych w rzeczywistości. Metody eksperymentalne, w których potrzebne wyniki uzyskuje się z obserwacji procesów rzeczywistych, są uciążliwe i kosztowne. Aby prześledzić zachowanie się układu należy zwykle budować model eksperymentalny lub prototyp. Znane są i często stosowane metody symulacji analogowej - wykorzystuje się je do przybliżonego rozwiązywania równań różniczkowych i całkowych oraz badania układów regulacji. Problemy te można rozwiązywać także numerycznie na maszynach cyfrowych.

Często spotykamy się jednak z problemami, których nie potrafimy rozwiązać ani analitycznie ani numerycznie na EMC lub których rozwiązanie jest czasochłonne. Pojawiają się również duże trudności już na etapie budowy modelu opartej na pojęciach analizy matematycznej. Do rozwiązania tej klasy zagadnień pomocną wydaje się być symulacja cyfrowa w odpowiednio zdefiniowanym języku. Budowa modelu w takich językach programowania, jak ALGOL czy FORTRAN, nie jest w takich wypadkach wskazana, gdyż języki te służą jedynie do sekwencyjnych obliczeń numerycznych. Nadają się do tego znacznie lepiej języki ukierunkowane problemowo, wyposażone w mechanizmy umożliwiające działanie na złożonych strukturach listowych oraz symulację procesów współbieżnych. Jednym z takich języków jest SIMULA-67. Język ten stanowi znaczne rozszerzenie języka ALGOL. SIMULA to właściwie język opisu procesów i systemów, zbliżony do języka naturalnego. Cechy obiektów oraz algorytmy ich działania, zapisane w języku SIMULA, stanowią jednocześnie program dla maszyny cyfrowej. Wynika stąd, że dowolnie złożone problemy sformułowane w terminach tego języka mogą być rozwiązywane bezpośrednio przez maszynę cyfrową.

Podstawowym pojęciem języka SIMULA jest pojęcie "klasy". Można we własnym zakresie definiować nowe klasy, w których opisuje się atrybuty i algorytmy działania obiektów należących do tych klas.

Systemowo zdefiniowano tu klasę Simset, do operacji na strukturach listowych (kolejkach) oraz klasę Simulation. Ta ostatnia zapewnia możliwość symulacji, przy czym przez stworzenie tzw. układu quasi-jednoczesnego dopuszczalna jest tu symulacja procesów, które w rzeczywistości zachodzą jednocześnie (zegar do planowania kolejnych zdarzeń rozwiązano systemowo). Język ten pozwala zatem na utworzenie dowolnego zbioru obiektów, o złożonych strukturach danych, należących do różnych klas. Obiekty te mogą być przesyłane z jednych zbiorów do innych, można również symulować ich jednoczesne działanie.

Dotychczas język SIMULA z powodzeniem stosowano do symulacji procesów technologicznych, transportowych, ekonomicznych, systemów operacyjnych maszyn cyfrowych, banków danych. W niniejszym opracowaniu postaramy się wykazać jej przydatność także przy modelowaniu i symulacji zjawisk zachodzących w biologii lub medycynie. Dotychczas większość procesów związanych z tymi dyscyplinami naukowymi analizowano bądź metodami analitycznymi, budując modele w postaci równań różniczkowych, bądź metodami symulacji analogowej. My natomiast skupimy się na przykładach rozwiązań problemów metodą symulacji cyfrowej. Pierwszy problem dotyczy symulacji zachowania się pojedynczego neuronu, drugi zaś symulacji procesu rozprzestrzeniania się epidemii. Dla zachowania przejrzystości przyjęto modele uproszczone, a nawet abstrakcyjne. Mogą być one jednak dowolnie komplikowane przez wprowadzenie dodatkowych atrybutów i bardziej szczegółowych algorytmów.

Problem 1. Symulacja pojedynczego neuronu

Neuron poddawany jest działaniu poissonowskiego strumienia impulsów o wartości średniej czasu między kolejnymi impulsami równej 1 ms. Amplitudy impulsów mają rozkład normalny N(10mV, 2.5mV). Neuron przewodzi impulsy, przy czym po przepuszczeniu impulsu neuron przechodzi w stan nieprzewodzenia na okres K x A , gdzie K - próg przewodzenia, A - amplituda impulsu. Wszystkie impulsy

przychodzące w tym czasie są przez neuron ignorowane. W tych warunkach należy zbadać przewodność neuronu w zależności od progu K.

Model analityczny dla nawet tak uproszczonego procesu przewodzenia impulsów przez neurony jest bardzo złożony. A oto jak się tworzy model symulacyjny wykorzystujący język SIMULA-67. Przede wszystkim należy zdefiniować obiekty występujące w procesie. Będą nimi: generator impulsów (streampulse), impuls oraz obiekt powodujący odblokowanie neuronu (unblock). Zdefiniujmy bliżej te obiekty. Generator impulsów jest procesem, którego zadaniem jest generowanie nowych impulsów, zgodnie z zadanym rozkładem prawdopodobieństwa.

```
process class streampulse;
begin
    while true do
    new impuls ; hold(negexp(1,U));
end ;
```

Należy zwrócić uwagę na to, że klasa streampulse jest poprzedzana (prefiksowana) systemową klasą process. Jest to bardzo ważna własność języka SIMULA. Prefiksowanie stwarza możliwość tworzenia dowolnych hierarchicznych struktur opisu obiektów. Obiekt należący do klasy prefiksowej inną klasą posiada zarówno własności danej klasy, jak i prefiksu. W naszym przykładzie generator impulsów, oprócz zdefiniowanego własnego algorytmu, ma własności klasy process. Obiekty tej klasy mogą być elementami wspomnianego układu quasi-jednoczesnego, a zatem mogą "brać udział" w symulacji. Procedura systemowa HOLD/./ powoduje przesunięcie chwili zdarzenia procesu streampulse o czas określony argumentem. Symuluje to upływ czasu między wygenerowaniem kolejnych impulsów.

W klasie impuls zdefiniowano jego amplitudę (amp). Algorytm działania obiektu tej klasy polega na wylosowaniu amplitudy zgodnie z rozkładem normalnym, a następnie, w wypadku odblokowania neuronu (zmienna boolowska collblocked), aktywacji obiektu odblokowującego neuron z opóźnieniem (delay) równym wielkości AMP*K. Oczywiście po uprzednim przypisaniu zmiennej collblocked stanu nieprzewodzenia.

```
class impuls ;
begin  real amp ;
  amp := normal/./;
  IF not collblocked then
  begin    collblocked := true ;
      activate unblock delay amp K ;
  end;
end;
```

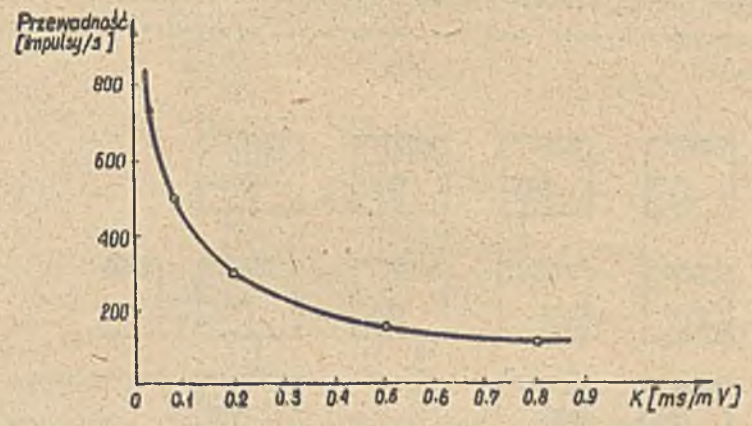
Obiekt powodujący odblokowanie neuronu (unblock) przypisuje jedynie zmiennej collblocked stan przewodzenie. Jego treść jest bardzo prosta:

```
process class unblock;
begin
while true do
collblocked := false;  passivate;
end;
```

Aby korzystać z takich procedur, jak hold, activate, passivate, zdefiniowanych w systemowej klasie Simulation, należy blok symulacji prefiksować tą klasą. Blok symulacji, oprócz deklaracji klas obiektów musi mieć instrukcję inicjującą symulację (instrukcją tą musi być instrukcja aktywacji generatora impulsów). Poniżej podano ogólnie zarys modelu (programu) symulacyjnego, przy czym pominięto treści klas obiektów, które były omówione wcześniej.

```
Simulation begin
process class streampulse .....;
class pulse ....;
process class unblock ...;
activate new streampulse;
end ;
```

Rys. 1 ilustruje wyniki przeprowadzonego eksperymentu. Krzywą otrzymano w wyniku uśrednienia pięciokrotnego przebiegu symulacyjnego.



Rys. 1. Zależność przewodności neuronu od współczynnika progowego K

Charakter krzywej jest niewątpliwie potwierdzeniem intuicyjnych oczekiwań - ze wzrostem wartości współczynnika progowego przewodność neuronu powinna maleć. Krzywą tę można aproksymować pewnymi funkcjami (np. wykładniczą) i próbować formułować ogólne prawa zachowania się procesu, pamiętając jednak o konieczności zachowania dużej ostrożności przy wszelkiego rodzaju uogólnieniach.

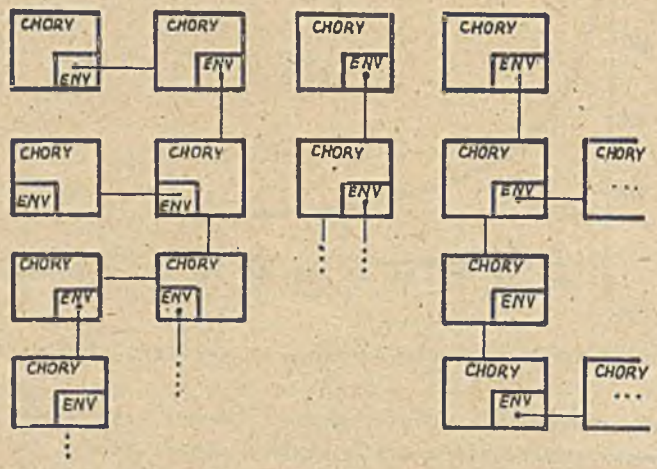
Problem 2. Model symulacyjny epidemii

W populacji o zadanej liczności POPL = 1000 szerzy się choroba zakaźna. Okres inkubacji infekcji wynosi INCUB= 3 dni, podczas którego osobnik nie zaraża otoczenia, a choroba nie wykazuje symptomów. W następnym okresie LTH= 14 dni osobnik nie leczony zaraża otoczenie. W okresie tym osobnik ma zadano prawdopodobieństwo dostania się do szpitala PROBTR(1:14). Zakłada się, że osoba poddana leczeniu zostaje wyleczona i jest odporna. Każda osoba ma oczekiwaną liczbę kontaktów CONT=20 każdego dnia, zgodną z rozkładem Poissona. Prawdopodobieństwo zarażenia osoby podczas kontaktu jest dane i wynosi PRINF=0.2. Nieleczona, po okresie czasu LTH, choroba ustaje. Zakłada się także możliwość wyleczenia środkami doraźnymi, przy czym dane jest prawdopodobieństwo sukcesu PRMASS=0.3. Należy zbadać procent zachorowań, czas trwania epidemii w zależności od rozkładu prawdopodobieństwa rozpoczęcia leczenia.

Budowę modelu symulacyjnego rozpoczniemy od napisania pierwszej wersji programu, w którym uwzględnimy tylko obiekty w nim występujące i ewentualnie ich atrybuty.

```
Simulation begin
        process class chory (Q);
        ref(head)Q;
        begin
            ref(head)ENV;
            boolean sympt;
            procedure infect/./;
            ..........
        end;
        process class leczenie(pacjent);
        ref(chory)pacjent;
        ..............
        activate new chory(11);
        end;
```

Z punktu widzenia symulacji chora osoba jest zatem procesem. Jej atrybutem jest zbiór ENV, w którym umieszczone są wszystkie osoby przez tę osobę zarażone. Atrybut Q reprezentuje zbiór, do którego dostaje się chory. Jeden chory może być co najwyżej elementem jednego zbioru ENV. Widać stąd, że chore osoby tworzą bardzo złożone drzewo o dynamicznie zmieniającej się strukturze. W trakcie symulacji, w miarę jak epidemia jest powstrzymywana, przekształca się ono w mniejsze struktury drzewiaste (rys. 2.).



Rys. 2. Drzewo epidemii

W klasie "chory" zdefiniowano ponadto procedurę infect, w której z prawdopodobieństwem PRINF generowani są nowi chorzy i umieszczani w zbiorze ENV chorego zarażającego. Zarażony osobnik po umieszczeniu w odpowiednim zbiorze ENV rozpoczyna okres inkubacji. Działanie programu dla takiej osoby należy zatem zatrzymać na okres INCUB, co w języku SIMULA realizuje instrukcja HOLD(INCUB). Następnie, zgodnie z rozkładem prawdopodobieństwa PROBTR, aktywowany musi być proces leczenia. Może to realizować np. instrukcja: if probtr(dzień) then activate new leczenie(this chory). Jeżeli osobnik nie może rozpocząć leczenia wywoływana jest procedura infect itd. Treść klasy "chory" można więc zapisać następująco:

```
process class chory(Q);
ref(head) Q;
begin
  ref(head)ENV; boolean  sympt;
  procedure infect(n);
  for i:=1 step 1 until n do
  if prinf then activate new chory(ENV);
  env :- new head;
  into(Q); hold(INCUB); sympt:=true;
  for dzień:=1 step 1 until lth do
  begin
  if PROBTR(dzień) then activate new leczenie(this chory);
  infect(poisson(CONT,U));
  hold(1);
  end;  out;
end;
```

Obiekty klasy "head" są zbiorami, a zmienne referencyjne Q,ENV - nazwami tych zbiorów. Klasa "head" jest klasą systemową w języku SIMULA.

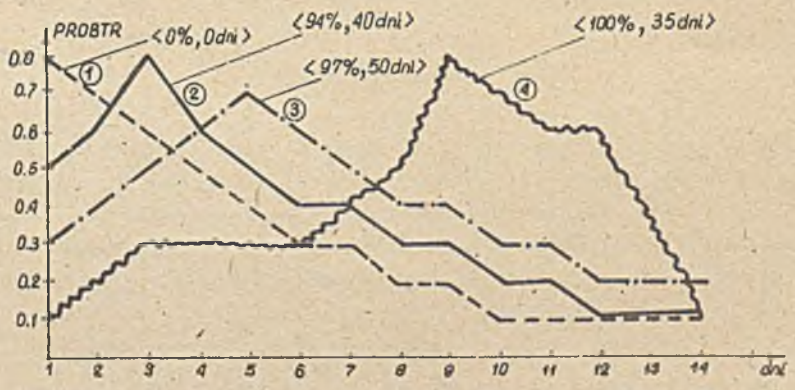
Proces "leczenie" ma jeden atrybut, którym jest chory pacjent. Chory musi być usunięty ze zbioru ENV, w którym został umieszczony w chwili zarażenia (realizuje to instrukcja out). Jeśli symptomy choroby są widoczne, chory poddawany jest leczeniu, a ponadto przeszukuje się jego otoczenie (zbiór ENV) w celu spowodowania leczenia osób, które zaraził (implikuje to przeszukiwanie kolejnych zbiorów ENV tych osób itd.). Krótko mówiąc: realizuje się obieganie pewnych gałęzi drzewa

opidemii. Pacjentu, u którego nie wykryto symptomów choroby leczy się profilaktycznie, z określonym prawdopodobieństwem sukcesu. Jego otoczenie ENV nie jest przeszukiwane. Pełną treść klasy "leczenie" podano poniżej:

```
process class leczenie(pacjent);
ref(chory)pacjent;
begin
 inspect pacjent do
 begin
   out;
   if sympt then
   begin
    cancel(pacjent);
    hold(1);
        p:- ENV.first;
    while p =/= none do
    activate now leczenie(p); p:-p.suc;
   end  else if PRMASS then
   cancel(pacjent);
 end;
end;
```

Jak wynika z ogólnego opisu programu pierwsza infokcja generowana jest w programie głównym.

Wyniki przeprowadzonego eksperymentu symulacyjnego ilustruje rys. 3.



Rys. 3. Procent zachorowań i czas trwania epidemii w zależności od rozkładu prawdopodobieństwa rozpoczęcia leczenia

Eksperyment przeprowadzono dla czterech różnych rozkładów prawdopodobieństwa rozpoczęcia leczenia. Z rysunku wynika, że natychmiastowe rozpoczynanie leczenia nie powoduje w ogóle powstania epidemii (krzywa 1). Opóźnienie, nawet trzydniowe (krzywa 2), powoduje gwałtowny wzrost liczby zachorowań. Czas trwania epidemii, gdy zostanie ona zalnicjowana (krzywe 2,3,4) zależy przede wszystkim od wielkości prawdopodobieństwa leczenia, mniej od chwili jego rozpoczęcia. Zauważmy, że duże wartości prawdopodobieństwa leczenia (krzywa 4), nawet w późniejszym czasie wydatnie skracają czas trwania epidemii (krzywa 3). Wynika stąd wniosek, że późniejsze lecz intensywne rozpoczęcie walki z epidemią może skrócić czas jej trwania w porównaniu z sytuacją, kiedy rozpoczynamy jej leczenie wcześnie lecz niedokładnie.

Jak widać z powyższych przykładów bardzo proste programy, które jednocześnie stanowiły modele procesów, umożliwiły ich zbadanie i wyciągnięcie ciekawych wniosków. Warto zaznaczyć, że budowa modeli, uruchomienie programów i przeprowadzenie eksperymentów zajęły jednej osobie niecałe 5 dni, włączając w to trudności, jakie wystąpiły w dostępie do maszyny cyfrowej.

Jedną z głównych zalet języka SIMULA-67 jest jego rozszerzalność. Umożliwia to łatwe dostosowanie tego języka do specjalistycznych zastosowań. Ponadto język ten odznacza się modułowością, co umożliwia łatwe dołączanie nowych podprogramów. Jest to bardzo ważne przy budowie i uruchamia-

niu złożonych modeli, kiedy rozpoczynamy od ogólnego opisu, a potem zwiększamy jego szczegółowość przez dodawanie różnych atrybutów i procedur działania. Jak się okazało SIMULA-67 wyposażona w mechanizmy komunikacji pomiędzy obiektami (procesami) umożliwia badanie tak złożonych układów, jak procesy zachodzące w organizmach żywych, które często przebiegają jednocześnie i między którymi występują liczne interakcje.

Literatura

[1] WAJTAK J.: Metody cybernetyczne w biologii i medycynie. PWN: Warszawa 1966

[2] WINKOWSKI J.: Programowanie symulacji procesów. WNT: Warszawa 1974

[3] BIRTWISTLE G.M., DAHL O.J., MYHRHAUG B., NYGAARD K.: Simula Begin. Auerbach Publishers Inc., Philadelphia 1973

mgr inż. Zbigniew POZNAŃSKI
Instytut Technologii Elektronowej

Simula 67 - uniwersalny język programowania

W numerze tym rozpoczynamy cykl artykułów pt. "Simula 67 - uniwersalny język programowania", w których przedstawimy opis języka Simula 67 wraz z przykładami jego zastosowań.

W części pierwszej omówimy: pojęcie klasy, obiekty proste, zasady ich tworzenia i komunikacji między nimi. Ponadto wprowadzimy nowy typ zmiennej, tj. zmienną referencyjną, a także wyrażenia referencyjne, podamy także zasady odległego dostępu do atrybutów obiektu. Tematem części drugiej będą zagadnienia prefiksowania klas i bloków, obiekty złożone oraz koncepcja wirtualności, ponadto zdefiniowana będzie systemowa klasa SIMSET umożliwiająca dokonywanie operacji na strukturach listowych. W części trzeciej zaznajomimy czytelników z operacjami na zmiennych tekstowych, z pojęciami klasy BASICIO do wprowadzania i wyprowadzania informacji oraz z generatorami liczb pseudolosowych. W części czwartej będzie opisany mechanizm quasi-jednoczesności oraz klasa SIMULATION orientująca język Simula 67 na zagadnienia symulacyjne. W licznych przykładach wykorzystano implementację Simuli 67 w maszynie cyfrowej IRIS-80.

Wstęp

Dynamiczny rozwój techniki komputerowej ukierunkowany jest między innymi na systemy złożone. Dotyczy to takich dyscyplin, jak biologia, medycyna, ekologia, automatyzacja procesów produkcyjnych itp. W szczególności ta ostatnia wymaga często budowy złożonych, zwykle hierarchicznych, systemów komputerowego sterowania. Opis i badanie takich systemów wymagają odpowiedniego języka. Stosowany powszechnie przez analityków język analizy matematycznej często okazuje się nieskuteczny.

Rozwój elektronicznej techniki obliczeniowej przyczynił się do powstania takich języków programowania, jak ALGOL, FORTRAN i COBOL. Służą one jednak do opisu i rozwiązywania problemów o małym, bądź średnim stopniu złożoności. Problemy złożone nakładają na język pewne wymagania, do których zaliczyć należy: możliwość opisu każdego elementu systemu w postaci pojedynczego programu, wyposażenie w mechanizmy opisujące łączne działanie tych programów z uwzględnieniem współdziałania między nimi, uniwersalność, łatwa rozszerzalność. Ta ostatnia cecha nabiera szczególnego znaczenia w chwili, gdy dąży się do definiowania języków problemowo zorientowanych na określone zastosowania. Wykorzystując prymitywy języka bazowego powinno się stworzyć projektantowi-programiście możliwość rozszerzania języka o nowe definicje, związane z analizowanym problemem.

Przedstawione wyżej własności posiada język Simula 67, który jest zarówno językiem opisu systemów, jak również językiem programowania. Wynika stąd, że opis dowolnego systemu złożonego w języku Simula 67 może stanowić zarazem program dla komputera. Język Simula 67 opracowali: O.J. Dahl, B. Myhrhaug, K. Nygaard z Norweskiego Centrum Obliczeniowego w 1968 r. Stanowi on znaczne rozszerzenie znanego języka programowania ALGOL 60. Podstawowym pojęciem Simuli 67 jest pojęcie klasy i obiektu. Obiekt może mieć swoją lokalną strukturę danych oraz algorytm działania, zdefiniowane w deklaracji klasy, do której należy. Klasy można prefiksować innymi klasami, przy czym obiekty należące do klas z prefiksami są obiektami złożonymi. W celu odwoływania się do obiektów wprowadzono zmienne referencyjne. Określono precyzyjnie zasady komunikacji między obiektami zarówno prostymi, jak i złożonymi. Ponieważ zakłada się jednoczesne działanie obiektów, zdefiniowano pojęcia ułatwiające posługiwanie się tzw. układem quasi-jednoczesnym. Ponadto systemowo zdefiniowano klasę Simset do operacji na strukturach listowych, klasę Simulation do symulacji procesów współbieżnych, klasę Basicio do wprowadzania i wyprowadzania informacji. Simula 67 ma także bardzo rozbudowany aparat do operacji na tekstach. Widać stąd, że wbrew temu, co sugeruje nazwa języka, nie jest to język typowo symulacyjny; przez systemową klasę Simulation może on być problemowo zorientowany

na symulację, jednak stanowi to tylko niewielki procent jego możliwości.

Przy wyborze języka programowania zwykle przyjmuje się pewne kryteria jego oceny. Poniżej przedstawiono własności Simuli 67 w odniesieniu do najczęściej stosowanych kryteriów:

- dostępność - aktualnie Simula dostępna jest na następujących komputerach: CII-10070, IRIS-80 (w Polsce), IBM 360,370, DEC PDP-10, CD CYBER 6000 (w Polsce), UNIVAC 1100 (w Polsce - Simula 1),
- powszechność - Simula jest językiem programowania ogólnego przeznaczenia, do większości zadań programistycznych, nie zawsze do symulacji,
- prostota - doświadczenie pokazuje, że Simula 67 może sprawić kłopoty początkującemu programiście, dopóki nie zrozumie: pojęcia klasy i mechanizmu quasi-jednoczesności,
- rozszerzalność - Simula ma duże możliwości rozszerzalności przez wprowadzenie koncepcji klasy; czyni to z niej język problemowo orientowalny,
- czytelność - programy pisane w Simuli odznaczają się świetną czytelnością i mogą z powodzeniem stanowić dokumentację,
- strukturalność modułowość - przez wprowadzenie koncepcji bloku, procedury i klasy Simulę można uważać za język programowania strukturalnego,
- efektywność - zależy od kompilatora; testy pokazały, że jest ona porównywalna z innymi językami,
- przetwarzanie list - Simula jest bardzo elastyczna pod tym względem (klasa Simset),
- przetwarzanie tekstów i operacje wejścia/wyjścia - Simula ma w tej dziedzinie możliwości, których nie ma żaden ze zwykłych języków symulacyjnych,
- procesy stochastyczne - w Simuli zdefiniowano generatory liczb pseudolosowych,
- wydruki raportów - brak wydruków standardowych.

Powyższa charakterystyka była podstawą wyboru języka Simula 67 jako języka programowania symulacji różnych systemów, do których zaliczyć można:

- porty lotnicze i morskie,
- transport kolejowy i drogowy,
- domy handlowe,
- fabryki,
- systemy operacyjne komputerów,
- banki danych,
- systemy socjologiczne, biologiczne, telefoniczne, ekonomiczne, sterowania, militarne
- modele administracji itp.

Wydaje się celowe rozpocząć prezentację języka Simula 67 od przypomnienia podstawowego pojęcia języka ALGOL 60, tzn. bloku, a następnie pojęcia procedury. Dalej omówione będą: pojęcie klasy, obiekty proste, zasady ich tworzenia i komunikacji między nimi. Ponadto wprowadzony będzie nowy typ zmiennej, tj. zmienna referencyjna, omówimy także wyrażenia referencyjne oraz zasady dostępu odległego od atrybutów obiektu.

Bloki

Mając do czynienia w praktycznych zastosowaniach z problemami złożonymi, dużego znaczenia nabierają metody dekompozycji tych problemów na prostsze. Podstawowym mechanizmem umożliwiającym dekompozycję w języku ALGOL i Simula jest blok. Blok stanowi formalny opis struktury danych i algorytmów. Podczas realizacji bloku powstaje jego dynamiczny egzemplarz jednostka dynamiczna , będący dokładną kopią opisu formalnego, w której zmienne lokalne oznaczają wydzielone dla niego miejsca pamięci komputera.
Ogólna struktura bloku ma postać

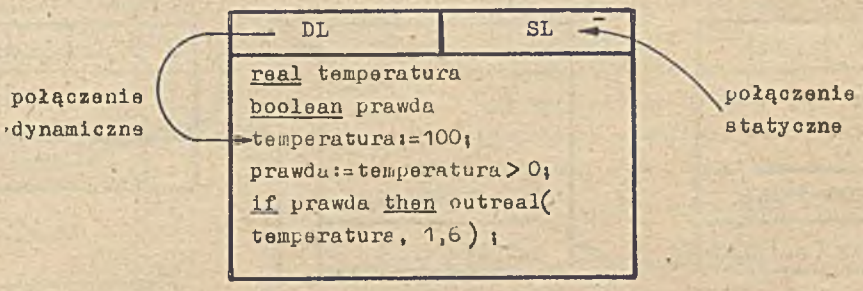
```
begin
    D1 ;
    D2 ;
    .
    Dk ;
    I1 ;
    .
    In ;
end ;
```

gdzie D_i - deklaracja i,
I_i - instrukcja i

Wygodna, szczególnie dla zrozumienia treści dalszych punktów, jest graficzna reprezentacja bloku. Rozważmy przykład:

```
begin
  real temperatura; boolean prawda;
  temperatura := 100;
  prawda := temperatura > 0 ;
  if prawda then outreal( temperatura,1,6);
end;
```

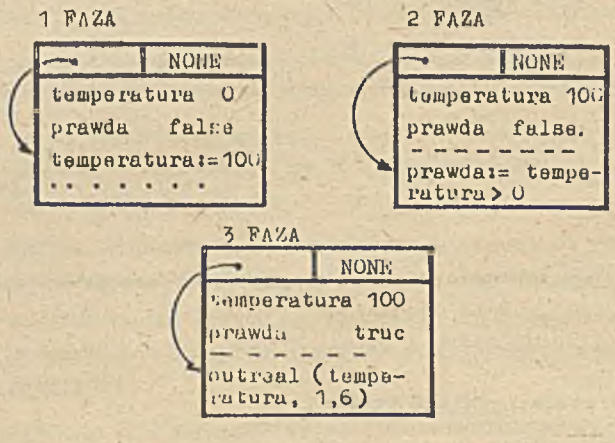
Graficzną reprezentację powyższego bloku ilustruje rys. 1.



Rys. 1.

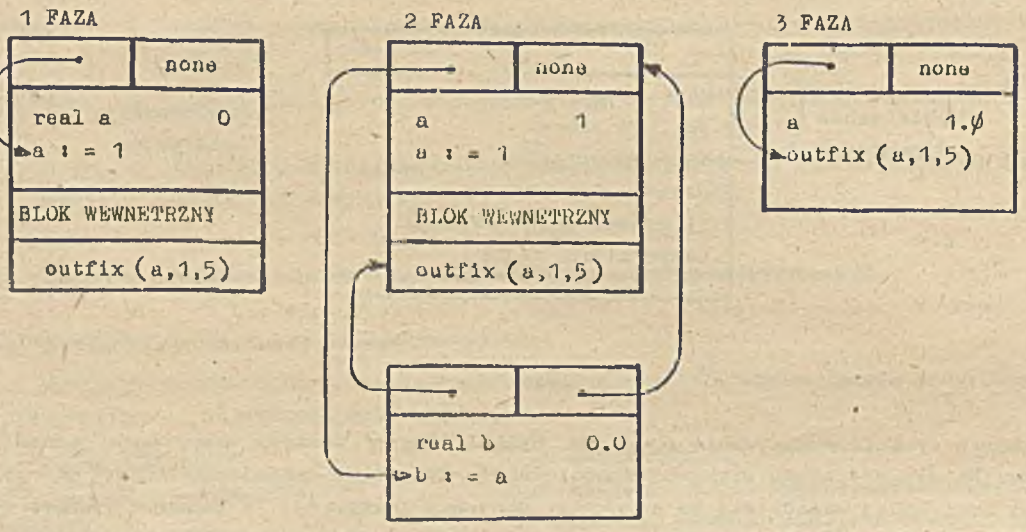
Blok powstaje w trakcie wykonywania programu. Symbol 'begin' oznacza utworzenie jednostki dynamicznej bloku. Na długość bloku wskazuje symbol 'end'. Jednostka dynamiczna bloku (na rys. 1). w reprezentacji graficznej składa się ze struktury danych i instrukcji, a ponadto posiada dwa atrybuty DL i SL. Atrybut DL stanowi połączenie dynamiczne i wskazuje na wykonywaną instrukcję bloku. SL jest połączeniem statycznym i wskazuje na najbardziej tekstowo przylegający blok. W przykładzie z rys. 1 blok taki nie istnieje i SL przyjmuje wartość none. Należy podkreślić, że w rzeczywistości jednostka dynamiczna bloku zawiera tylko miejsca na strukturę danych bloku, tj. w trakcie generowania nowej jednostki dynamicznej uwzględniany jest tylko obszar na zmienne lokalne bloku. Dokładniejszą analizę realizacji treści rozważanego bloku ilustruje rys. 2.
Po wykonaniu ostatniej instrukcji wartość DL jednostki dynamicznej bloku przyjmuje wartość none. Jednostka ta przestaje istnieć. Rozważamy bardziej złożony przykład:

```
begin
    real a; a;=1;
    begin
        real b
        b:=a;
    end;
    outfix a,1,5 ;
end;
```



Rys. 2.

Należy zwrócić uwagę na wystąpienie tu bloku wewnętrznego. Realizację powyższego bloku ilustruje rys. 3.



Rys. 3.

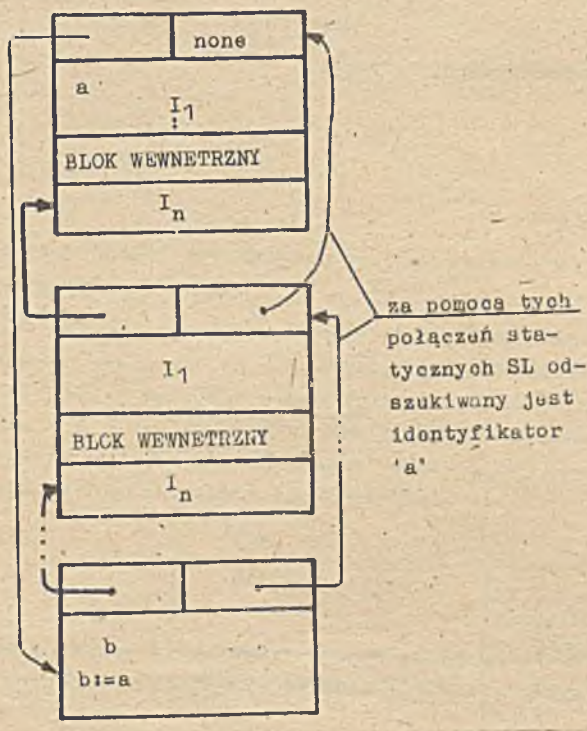
Rozważmy najbardziej interesującą drugą fazę obliczeń.

Z chwilą napotkania identyfikatora 'begin' bloku wewnętrznego utworzona została jednostka dynamiczna bloku wewnętrznego i przyłączona do jednostki dynamicznej bloku zewnętrznego. DL jednostki dynamicznej bloku zewnętrznego wskazuje na wykonywaną instrukcję. DL jednostki dynamicznej bloku przyłączonego zawsze wskazuje na pierwszą instrukcję w bloku zewnętrznym występującą po bloku wewnętrznym (na rys. 3 instrukcja wyjścia outfix). W bloku wewnętrznym odszukana jest zmienna 'b' jako lokalna. Zmienna 'a' jest globalna w stosunku do bloku wewnętrznego. Jest ona odnaleziona za pomocą SL jednostki dynamicznej bloku wewnętrznego, która wskazuje na blok zewnętrzny, jako blok najbardziej przylegający tekstowo. Po wykonaniu treści bloku wewnętrznego atrybuty jego jednostki dynamicznej DL i SL przyjmują wartość none i blok ten przestaje istnieć. Rozpoczyna się 3 faza realizacji bloku zewnętrznego.

Obliczenie bloku polega więc na tworzeniu i likwidowaniu wzajemnie powiązanych jednostek dynamicznych bloku, w których realizują się pewne operacje. Przy realizacji bloków złożonych istnieje określona zasada odszukiwania zmiennych:

- jeśli identyfikator zmiennej jest zdefiniowany w bloku najbardziej tekstowo przylegającym (rys. 3.) do bloku, w którym zmienna ta wystąpiła, to zostaje on związany z tą zmienną,
- w przeciwnym razie rozważany jest blok najbardziej tekstowo przylegający do bloku rozpatrywanego poprzednio. Krok ten jest powtarzany do chwili odnalezienia identyfikatora zmiennej.

Powyższa procedura realizowana jest za pomocą połączeń statycznych SL jednostek dynamicznych kolejnych bloków (rys. 4.). Wprowadzenie pojęcia jednostki dynamicznej bloku wraz ze zmiennymi DL i SL znacznie ułatwia zrozumienie koncepcji bloku. Zmienne te mają specyficzne własności, ponieważ wskazują na pewne obiekty (bloki), a zatem na dowolne struktury danych. Zmienne tego typu będą odgrywały zasadniczą rolę przy wprowadzeniu pojęcia klasy i obiektu, dlatego dobrze jest zrozumieć ich istotę już teraz.



Rys. 4.

Procedury

Z dotychczasowych rozważań wynika, że bloki realizują określone funkcje. Jeżeli zachodzi konieczność realizacji tych funkcji wielokrotnie, dla różnych wartości parametrów, wygodnie jest wprowadzić pojęcie procedury.

Ciało procedury jest blokiem. Istnieje mechanizm wywoływania procedury, co umożliwia rozróżnienie tekstowe między jej definicją i wykorzystaniem. Zdefiniowane są także zasady przesyłania parametrów podczas jej wykonywania. Pojęcie procedury umożliwia zatem dalszą dekompozycję bloku od jego "otoczenia w programie".

Deklaracja procedury ma następującą postać:

(B) procedure $A(P_1^F, P_2^F, \ldots, P_n^F)$;
< specyfikacja sposobu przesyłania parametrów>;
< specyfikacja parametrów formalnych > ;

begin
  D_1 ;
  D_2 ;
  :
  D_k ;
  I_1 ;
  :
  I_k ;
  bloki wewnętrzne ;
  I_{k+1} ;
  :
  I_n ;
  (A := .....) ;
end ;

} ciało procedury

gdzie: A - nazwa procedury,
B - jeśli procedura jest funkcyjna, określa typ wartości funkcji (integer, real, boolean, ref*)
$P_1^F, \ldots, P_n^F$ - parametry formalne procedury,
D_i - deklaracja i,
I_i - instrukcja i.

Uwaga: W wypadku deklaracji procedury funkcyjnej B należy pod jej nazwę A podstawić wartość odpowiedniego typu (A :=...).

Wywołanie procedury ma postać $A(P_1^A, P_2^A, \ldots, P_n^A)$,

gdzie $P_1^A, \ldots, P_n^A$ - parametry aktualne.

* Jest to typ referencyjny. Wartością procedury jest referencja do obiektu. Zmienne tego typu omówione będą w punkcie "Klasy".

Typy parametrów aktualnych muszą być zgodne z typami odpowiednich parametrów formalnych, np.

```
begin
    procedure algorytm(ciśnienie, temperat, przełącz, danepomiar);
    real ciśnienie, temperat; boolean przełącz; array danepomiar;
    begin
        <ciało procedury>
    end;
    algorytm(a+5.0,100.0,z,f);
end;
```

Zasady przesyłania parametrów w procedurach

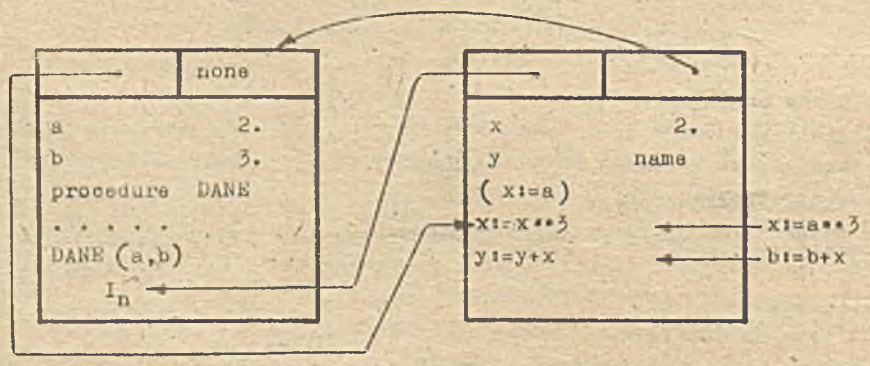
Istnieją trzy sposoby przesyłania parametrów przy wywoływaniu procedur:

- przez wartość,
- przez nazwę,
- przez referencję.

Rozważmy pierwsze dwa z nich na następującym przykładzie:

```
begin
    real a,b;
    procedure DANE(x,y);
    value x; name y; real x,y;
    begin
        x := x**3;
        y := y + x;
    end ;
    a := 2.0; b := 3.0;
    DANE(a,b);
    I_n;
end
```

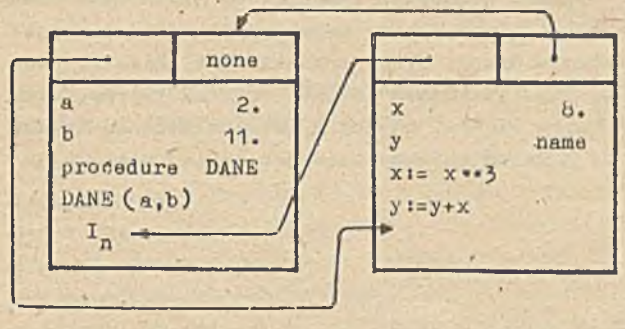
Wykorzystując wprowadzoną wcześniej graficzną reprezentację bloku możemy przedstawić sytuację, po wykonaniu pierwszych instrukcji podstawienia, tj. a:=2.0, b:=3.0 oraz po wywołaniu procedury DANE (rys. 5).



Rys. 5.

Wywołanie procedury DANE spowodowało powstanie nowej jednostki dynamicznej i przyłączenie jej do bloku, w którym procedura ta została wywołana. Z chwilą utworzenia jednostki dynamicznej procedury dla parametrów przesyłanych przez wartość, dokonuje się podstawienie: parametr formalny := obliczony parametr aktualny (w naszym przykładzie x:=a). Wartości parametrów aktualnych zostają odszukane przez DL jednostki dynamicznej procedury. Realizacja pierwszej instrukcji procedury DANE ma zatem postać : x := a**3.

W wypadku przesyłania parametrów przez nazwę następuje tekstowe zastąpienie parametrów formalnych przez parametry aktualne (podczas konfliktu nazw postępowanie jest bardziej złożone), także z wykorzystaniem DL jednostki dynamicznej procedury. Stąd wykonanie drugiej instrukcji procedury DANE ma postać: b:=b+x. Końcową sytuację ilustruje rys. 6.



Rys. 6.

Z przykładu tego wynika, że w odróżnieniu od przesyłania parametru przez nazwę, kiedy to wartość parametru aktualnego może ulec zmianie, przesyłanie parametru przez wartość nie powoduje zmiany wartości parametru aktualnego.

Uwaga: Parametr formalny przesyłany przez nazwę, występujący z lewej strony instrukcji podstawienia w ciele procedury, może odpowiadać jedynie takiemu parametrowi aktualnemu, który jest zmienną.

Przesyłanie przez wartość parametrów będących zmiennymi tekstowymi jest bardziej złożone i będzie omówione w dalszych częściach opracowania. Przesyłanie parametrów przez referencję, w szczególności gdy są nimi zmienne referencyjne, podane będzie przy omawianiu zmiennych tego typu w punkcie "Klasy".

Należy podkreślić, że przez referencję przesyłane są także takie parametry jak: tablice, procedury, etykiety i przełączniki, które w rzeczywistości nie są zmiennymi referencyjnymi, np.w wypadku tablic można to sobie wyobrazić jako przesyłanie adresu tablicy. Parametry te nie mogą ulec zmianie w ciele procedury. Nie wyklucza to jednak możliwości zmian np. elementów tablicy. Wszystkie typy parametrów procedur włącznie z dotychczas nie omówionymi (referencyjne i tekstowe) oraz zasady ich przesyłania ilustruje tab. 1.

Tab. 1.

Parametr typy	Sposób przesyłania		
	przez wartość (value)	przez referencję	przez nazwę (name)
real, integer, boolean, character	normalny	niemożliwy	możliwy
ref	niemożliwy	normalny	możliwy
text	możliwy	normalny	możliwy
(real, boolean, integer, character) array	możliwy	normalny	możliwy
(text, ref) array	niemożliwy	normalny	możliwy
procedura	niemożliwy	normalny	możliwy
etykieta	niemożliwy	normalny	możliwy
przełącznik	niemożliwy	normalny	możliwy

Jeśli przed specyfikacją parametrów formalnych nie podaje się sposobu przesyłania parametrów np. name lub value, to parametry przesyłane są w sposób normalny. Dotyczy to także zasad przesyłania parametrów w klasach.

Oprócz wspomnianych na początku niewątpliwych korzyści wynikających z wykorzystania procedur, istnieją jeszcze inne, które są wystarczającą motywacją do ich stosowania w programach. Są to:modułowość programów oraz przejrzystość opisu modeli. Stwierdzenie to nabiera szczególnej wagi przy analizie złożonych systemów i procesów, kiedy to niezbędna jest czytelność i modułowość programów, ułatwiające ich uruchamianie i umożliwiające wprowadzanie różnych modyfikacji.

Klasy

Wprowadzenie bloku (procedury) umożliwia, przy każdym jego zadziałaniu, utworzenie jego jednostki dynamicznej i dokonanie sekwencji operacji, zgodnie z zawartymi w nim instrukcjami (algorytmem). Aby to osiągnąć każdy blok musi mieć własną strukturę danych. Po wykonaniu wszystkich instrukcji jednostka dynamiczna bloku jest niszczona. Pojęcie jednostki dynamicznej bloku daje możliwość tworzenia wielu jednostek dynamicznych tego samego bloku, które mogą współistnieć i wzajemnie na siebie oddziaływać. W związku z tym wprowadzono w Simuli 67 pojęcie klasy i obiektu, które jest uogólnieniem pojęcia bloku. Podobnie jak przy "wystąpieniach" bloków - można tworzyć wiele współistniejących obiektów, tj. jednostek dynamicznych bloków, mających te same zmienne lokalne i algorytmy działania, opisane w deklaracji klasy. A zatem obiekty należące do tej samej klasy opisane mogą być przez deklarację klasy.

Deklaracja klasy

Deklaracja klasy jest podobna do deklaracji procedury i ma postać:

```
class A(PA);SA;
begin
    DA;
    IA;
    inner;
    FA;
end;
```

gdzie:
- A - nazwa (identyfikator) klasy,
- PA - zbiór parametrów formalnych klasy A,
- SA - zbiór specyfikacji parametrów formalnych (PA),
- DA - zbiór deklaracji zmiennych klasy A,
- IA - zbiór instrukcji początkowych treści klasy A,
- FA - zbiór instrukcji końcowych treści klasy A.

Symbol '*inner*' wyjaśniony będzie później. Na tym etapie traktujmy go jako instrukcję pustą.

Wielkości podane w deklaracjach PA i DA nazywamy atrybutami klasy, przy czym $PA \cap DA = \emptyset$. Są one jednocześnie atrybutami każdego obiektu należącego do tej klasy. Zauważmy, że jedną z deklaracji DA może być deklaracja innej klasy.

Przykłady

- Punkt na płaszczyźnie określony jest przez jego współrzędne. Stąd instrukcja deklaracji klasy 'punkt' ma postać:

```
class punkt (x,y); real x,y;
```

Do klasy tej mogą należeć wszystkie obiekty posiadające własności punktu.

- Zdefiniujmy teraz klasę opisującą komputer:

```
class komputer (typ,nazwa);          <-------- PA
integer typ; text nazwa;             <-------- SA
begin
   integer pao,liczinstr,generacja;  }
   real cykl;                        } <-------- DA
   cykl := inreal;                   }
   pao := inint;                     }
   liczinstr := inint;               } <-- IA + FA
   generacja := uniform(1,3,U);      }
   <algorytm działania>              }
end;
```

Zwróćmy uwagę na przejrzystość opisu obiektów przez deklarację klasy. Jest zupełnie naturalne, że przy analizie obiektów (systemów), należy najpierw opisywać ich atrybuty (np. pao, liczinstr), a następnie algorytmy ich działania. Dla porządku podajemy, że procedury inreal i inint służą do wprowadzania wartości typu odpowiednio rzeczywistego i całkowitego, a procedura uniform (a,b,U) losuje liczbę z przedziału $[a,b]$ zgodnie z rozkładem równomiernym.

Tworzenie obiektów prostych

Deklaracje klas stanowią tylko formalny opis obiektów. Są one jednocześnie pewną strukturą danych w programie (podobnie do procedur).

Instrukcja

$$\underline{\text{new}}\ A\left(p_1^A, p_2^A, \ldots\ p_n^A\right),$$

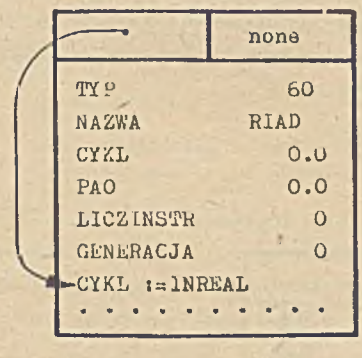
gdzie A jest nazwą klasy, $p_1^A, \ldots, p_n^A$ parametrami aktualnymi, powoduje utworzenie jednostki dynamicznej obiektu (dalej nazywanej obiektem) należącego do klasy A.

Obiektem prostym nazywamy obiekt należący do klasy bez prefiksu (przedrostka). W punkcie tym zajmiemy się tylko obiektami prostymi. Z chwilą napotkania w programie wyrażenia generującego new A zostaje utworzony obiekt i rozpoczyna się wykonywanie początkowych instrukcji IA treści klasy A. Trwa to do chwili osiągnięcia końca klasy end lub do chwili wywołania jednej z procedur powodujących zawieszenie obiektu (np. detach). Po wykonaniu wszystkich instrukcji obiektu jego struktura danych, w odróżnieniu od bloku, pozostaje nadal w pamięci komputera tak długo, jak długo możliwy jest do niej dostęp.

Zasady przesyłania parametrów w klasach podano w podpunkcie "Zasady przesyłania parametrów w klasach". Nawiązując do przykładów z podpunktu "Deklaracja klasy" mamy:

- new punkt (4.0,16.2)
  Utworzono obiekt klasy punkt o współrzędnych (4,16.2).
- new komputer (60,'riad');

Z chwilą utworzenia obiektu klasy komputer powstaje jego jednostka dynamiczna z atrybutem DL wskazującym na pierwszą instrukcję treści klasy komputer oraz atrybutem SL wskazującym, podobnie jak w wypadku bloku, na najbardziej tekstowo przylegający blok (rys. 7).



Rys. 7.

Rozpoczyna się wykonywanie kolejnych instrukcji ciała tej klasy.

Zmienne referencyjne

W ciele klasy podany jest zwykle zbiór atrybutów lokalnych w danej klasie. Często zachodzi potrzeba wykorzystania tych atrybutów przez inne obiekty, które mogą należeć do innych klas. Współistniejące obiekty mogą znajdować się w różnych stanach. Chcąc określać te stany należy stworzyć możliwość dostępu do tych obiektów, a zarazem do wszystkich ich atrybutów*. Do odwoływania się do obiektów służą tzw. zmienne referencyjne. Referencja wskazuje położenie struktury danych obiektu w pamięci komputera. Deklaracja zmiennych referencyjnych ma postać:

ref(identyfikator klasy) < lista identyfikatorów>.

Np. ref(punkt)P;
ref(komputer)array K(1:3);

Każda zmienna referencyjna musi być zakwalifikowana do danej klasy. Podanie identyfikatora klasy w deklaracji zmiennej referencyjnej kwalifikuje tę zmienną do tej klasy. I tak np. zmienna referencyjna P kwalifikowana klasą punkt może być referencją do obiektu tej klasy.

Referencje są wartościami wyrażeń referencyjnych. Wyrażeniami referencyjnymi są:

---

* W wypadku zwykłego bloku dostęp do jego zmiennych lokalnych z zewnątrz jest niemożliwy.

- każde wyrażenie generujące new,
- zmienna prosta typu referencyjnego (ref),
- zmienna indeksowana typu referencyjnego (ref),
- referencja lokalna this.

A zatem wartością wyrażenia generującego new jest referencja do utworzonego obiektu. Wartością początkową każdej zmiennej referencyjnej jest none.

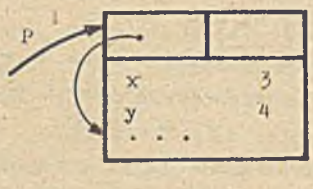
Podstawienia referencyjne

Podobnie, jak dla zmiennych prostych istnieje instrukcja podstawienia w postaci W1 := W2, także dla zmiennych referencyjnych wprowadzono tzw. instrukcję podstawienia referencyjnego w postaci W1 :- W2, gdzie W1 - zmienna referencyjna, W2 - wyrażenie referencyjne, tzn.

< zmienna referencyjna > :- < wyrażenie referencyjne > .

Przykłady: (patrz podpunkt "Deklaracja klasy")

- ref(punkt)P;
  P :- new punkt(3.,4.);
  Zmienna P jest referencją do obiektu klasy punkt (rys. 8).
  Często mówi się, że jest ona nazwą tego obiektu.



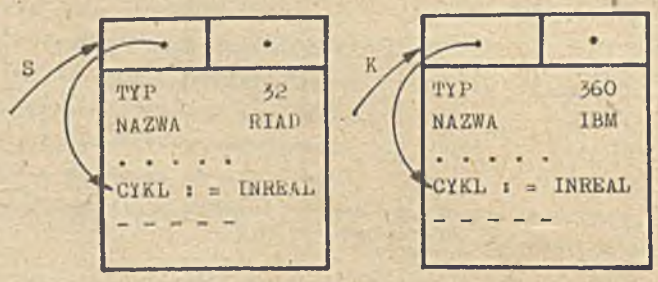
Rys. 8.

- ref(komputer)S,K;
  S :- new komputer(32,'riad');
  K :- new komputer(360,'ibm');

Rys. 9 ilustruje nowo utworzone obiekty S i K klasy komputer. W podstawieniach referencyjnych musi być zachowana zgodność w zakresie kwalifikacji zmiennych. Zmienna referencyjna z lewej strony musi być kwalifikowana tą samą klasą, co wyrażenie referencyjne z prawej strony instrukcji podstawienia referencyjnego, np. jeśli zmienna P kwalifikowana jest klasą punkt (ref(punkt)P), a zmienna S klasą komputer (ref(komputer)S), to instrukcje:

- P :- new komputer(80,'iris');
- S :- new punkt(1.,2,);

są nieprawidłowe.

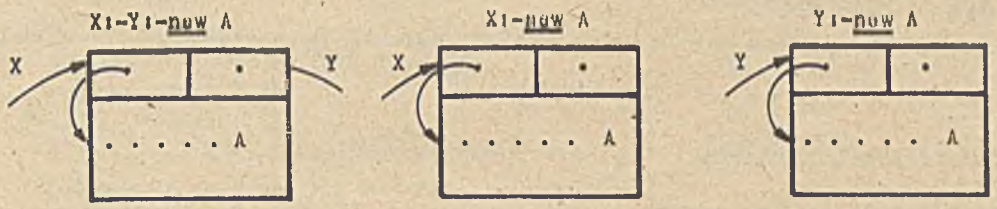


Rys. 9.

Uwaga: Istnieje zasadnicza różnica (rys. 10) wyniku działania w następujących podstawieniach referencyjnych (nie występuje ona przy zwykłych podstawieniach typu ':='):

- X :- Y :- new A ;
- X :- new A; Y :- new A;

gdzie X,Y kwalifikowane są klasą A.
W wypadku pierwszym powstaje jeden obiekt, do którego referencję dają dwie zmienne X,Y, w wypadku drugim powstają dwa różne obiekty, wskazywane przez zmienne referencyjne odpowiednio X i Y.



Rys. 10.

Zasady przesyłania parametrów w klasach

Podczas generowania nowych obiektów parametry formalne klas, do których należą, zastępuje się parametrami aktualnymi. Dopuszczalne typy parametrów klas oraz zasady ich przesyłania ilustruje tab. 2.

Tab. 2.

Parametr typy	Sposób przesyłania		
	przez wartość (value)	przez referencję	przez nazwę (name)
Real, integer, boolean, character	normalny	niemożliwy	niemożliwy
ref	niemożliwy	normalny	niemożliwy
text	możliwy	normalny	niemożliwy
(real, integer, boolean, character) array	możliwy	normalny	niemożliwy
(ref, text) array	niemożliwy	normalny	niemożliwy
PROCEDURA	niemożliwy	niemożliwy	niemożliwy
Etykieta	niemożliwy	niemożliwy	niemożliwy
Przełącznik	niemożliwy	niemożliwy	niemożliwy

Poznanie zmiennych referencyjnych i podstawień referencyjnych umożliwia łatwe wyjaśnienie sposobu przesyłania parametrów przez referencję, który zasygnalizowano przy omawianiu procedur. Sposób ten jest analogiczny do przesyłania parametrów przez wartość z tym, że zwykłą instrukcję podstawienia zastępuje się instrukcją podstawienia referencyjnego tzn.

<parametr formalny> :- <parametr aktualny> .

A zatem pod parametr formalny podstawiana jest referencja do obiektu, wyznaczona przez obliczenie wartości parametru aktualnego. Powyższy sposób przesyłania parametrów przez referencję dotyczy zarówno klas jak i procedur.

Relacje referencyjne

Do stwierdzenia, czy wartość dwóch wyrażeń referencyjnych W1 i W2 są referencjami do tego samego obiektu służą dwa wyrażenia boolowskie:

W1= =W2 - które przyjmuje wartość true, jeśli W1 i W2 mają tę samą wartość, tzn. odnoszą się do tego samego obiektu lub gdy są none,

W1=/=W2 - które przyjmuje wartość true, jeśli W1 i W2 mają różne wartości, tzn. odnoszą się do różnych obiektów,

np. ref(punkt) W1,W2,W3;
W1:-new punkt(1,2) ;W3:-new punkt(0,0);

if W1=/=W3 then W2:-new punkt(2,1);

Ponieważ W1 i W3 są referencjami do dwóch różnych obiektów, więc wygenerowany zostanie nowy obiekt W2.

Odległy dostęp do atrybutów obiektu

Niech będzie dana deklaracja klasy:

class A($P_1,\ldots,P_n$);
<specyfikacja $P_1,\ldots,P_n$>;
begin
    real a,b,c;
    D ;
    D_2 ;
    :
    I_1 ;
    :
    I_n ;
end;

W ciele klasy atrybuty deklaruje się tak jak zmienne lokalne w bloku. Aby mieć dostęp do atrybutów obiektu klasy A z zewnątrz tego obiektu (jest to niemożliwe w wypadku zwykłego bloku) stosuje się zapis:

(1) X.a ,

gdzie X jest referencją do obiektu klasy A, tzn. ref(A)X, a - atrybutem klasy A.
Zapis (1) ma sens tylko po wykonaniu instrukcji tworzącej obiekt, tzn. X:-new A(.), ponieważ dopiero wtedy zmienna referencyjna X staje się referencją do obiektu klasy A, w której zadeklarowano atrybut 'a'.

W Simuli 67, oprócz "kropkowanego", istnieją jeszcze inne sposoby odległego dostępu do atrybutów obiektu, np. mechanizm połączenia (inspect). Przedstawiono je przy omawianiu obiektów złożonych. Pamiętać jednak należy, że obowiązują one także dla obiektów prostych.

Możemy teraz napisać prosty program utworzenia dwóch punktów P(0,0),Q(1,2) oraz linii L łączącej te punkty. Zdefiniujmy zatem 2 klasy: punkt i linia. Punkt reprezentowany jest przez dwie współrzędne, tzn.

class punkt(x,y); real x,y;

natomiast linia przez współrzędne dwóch punktów, tzn.

class linia (x_1,y_1,x_2,y_2);
real x_1,y_1,x_2,y_2;
if $x_1 = x_2 \wedge y_1 = y_2$ then BLAD;

W deklaracji klasy linia sprawdza się warunek, czy dwa punkty nie pokrywają się ze sobą. Pełny program ma postać:

begin
    class punkt(x,y); real x,y;
    class linia(x_1,y_1,x_2,y_2);
    real x_1,y_1,x_2,y_2;
    if $x_1 = x_2 \wedge y_1 = y_2$ then BLAD;
    § należy teraz zadeklarować zmienne referencyjne kwalifikowane
    klasami punkt i linia §
end*

---

* Między znakami § znajdują się komentarze. Zasada ta obowiązuje w całej pracy.

```
        ref(punkt)P,Q;
        ref(linia)L;
        $ utwórzmy dwa punkty $
        P:- new punkt(0,0);
        Q:- new punkt(1,2);
        $ utwórzmy linię przechodzącą przez punkty  P  i  Q $
        L:- new linia(P.x,P.y,Q.x,Q.y);
        $ P.x  oznacza współrzędną  x  punktu P; jest to jedyny sposób dostępu do tego a-
              trybutu, ponieważ instrukcja generacji linii  L  znajduje się na zewnątrz
              obiektu klasy punkt (w programie głównym)$
    end;
```

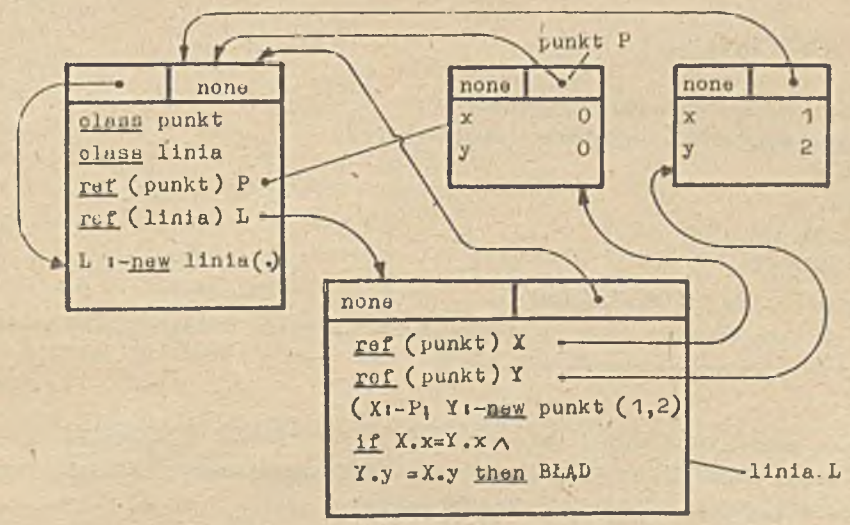
Obiekt klasy linia można zdefiniować inaczej, wykorzystując jako parametry formalne nie współrzędne lecz same punkty (ściślej referencje do punktów), tzn.

```
    class linia(X,Y);
    ref(punkt) X,Y;
    if  X.x=Y.x ∧ X.y=Y.y then BLAD;
```

Poprzedni program przyjmie wtedy postać:

```
begin
    class punkt.......
    class linia X,Y ; .......
    ref(punkty)P;
    ref(linia)L;
    $ wygenerujmy punkt $
    P:- new punkt(0,0);
    $ wygenerujmy linię $
    L:- new linia(P,new punkt(1,2)) ;
    $ przesłanie parametrów klasy odbywa się przez referencję, tzn.
    X:-P; Y:-new punkt(1,2); $
end;
```

Realizację powyższego bloku ilustruje rys. 11. Po wejściu do bloku zostaje utworzona jego jednostka dynamiczna. Wygenerowany zostaje punkt P, a następnie obiekt klasy linia L. W treści klasy linia pierwszymi instrukcjami są instrukcje podstawienia referencyjnego parametrów aktualnych, tzn. X:-P i Y:-new punkt(1,2). Stąd zmienna X wskazuje na jednostkę dynamiczną obiektu P,



Rys. 11.

Y zaś na jednostkę dynamiczną nowo utworzonego obiektu klasy punkt, do której nie ma referencji z programu głównego (chociaż mogłaby być). Połączenia SL wszystkich utworzonych obiektów wskazują na najbardziej-tekstowo przylegający blok, którym w tym wypadku, dla każdego z tych obiektów jest program (blok) główny.

Przykłady

Napisać program, który generuje dwa punkty R(3,4),S(5,7) i punkt o współrzędnych stanowiących sumę odpowiednich współrzędnych punktów R i S.
Rozwiążmy to zadanie trzema sposobami, aby pokazać jak dużą swobodę ma programista w definiowaniu problemu.

- ```
  begin
     class punkt(x,y); real x,y;
     ref(punkt)R,S,T;
     R:- new punkt(3,4);
     S:- new punkt(5,7);
     T:- new punkt(R.x+S.x,R.y+S.y);
  end;
  ```

- W celu obliczenia współrzędnych punkt T zdefiniujmy procedurę funkcyjną 'dodaj'. Wartością tej procedury będzie referencja do obiektu klasy punkt.

  ```
  begin
     class punkt(x,y); real x,y;
     ref(punkt)procedure dodaj(P,Q);
     ref(punkt)P,Q;
     dodaj :- new punkt(P.X+Q.x,P.y+Q.y);
     ref(punkty)R,S,T;
     R:- new punkt(3,4);
     S:- new punkt(5,7);
     T:- dodaj(R,S);
  end;
  ```

- Zadeklarujmy teraz procedurę 'dodaj' w ciele klasy punkt. W ten sposób wyeliminujemy jeden parametr formalny, gdyż w treści procedury współrzędne obiektu-punktu, w którym jest ona zadeklarowana są dostępne bezpośrednio, jako lokalne

  ```
  begin
    class punkt(x,y); real x,y;
    begin
      ref(punkt)procedure dodaj(Q); ref(punkt)Q;
      if Q=/= none then dodaj:-new punkt(x+Q.x,y+Q.y);
    end; $ punkt $
    ref(punkt)R,S,T;
    R:- new punkt(3,4);
    S:- new punkt(5,7);
    T:- R.dodaj(S); (lub T:- S.dodaj(R);)
     $ poprzez zmienną referencyjną R uzyskujemy dostęp do obiektu klasy punkt, w którym zadeklarowany jest atrybut - procedura dodaj $
  end;
  ```

Napisać program generujący punkty R(3,4),S(5,2) oraz punkt $T(x_R+x_S, y_R+y_S)$. Wygenerować linię L_1 przechodzącą przez punkty R i S oraz punkt przecięcia się linii L_1 z linią L_2 przechodzącą przez początek układu współrzędnych i punkt T.
  ```

```
begin
  $ definiujemy klasę punkt $
 class punkt(x,y); real x,y;
 begin
   ref(punkt) procedure dodaj(Q); ref(punkt)Q;
   if Q=/= none then dodaj:-new punkt(x+Q.x,y+Q.y);
   $ zdefiniujmy teraz procedurę generacji linii przechodzącej przez dowolny punkt  Q  i dany
     punkt tzn. ten, w którym procedurę tę deklarujemy $
   ref(linia) procedure poprowadź linię(Q);
   ref(punkt) Q;
   if  Q =/= none then
   poprowadź linię :- new linia(y-Q.y,Q.x-x,x*Q.y-y*Q.x);
   $ linia zdefiniowana będzie niżej jako trójka: współrzędne punktu (x1,y1) i współczynnik kie-
     runkowy, tzn. y-y1=m(x-x1)
   end; $ punkt $
   $ deklarujemy klasę linia $
   class linia(A,B,C); real A,B,C;
   begin
    $  definiujemy procedurę generacji punktu będącego przecięciem się linii L z linią daną,tzn.
      tą, w której deklarujemy tę procedurę $
     ref(punkt) procedure przecięcie linii(L);
     ref(linia)L;
     begin
       if L =/= none then
       begin
         real T;
         T:=A*L.B-B*L.A;
         if T /= Ø then
         begin
           T:=1/T;
           przecięcie linii :- new punkt(T*(L.C*B-L.B C),T*(L.A*C -L.C*A));
           $ rozwiązanie układu równań z dwiema niewiadomymi $
         end;
       end;
     end; $ linia $
     ref linia L,M;
     ref punkt R,S,T,U;
      $ tworzymy punkty R,S,T $
    R: - new punkt(3,4);
    S: - new punkt(5,2);
    T: - R.dodaj(S);
     $ tworzymy linię L przechodzącą przez punkty R i S $
    L:-  R.poprowadź linię(S);
     $ tworzymy linię M przechodzącą przez punkty T i (0,0) $
    M:-  T.poprowadź linię(new punkt(0,0));
     $ tworzymy punkt przecięcia się linii L i M $
    U:-  L.przecięcie linii(M);
 end; $ programu $
```

Wykorzystajmy teraz powyższe przykłady do rozwiązania bardziej praktycznego zadania.

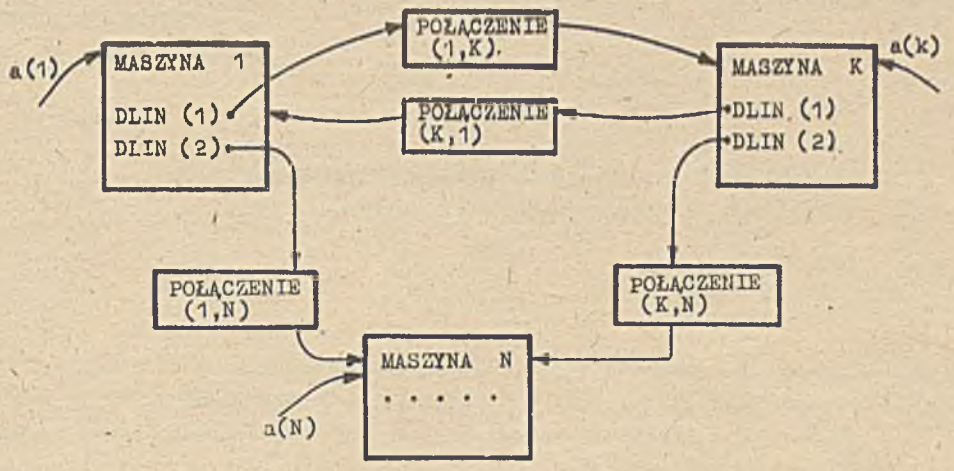
- W hali znajduje się N maszyn, z których każda wykonuje operację o czasie trwania T_i T_i(i=1,2,..,N). Macierz D określa odległości między maszynami, przy czym d_{ij} jest odległością między maszyną i i j. Warunek $d_{ij} = 0$ oznacza, że wyklucza się transport półfabrykatów w procesie technologicznym między maszynami i,j. Należy wygenerować strukturę parku maszyn tj. wszystkie maszyny oraz połączenia między nimi, według zadanej macierzy D.

```
begin
  $ zdefiniujemy klasę maszyna $
  class maszyna(numer);
  integer numer;
  begin
    integer k; $ k oznacza liczbę maszyn, z którymi połączona jest technologicznie dana maszyna $
    ref(połączenie) array dlin(1:n); $ dlin jest wektorem obiektów klasy połączenie związanych
                                      z daną maszyną(rys. 12) $
    ref(połączenie) procedure join(f);
    ref(maszyna)f ;
    join :- new połączenie(this maszyna, f);
    $ procedura join generuje obiekt klasy połączenie stanowiący połączenie danej maszyny z ma-
      szyną f określoną parametrem procedury $
    real T;
    T :- inreal;
  end; $ maszyna $
  $ zdefiniujmy klasę połączenie, której atrybutami są referencje do dwóch maszyn $
  class połączenie (M1,M2) ;
  ref maszyna(M1,M2);
  begin real d;
     d := inreal; $ wczytanie odległości między maszynami $
  end; $ połączenie $
$ zdefiniujmy procedurę genstruct generującą strukturę parku maszyn, z parametrem d określają-
  cym macierz połączeń maszyn $
procedure genstruct(d);
array d:
begin
$ wygenerujmy obiekty klasy maszyna $
  for i:=1 step 1 until N do
  a(i) :- new maszyna(i);
  $ dla każdej maszyny wygenerujmy połączenie wg macierzy d $
  for i:=1 step 1 until N do
  begin
    a(i).k:=0;
    for j:=1 step 1 until N do
    if d(i,j) /= 0 then
    begin
      a(i).k:=a(i).k + 1;
      a(i).dlin(k) :- a(i).join(a(j));
    end;
  end;
end; $ genstruct $
ref(maszyna) array a(1:N);
array D(1:N);
genstruct (D); $ generuj strukturę $
end; $ programu $
```

Zwróćmy uwagę na to, że obiekty klasy maszyna można opisać dużo szczegółowiej. Również połączenia między nimi (class połączenie) mogą zawierać znacznie więcej informacji. Wygenerowana struktura może stanowić zatem bank informacji o parku maszyn i stanowić punkt wyjścia do symulacji procesu produkcyjnego.

---

* this jest referencją lokalną wskazującą na dany obiekt, w którym występuje; będzie ona szczegółowiej omówiona w dalszych częściach.



Rys. 12.

W następnej części podamy możliwości Simuli 67 w zakresie tworzenia hierarchicznych struk opisu procesów (podklasy) oraz operacji na strukturach listowych (systemowa klasa SIMSET).

Bibliografia

[1] BIRTHWISTLE G. i in.: Notes on the Simula Language. Publication No.S-7 NCC Forsknings 1 B, Oslo 1969

[2] Common Base Language. Simula Information. Publication No. S-22 NCC Forskningsveien 1 B 1970

[3] Simula Begin. Auerbach Publishers Inc. Philadelphia, Pa, 1973

[4] Simula sous SIRIS7/SIRIS8 manuel d'utillisation tome 1. CII 1973

Biuletyn Informacyjny OBIEKTOWE SYSTEMY KOMPUTEROWE 3/79

mgr inż. Halina GUTOWSKA
Instytut Maszyn Matematycznych

Problemy przygotowywania programów obróbki części oraz właściwej eksploatacji systemu APT

Wstęp

W niniejszym opracowaniu zaproponowano pewien sposób podejścia do problemu przygotowywania pojedynczych programów obróbki części w języku APT (Automatically Programmed Tools), jak również zasygnalizowano pewne problemy związane z właściwą eksploatacją systemu APT.

Przyjmijmy sytuację modelową, w której mamy do czynienia z maszyną cyfrową z zaimplementowanym systemem APT, pracującym w trybie wsadowym. W systemie tym funkcjonuje postprocesor dla obrabiarki, dla której będą pisane programy obróbki części, natomiast programista dysponuje wszelkimi danymi potrzebnymi do napisania takich programów. Przy takim założeniu rozpatrywany będzie problem przygotowania pojedynczych programów obróbki części, tzn. takich, że jeden program opisuje obróbkę jednej części. Pod pojęciem przygotowania programu obróbki części rozumiemy:

- opracowanie projektu programu obróbki części,
- jego zakodowanie w postaci instrukcji języka APT, a następnie
- uruchomienie i sprawdzenie poprawności napisanego programu oraz
- wygenerowanie taśmy sterującej obrabiarką.

W zasadzie do procesu przygotowywania programu obróbki części należy zaliczyć również opracowanie dokumentacji programu. Problem ten - dosyć rozległy - stanowi jednak odrębne zagadnienie, wymagające zebrania dodatkowych doświadczeń i będzie tutaj tylko zasygnalizowany. Należy zaznaczyć, że przedstawione w niniejszym opracowaniu podejście do sposobu programowania w języku APT jest jedynie pewną propozycją, która została wypracowana na podstawie doświadczeń uzyskanych przy programowaniu w tym języku w Zakładzie Programów dla Konstrukcji, Technologii i Eksploatacji Maszyn Instytutu Maszyn Matematycznych.

Artykuł ten przeznaczony jest w zasadzie dla osób, które znają bądź uczą się języka APT i jest próbą usystematyzowania doświadczeń w postaci pewnego zbioru reguł mówiących, w jaki sposób można programować w tym bardzo wyspecjalizowanym języku. Stąd podawane w nim fakty mogą się wydać oczywiste dla doświadczonych programistów. Dokładne informacje na temat języka APT znajdują się w pozycjach [1] i [2], zaś przykłady programów napisanych w APT podaje pozycja [3].

W tym miejscu warto również podkreślić fakt, że APT jest systemem dużym, złożonym, przeznaczonym głównie do przygotowywania programów dla OSN sterowanych w 3 do 5 osiach, chociaż, oczywiście, może być używany dla OSN sterowanych w 2 lub 2 1/2 osiach. W swojej obecnej wersji nie ma możliwości automatycznego projektowania procesu obróbki od jego strony technologicznej, a więc automatyczny dobór parametrów obróbki, narzędzia itp. - te wszystkie informacje musi podać programiście. System APT ogranicza się w zasadzie jedynie do geometrycznego obliczenia drogi narzędzia, chociaż pewne funkcje, pozwalające na częściową automatyzację w zakresie projektowania procesu obróbczego, może pełnić odpowiednio napisany postprocesor.

W świetle tych uwag bardziej zrozumiałe mogą się wydać pewne stwierdzenia dotyczące kwalifikacji programisty (zob. "Dane potrzebne do opracowania programu obróbki części") bądź też zestawu danych potrzebnych do przygotowania programu obróbki części.

Ponadto będą zasygnalizowane pewne problemy związane z właściwą eksploatacją systemu APT, a wiążące się zarówno z konfiguracją sprzętową maszyny cyfrowej, na której jest eksploatowany system APT, jak i z efektywnym użytkowaniem tego systemu. Będzie również poruszona sprawa przypuszczalnych kierunków rozwoju systemu APT, wynikających z zapotrzebowania użytkowników.

Dane potrzebne do opracowania programu obróbki części. Kwalifikacje programisty

Przystępując do pisania programu obróbki części programista powinien dysponować odpowiednim zestawem informacji na temat części, jaka ma powstać w wyniku wykonania programu przez obrabiarkę, jak również samej obrabiarki.

Tak więc programiście potrzebny będzie:

- rysunek konstrukcyjny części, który wykorzysta do zdefiniowania kształtu geometrycznego części zgodnie z wymaganiami języka APT (zob. "Proces przygotowywania programu obróbki części"),
- projekt procesu technologicznego obróbki danego detalu, a w nim przede wszystkim informacje dotyczące:
  - rodzaju obrabiarki, na której będzie wytwarzany detal,
  - kolejności operacji wykonywanych podczas obróbki części wraz z ich parametrami technologicznymi, takimi jak: szybkość posuwu, szybkość obrotów wrzeciona, itp. oraz z opisem narzędzi potrzebnych do wykonania każdej operacji;
  - wymiarów materiału, z którego będzie wykonywana część oraz sposobu jego zamocowania na stole obrabiarki.

Projekt procesu technologicznego powinien być wykonany przez technologa na podstawie informacji o możliwościach technologicznych obrabiarki, o jej wyposażeniu narzędziowym, oprzyrządowaniu normalnym i specjalnym itp. Postępowanie jest w tym wypadku analogiczne, jak przy programowaniu ręcznym OSN - za wyjątkiem określania współrzędnych kolejnych punktów drogi narzędzia, gdyż wszelkie obliczenia dotyczące współrzędnych drogi narzędzia wykona automatycznie system APT na podstawie programu obróbki części.

Przedmiotem dalszych rozważań będzie omówienie postępowania programisty podczas przygotowywania programu obróbki części w języku APT (zob. "Proces przygotowywania programu obróbki części"); natomiast praca technologa nad opracowaniem projektu procesu technologicznego obróbki części nie będzie tu poruszana. Wyniki pracy technologa będą traktowane jako dane dla programisty. Pomimo że programista nie bierze udziału w opracowaniu procesu technologicznego wskazane jest, aby oprócz umiejętności programowania w języku APT, współpracy z maszyną cyfrową na poziomie użytkownika systemu oraz wiadomości na temat stosowanego programu postprocesora - posiadał również pewne wiadomości z zakresu technologii. Jest to ważne ze względu na bardzo duży związek programów obróbki części z samym procesem technologicznym, co z kolei wiąże się z koniecznością optymalizacji wykonania programu obróbki części przez OSN, natomiast program nie musi być optymalny z punktu widzenia jego wykonania przez maszynę cyfrową. Można by więc stwierdzić, że pożądana byłaby sytuacja, w której technolog, opracowujący projekt procesu technologicznego, pisałby również program obróbki części (oczywiście musiałby on znać język APT). Wówczas opracowanie projektu procesu technologicznego stanowiłoby pierwszy etap pracy programisty nad programem obróbki części. Sposób przygotowania samego programu, czyli pracę programisty omawiamy w następnym punkcie.

Proces przygotowywania programu obróbki części

Dysponując odpowiednimi danymi, potrzebnymi do opracowywania programu obróbki części, programista może przystąpić do przygotowywania tego programu.

Jak to już zostało wspomniane we wstępie, przez przygotowanie programu obróbki części będziemy rozumieć jego napisanie w języku APT, uruchomienie programu na maszynie cyfrowej oraz wygenerowanie taśmy sterującej obrabiarką, jak również sprawdzenie poprawności programu obróbki części tzn. stwierdzenie, czy w wyniku wykonania przez obrabiarkę programu opisanego taśmą sterującą otrzymamy detal zadany rysunkiem konstrukcyjnym.

W celu przedstawienia tego problemu w bardziej usystematyzowany sposób, w procesie przygotowania programu obróbki części wyróżniono kilka etapów.

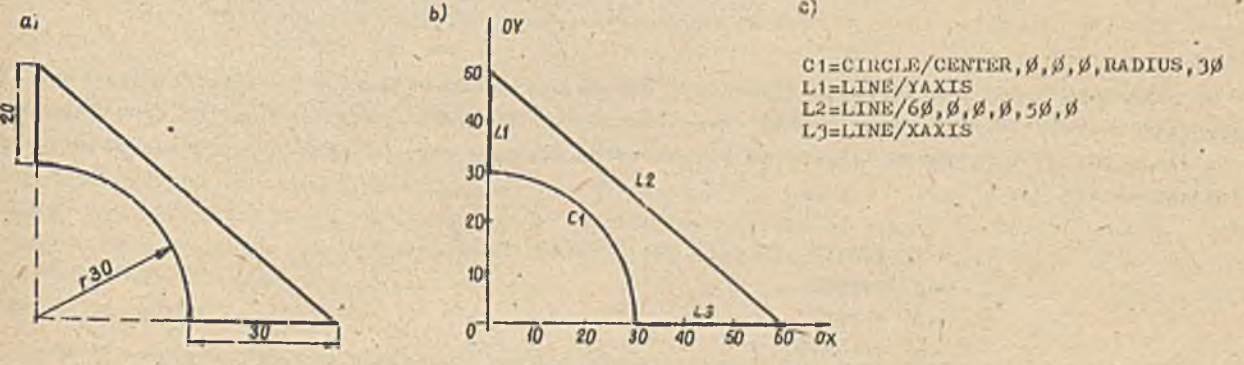
• Etap wstępny

Programista musi zaznajomić się z rysunkiem konstrukcyjnym oraz projektem procesu technologicznego. Na podstawie tych informacji powinien powstać projekt całego programu. W szczególności należy ustalić, czy obróbka części będzie opisywana przez jeden lub kilka programów (wykonywanych na jednej bądź kilku obrabiarkach), podzielić program na pewne fragmenty, które mogą być oddzielnie uruchamiane, jak również ogólnie zaprojektować sposób definiowania geometrii części i ruchu narzędzia. Po wykonaniu tych wszystkich prac wstępnych programista może przystąpić do pisania programu obróbki części.

• Etap I

Pisanie programu obróbki części

Przystępując do pisania programu obróbki części programista powinien najpierw dokonać wyboru układu współrzędnych, w którym następnie będzie opisywać kształt geometryczny części - zgodnie z wymaganiami języka APT. Przy wyborze układu współrzędnych należy przede wszystkim brać pod uwagę łatwość późniejszego definiowania kształtu - dlatego też nawet wówczas, gdy obrabiarka narzuca wybór innego układu współrzędnych, opłaca się zdefiniować geometryczny kształt części w wygodniejszym, wybranym przez programistę układzie, a następnie dokonać przekształcenia tego układu zgodnie z wymaganiami obrabiarki, wykorzystując instrukcję REFSYS bądź TRACUT.
W wybranym układzie współrzędnych, programista opisuje kształt geometryczny części, czyli podaje zbiór definicji geometrycznych (prostych, okręgów, płaszczyzn itp.), będących instrukcjami języka APT. Na tym etapie programista posługuje się danymi z rysunku konstrukcyjnego części. Powyższe wywody zilustrowano przykładem programu, napisanego w języku APT. Na rys. 1 przedstawiono schemat postępowania przy definiowaniu kształtu części. Tak więc na podstawie rysunku konstrukcyjnego (rys. 1a) programista dokonuje wyboru układu współrzędnych (rys. 1b), a następnie pisze zbiór instrukcji języka APT, definiujących poszczególne elementy części (rys. 1c).



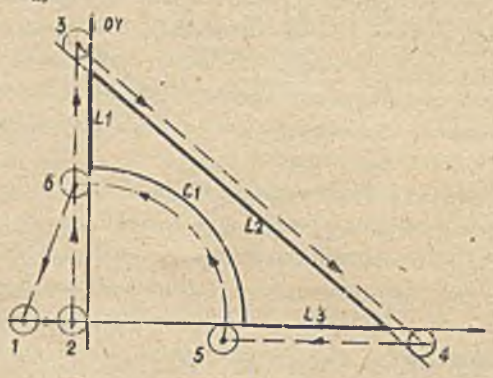
Rys. 1. Przykład definiowania kształtu geometrycznego części w języku APT
a) rysunek konstrukcyjny części,
b) rysunek części w wybranym układzie współrzędnych dla potrzeb języka APT,
c) definicje geometryczne w języku APT

Następną fazą I etapu prac nad programem obróbki części jest podanie zbioru odpowiednich instrukcji ruchu narzędzia, czyli przy wykorzystaniu uprzednio zdefiniowanych elementów geometrycznych, określenie takiej drogi narzędzia, która zapewni uzyskanie części o wymaganym kształcie. W tej fazie pracy programista musi uwzględnić kolejność operacji, podaną w projekcie procesu technologicznego, gdyż określa ona, w jaki sposób ma się poruszać narzędzie podczas obróbki przedmiotu. Na rys. 2a przedstawiono drogę narzędzia, która realizuje obróbkę części z rys. 1, natomiast na rys. 2b przedstawiono zbiór instrukcji ruchu narzędzia opisujących tę właśnie drogę.

Aby otrzymać program, który można uruchamiać w systemie APT, do definicji geometrycznych (por. rys. 1c) oraz instrukcji ruchu narzędzia (por. rys. 2b) programista musi dołączyć pewne instrukcje dodatkowe, a mianowicie:

- definiujące kształt narzędzia dla poszczególnych operacji, zgodnie z wymaganiami podanymi w projekcie procesu technologicznego (instrukcja CUTTER),
- definiujące zakres tolerancji (instrukcje INTOL, OUTTOL, TOLER), z jaką będzie aproksymowany przez system APT kształt części - programista powinien to określić zgodnie z nałożonymi wymaganiami na dokładność wykonania części, przy czym celowe jest stosowanie tolerancji takiego rzędu, jakie można uzyskać na obrabiarce podczas obróbki części,
- instrukcje o charakterze organizacyjnym, a więc określające początek i koniec programu obróbki części (odpowiednio instrukcje PARTNO i FINI), powodujące ominięcie wykonania fazy postprocesora, polecające wydrukowanie kolejnych punktów drogi narzędzia (instrukcja CLPRNT), czy dokonanie pewnych specjalnych wydruków (instrukcja PPRINT,TITTLES, PRINT) i inne.

a)



b)

```
FROM/-1Ø,Ø,Ø
GO/TO,L1
TLLFT,GOLFT/L1,PAST,L2
GORGT/L2,PAST,L3
GORGT/L3,PAST,C1
GORGT/C1,PAST,L1
GOTO/-1Ø,Ø,Ø
```

Rys. 2. Definiowanie drogi narzędzia
a) zilustrowanie drogi narzędzia - narzędzie przesuwa się wzdłuż pozycji 1-2-3-4-5-6-1
b) instrukcje ruchu narzędzia w języku APT

Po dokonaniu wyżej wymienionych czynności, uzyskujemy pierwszą wersję programu obróbki części, opisującą jedynie kształt geometryczny drogi narzędzia, nadającą się już jednak do uruchomienia w systemie APT. W rozważanym dotychczas przykładzie, program obróbki części przyjmie postać przedstawioną na rys. 3.

```
→1.  PARTNO  PRZYKLADOWY PROGRAM OBROBKI CZESCI
→2   NOPOST
→3   CLPRNT
→4   TOLER/Ø.1
  5      C1=CIRCLR/CENTER,Ø,Ø,Ø,RADIUS,3Ø
  6      L1=LINE/YAXIS
  7      L2=LINE/6Ø,Ø,Ø,Ø,5Ø,Ø
  8      L3=LINE/XAXIS
→9   CUTTER/2
 10      FROM/-1Ø,Ø,Ø
 11      GO/TO,L1
 12      TLLFT,GOLFT/L1,PAST,L2
 13      GORGT/L2,PAST,L3
 14      GORGT/L3,PAST,C1
 15      GORGT/C1,PAST,L1
 16      GOTO/-1Ø,Ø,Ø
→17  FINI
```

Rys. 3. Przykład programu obróbki części - wersja 1.
→ wskazuje instrukcje dołączone do programu złożonego z fragmentów przedstawionych na rys. 1c i 2b

• Etap II

Uruchamianie programu obróbki części na maszynie cyfrowej w systemie APT bez wykonania fazy postprocesora

Jak już zostało wyżej wspomniane, napisany w etapie I program obróbki części stanowi fragment całego programu, który to fragment decyduje o geometrycznym kształcie obrabianej części, z całkowitym pominięciem instrukcji definiujących parametry technologiczne obróbki. Wskazane jest jednak uruchamianie na maszynie cyfrowej takiego właśnie okrojonego programu w celu wyeliminowania błędów o charakterze geometrycznym. Błędy takie z pewnością wystąpią w programie obróbki części, niekorzystne jest więc przedłużanie czasu wykonania programu podczas uruchamiania jeszcze o fazę wykonania programu postprocesora, gdyż w tej sytuacji nie wnosi to żadnych nowych informacji.

System APT ma dosyć rozbudowaną diagnostykę błędów, sygnalizuje różnego rodzaju sprzeczności w definicjach geometrycznych, bądź w instrukcjach ruchu. Należy jednak podkreślić, że usunięcie z programu obróbki części wszystkich błędów wykrywanych przez system APT, czyli otrzymanie formalnie poprawnego programu nie gwarantuje jeszcze, że opisany w programie kształt części będzie zgodny z jej rysunkiem konstrukcyjnym. Może np. zaistnieć sytuacja, w której na skutek podania przez programistę błędnych wymiarów w definicjach geometrycznych, program będzie opisywał część o kształcie innym, niż na rysunku konstrukcyjnym. Tego typu błędów nie może wykryć system APT - może je wyeliminować jedynie programista, dokonując sprawdzenia poprawności geometrycznej programu, polegającej na władaniu, czy opisany w programie kształt części jest zgodny z zadanym rysunkiem konstrukcyjnym.

Sprawdzanie geometrycznej poprawności programu obróbki części można zrealizować przez wydrukowanie i sprawdzenie położenia kolejnych punktów drogi narzędzia (co jest postępowaniem niezwykle pracochłonnym), bądź też przez wykreślenie drogi narzędzia na ploterze, co znacznie ułatwia pracę. Najwygodniej jest, gdy ploter jest podłączony bezpośrednio (on-line) do maszyny cyfrowej, na której uruchamiany jest program. Wówczas przez wykorzystanie instrukcji języka APT, takich jak PLOT i NOPLOT, można na bieżąco wykreślać interesujące fragmenty drogi narzędzia i w ten sposób dokonywać kontroli poprawności geometrycznej. W sytuacji, gdy nie ma plotera w konfiguracji maszyny cyfrowej, należy zapisać kolejne punkty drogi narzędzia (generowanie przez system APT) na odpowiednim nośniku (np. taśmie magnetycznej) i następnie wykreślić je na ploterze zewnętrznym.

W wypadku wykrycia błędów w programie obróbki części, czy to o charakterze formalnym (a więc sygnalizowanych odpowiednim komunikatem przez system APT), czy też polegających na niezgodnym z wymaganym kształtem drogi narzędzia, programista powinien zbadać przyczynę tego błędu, usunąć ją, a następnie ponownie uruchomić program. Programista musi więc powrócić do I etapu przygotowywania programu obróbki części i dokonać poprawek w definicjach geometrycznych, czy też w instrukcjach ruchu narzędzia.

Warto też zauważyć, że w czasie uruchamiania programu można zrezygnować z prowadzenia obliczeń z dużą dokładnością i zwiększyć tolerancję (oczywiście do rozsądnej wielkości). Zmniejsza się wówczas liczba koniecznych obliczeń przeprowadzanych przez system APT, a przez to znacznie zaoszczędza się czas obliczeniowy maszyny cyfrowej, jak również zmniejsza objętość wydruku punktów drogi narzędzia (generowanego przez system APT), który programista musi przejrzeć.

Na rys. 4 przedstawiono wydruk kolejnych punktów drogi narzędzia dla rozważanego przykładu programu obróbki części (uzyskanie tego wydruku jest spowodowane umieszczeniem w programie instrukcji CLPRNT).

Reasumując, efektem wykonania etapu II powinno być uzyskanie poprawnego pod względem geometrycznym programu obróbki części. Wówczas można przystąpić do prac następnych, czyli włączenia do programowania elementów związanych z parametrami technologicznymi kolejnych operacji w procesie obróbki.

• Etap III

Dołączenie instrukcji postprocesora

Dysponując uruchomionym programem obróbki części, który zapewnia, że wykonana część będzie miała

```
*****SECTION 3*****

PARTNO    PRZYKLADOWY [illegible] CZESCI

          OUTTOL/   0.100000  0.100000  0.100000  0.100000
          INTOL/    0 0       0 0       0 0       0 0

          CUTTER/   2.000000

          *** FROM/      POINT (    0)
                                    10.0000000      0.0             0.0

          *** GO---/     L1 (    0)
                                    1.0000000       0.0             0.0

          *** GO---/     L1 (    0)
                                    1 0000000       52.1350416      0.0

          *** GO---/     L2 (    0)
                                    62 7620499      1 0000000       0.0

          *** GO---/     L3 (    0)
                                    28 9827535      1 0000000       0 0

          SURFACE DATA - CYCLE (  DS  ) **      C1  (    0) ** IMPL  TO **
                CAUTION PROFILE      0 0     ,     0 0     ,     0 0     ,   0 0     ,     0 0     ,   1 0000000,
                                    30.0000000,
          *** GO---/     C1 (    0)
                                    28 9672917      1 3769583       0.0
                                    28.3539383      0.0872149       0.0
                                    26.5798529      10 6341610      0 0
                                    24 8820190      14 8958091      0.0
                                    22 1165994      16.7578259      0.0
                                    18.7578259      22 1165994      0.0
                                    14.8958091      24 8820190      0.0
                                    10 6341610      26 5798529      0 0
                                     0.0872149      28.3539383      0.0
                                     1 3769583      28 9672917      0 0
                                    -1.0000000      28 9827535      0.0

          *** GO-- /     POINT (    0)
                                    10 0000000      0 0             0.0

          END,

          *** FINI ***

          END OF SECTION
```

```
RECND     1 CARD      SEQ     1
RECND     3 CARD      SEQ     4
RECND     6 CARD      SEQ     9
RECND     8 CARD      SEQ    10
RECND    10 CARD      SEQ    11
RECND    12 CARD      SEQ    12
RECND    14 CARD      SEQ    13
RECND    16 CARD      SEQ    14
RECND    18 CARD      SEQ    15
RECND    21 CARD      SEQ    16
RECND    23 CARD      SEQ    17
RECND    25 CARD      SEQ    18
```

Rys.4. Wydruk kolejnych punktów drogi narzędzia w przykładzie programu obróbki ozębień wygenerowany przez system APT

wymagany kształt geometryczny, programista powinien umieścić w opracowywanym programie instrukcje definiujące parametry technologiczne procesu obróbki takie, jak szybkość posuwu, szybkość obrotów wrzeciona, włączenie lub wyłączenie dopływu chłodziwa itp.

W systemie APT wszelkie funkcje wymagające dostosowania wyników podawanych przez procesor APT do wymagań konkretnej obrabiarki pełni postprocesor. Są to: zamiana kodu znaków na taśmie sterującej na kod wymagany przez obrabiarkę, sprawdzenie czy nie są przekroczone pewne ograniczenia narzucone przez obrabiarkę, jak również umieszczanie na taśmie sterującej symboli wymaganych przez układ sterowania obrabiarki, a m.in. ustawiających wartości parametrów technologicznych procesu obróbki. Instrukcje definiujące wartości tych parametrów, tzw. instrukcje postprocesora są przetwarzane przez postprocesor, a postać ich musi być zgodna z jego wymaganiami. Wartości parametrów technologicznych, jak również kolejność, w jakiej mają wystąpić w programie obróbki części pochodzi z projektu procesu technologicznego. Należy również dołączyć instrukcję powodującą wywołanie odpowiedniego postprocesora (instrukcją MACHIN). W ten sposób uzyskujemy kompletny program obróbki części. Na rys. 5 przedstawiono taki program dla rozważanego przez nas przykładu.

```
   1  PARTNO  PRZYKLADOWY PROGRAM OBROBKI CZESCI
   2  CLPRNT
-->3  MACHIN/POST
   4  TOLER/Ø.1
   5      C1=CIRCLE/CENTER,Ø,Ø,Ø,RADIUS,3Ø
   6      L1=LINE/YAXIS
   7      L2=LINE/6Ø,Ø,Ø,Ø,5Ø,Ø
   8      L3=LINE/XAXIS
-->9  SPINDL/16ØØ,CCLW
-->10 FEDRAT/2ØØ
  11  CUTTER/2
  12     FROM/-1Ø,Ø,Ø
  13     GO/TO,L1
  14     TLLFT,GOLFT/L1,PAST,L2
  15     GORGT/L2,PAST,L3
  16     GORGT/L3,PAST,C1
  17     GORGT/C1,PAST,L1
  18     GOTO/-1Ø,Ø,Ø
-->19  END
  20  FINI
```

Rys. 5. Przykład programu obróbki części (kompletny program)
—► wskazuje instrukcje dołączone do programu obróbki części w etapie III.

• Etap IV

Uruchomienie programu obróbki części na maszynie cyfrowej z wykonaniem fazy postprocesora

Kompletny program obróbki części powinien programista uruchomić na maszynie cyfrowej, przechodząc tym razem również przez fazę postprocesora. Na tym etapie błędy występują zwykle w instrukcjach definiujących parametry technologiczne procesu obróbki, np. zostają przekroczone możliwości obrabiarki, co sygnalizuje postprocesor. Może się też zdarzyć, że poprawienie błędu będzie wymagało powrotu na etap pisania programu i dokonania poprawek np. w definicjach geometrycznych, czy też instrukcjach ruchu. Np. dzieje się tak wówczas, gdy wykonamy ruch narzędziem poza obszar dopuszczalny dla danej obrabiarki.

Oczywiście rodzaj i zakres błędów, sygnalizowanych przez postprocesor, zależy od konkretnego postprocesora i warto zdawać sobie sprawę, że różnią się one znacznie pod tym względem. Na tym etapie programista uzyskuje taśmę sterującą obrabiarką, czyli przetworzony program obróbki części do postaci przyswajalnej przez OSN.

- Etap V

Próbne wykonanie programu obróbki części na OSN

Po otrzymaniu taśmy sterującej obrabiarką, należy sprawdzić poprawność programu obróbki części przez jego wykonanie na OSN początkowo w materiale próbnym, a następnie przez wykonanie prototypu we właściwym materiale i w obu wypadkach dokonanie pomiarów otrzymanej części. Nieraz też pierwszą fazą sprawdzania programu na obrabiarce jest próbne wykonanie programu obróbki części bez zamocowanego materiału - jedynie w celu sprawdzenia, czy narzędzie nie wykonuje pewnych ruchów powodujących przekroczenie ograniczeń narzucanych przez obrabiarkę. Jeżeli okaże się, że otrzymana część spełnia założone wymagania, to program obróbki można uznać za poprawny. W przeciwnym razie, gdy wymiary detalu są niezgodne z żądanymi lub wystąpiły inne nieprawidłowości w programie obróbki części, np. należy skorygować wartość pewnych parametrów technologicznych, to programista musi powrócić na odpowiednio wcześniejszy etap przygotowywania programu obróbki części, usunąć przyczyny błędów i przejść do kolejnego etapu (pewne układy sterowania OSN pozwalają na wprowadzanie poprawek w aktualnie wykonywanym programie, przez ręczne wprowadzenie z pulpitu operatorskiego potrzebnego bloku, bądź też usunięcie błędnego bloku z taśmy sterującej).

Proces przygotowywania programu obróbki części schematycznie przedstawiono na rys. 6.

Równolegle, na wszystkich etapach powstawania programu obróbki części, powinna być tworzona jego dokumentacja, zarówno w postaci opisu, jak i wybranych wydruków programu. Ze względu na swoją złożoność, problem ten nie będzie w niniejszym opracowaniu omawiany.

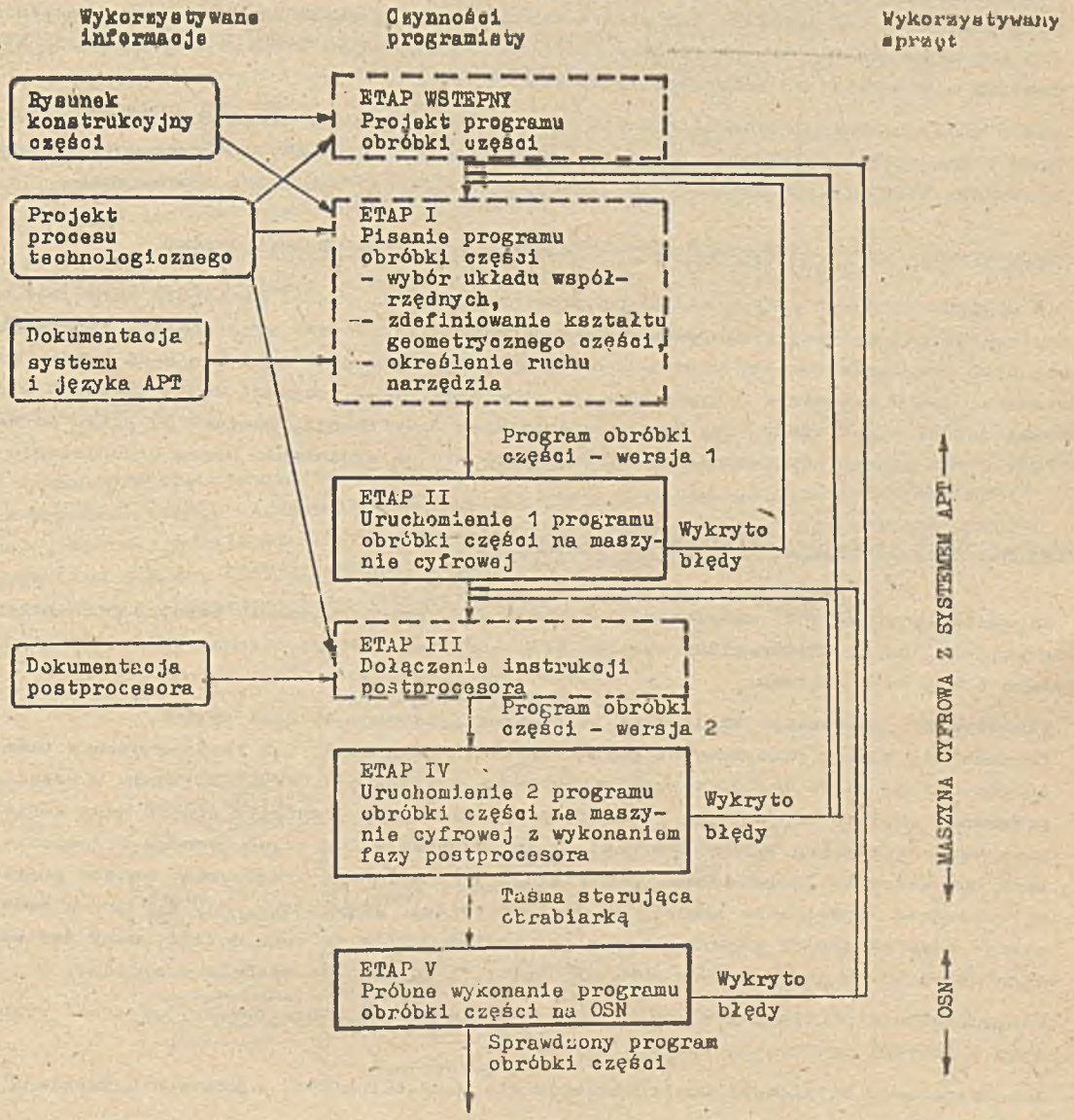
Sposób zorganizowania programu obróbki części

Programy obróbki części, napisane w języku APT, z reguły opisują wykonanie złożonych detali, w związku z tym są one długie, liczą zwykle od kilkuset do kilku tysięcy instrukcji, chociaż struktura ich jest dosyć prosta. Warto więc podać kilka uwag, które w pewnym stopniu mogą ułatwić pracę podczas pisania i uruchamiania tych długich programów.

Zapewnienie przejrzystej struktury programu obróbki części

Jak już to zostało wyżej wspomniane, programy obróbki części, napisane w języku APT, są długie - co między innymi utrudnia odszukiwanie potrzebnych fragmentów, poprawianie błędów itd. - dlatego celowe jest więc zapewnienie przejrzystego, czytelnego wydruku programu. Można to łatwo uzyskać przez:

- zgrupowanie definicji geometrycznych w jednym lub kilku wybranych miejscach programu - podobnie z instrukcjami ruchu,
- przyjęcie określonej nomenklatury przy nadawaniu nazw elementom geometrycznym.
  Na przykład można przyjąć, że:
  Pnn - oznacza nazwę punktu, przy czym nn jest numerem kolejnego punktu, natomiast
  Lnn - oznacza nazwę linii, przy czym nn jest numerem kolejnym linii, itp.
  W ten sposób łatwiej uniknąć kolizji nazw elementów, szczególnie przy składaniu programu obróbki części z kilku wcześniej uruchomionych fragmentów; o ile wchodzące w skład nazw elementów geometrycznych kolejne numery pochodziły z różnych przedziałów dla poszczególnych fragmentów. Łatwiej jest również zorientować się, czy występująca w programie zmienna oznacza prostą, punkt, płaszczyznę itp. Jest to szczególnie ważne, gdy nazwy te występują w instrukcjach ruchu.
- Przyjęcie pewnych zasad dotyczących formatu zapisu instrukcji. Język APT pozwala na zapis instrukcji w dowolnym formacie - nie jest więc ustalone, od której kolumny muszą rozpoczynać się etykiety, instrukcje. Wyjątek stanowi kilka słów specjalnych, których zapis musi rozpoczynać się od pierwszej kolumny. W celu łatwiejszego odszukiwania potrzebnych fragmentów programu, wygodnie jest przyjąć, aby np.:
  - definicje geometryczne rozpoczynały się od 6 kolumny,
  - instrukcje ruchu narzędzia i przesuwu wstępnego rozpoczynały się od 3 kolumny,
  - instrukcje postprocesora, definiujące parametry technologiczne rozpoczynały się od 1 kolumny itp.



Rys. 6. Schemat procesu przygotowywania programu obróbki części. Prostokąty narysowane liniami przerywanymi oznaczają czynności programisty, przy których nie musi wykorzystywać dodatkowego sprzętu, zaś narysowane liniami ciągłymi - czynności wymagające wykorzystania dodatkowego sprzętu (maszyny cyfrowej lub OSN).

Zgodnie z powyższymi zasadami jest napisany przykład programu obróbki części, przedstawiony na rys. 5. Chociaż program ten jest bardzo krótki, widać jednak, jak w wyniku tych zabiegów zyskał na czytelności.

Warto w tym miejscu również podkreślić ogromną rolę komentarzy i opisów, umieszczonych przez programistę w długich programach obróbki części.

Maksymalne wykorzystanie możliwości języka APT

APT jest językiem złożonym o dużej liczbie instrukcji, które przy umiejętnym zastosowaniu bardzo ułatwiają pracę programisty, co w efekcie znacznie skraca czas przygotowania programu obróbki części. Należy więc podkreślić celowość:

- stosowania pętli - przy kolejnym wykonywaniu zespołu takich samych lub bardzo podobnych instrukcji,

- stosowania makroinstrukcji, gdy pewne czynności wykonuje się wielokrotnie, w różnych miejscach programu, bądź też istnieją już gotowe makroinstrukcje zapisane w bibliotece makroinstrukcji lub też napisane dla innego programu obróbki części, które w takim wypadku warto wykorzystać chociaż w programie są potrzebne tylko raz,
- stosowania instrukcji przekształcania bądź kopiowania drogi narzędzia TRACUT, COPY , aby uniknąć w ten sposób opisywania drogi narzędzia w złożony sposób, w niewygodnym układzie współrzędnych, bądź też powtarzania drogi narzędzia opisanej wcześniej w programie.

Wykorzystywanie gotowych fragmentów istniejących programów obróbki części

Ze względu na dosyć prostą strukturę programów obróbki części (z reguły mało jest w nim skoków i instrukcje najczęściej wykonywane są sekwencyjnie) łatwo jest wyodrębnić z istniejących i uruchomionych programów obróbki części pewne fragmenty, opisujące taki sam bądź bardzo podobny ruch narzędzia, jak w aktualnie przygotowywanym programie. Taki fragment warto umieścić w programie obróbki części nawet wtedy, gdy wymaga niewielkich modyfikacji, zamiast go pisać od nowa, narażając się w ten sposób na popełnienie błędu i przez to na wydłużenie czasu uruchamiania programu.

Uwagi na temat właściwej eksploatacji systemu APT

Zapewnienie efektywnej eksploatacji systemu APT wymaga wykonania pewnych prac przez samych programistów, czyli użytkowników systemu APT, bądź też obsługę ośrodka obliczeniowego dbającą o sprawną pracę tego systemu.

- Właściwe zorganizowanie biblioteki tworzonych programów obróbki części.
  Powinno się w nich przechowywać postać źródłową programów, jak również obrazy taśm sterujących, co pozwala na łatwe ich odtwarzanie w wypadku zniszczenia. Przechowywanie w ustalonym miejscu programów obróbki części, jak również opisywanie ich w jednolity sposób jest konieczne, gdy mamy dużą liczbę (np. kilka tysięcy) części wchodzących w skład jednego urządzenia. Wówczas przy wprowadzaniu jakichkolwiek modyfikacji w konstrukcji urządzenia, należy poprawić programy opisujące odpowiednie części, co jest oczywiście łatwiejsze, gdy np. przez podanie nazwy całego zespołu części składającego się z sąsiadujących ze sobą detali, mamy dostęp do wszystkich opisujących go programów obróbki części, i gdy są one właściwie opisane.
- Zorganizowanie biblioteki makroinstrukcji systemowych, które powinny opisywać wykonanie pewnych typowych, powtarzających się czynności.
- Zorganizowanie biblioteki postprocesorów dla wszystkich OSN, z których korzystają użytkownicy systemu APT.

W niniejszym opracowaniu omawiany był sposób przygotowywania pojedynczych programów obróbki części, a więc takich, z których każdy opisuje obróbkę dokładnie jednej części. W sytuacji, gdy mamy do czynienia z kilkoma częściami o podobnym kształcie, różniącymi się tylko pewnymi wymiarami, można napisać program uogólniony, który będzie opisywać obróbkę wszystkich tych części. Napisanie takiego programu polega na podaniu w programie - w postaci parametrów - tych wymiarów, które są różne dla poszczególnych części. Przez wykonanie programów obróbki części dla różnych wartości parametrów, uzyskuje się taśmy sterujące, opisujące obróbkę kilku części. Stosując tę metodę zmniejsza się liczbę potrzebnych do napisania programów, natomiast są one bardziej skomplikowane, a więc trudniejsze do napisania. W efekcie pisanie programów uogólnionych jest opłacalne, chociaż wymaga zatrudnienia wyżej kwalifikowanych programistów.

Dla potrzeb przedsiębiorstwa, produkującego podobne części, ilość informacji podawanych w programach obróbki części można znacznie zmniejszyć. W tym celu, dla poszczególnych typów części, należy napisać specjalne programy zwane preprocesorami*), które na podstawie podawanych danych, dotyczących szczegółowych informacji na temat części określonego typu, a więc głównie jej charakterystycznych wymiarów, będą generować programy obróbki części w języku APT.

*) Przykładem preprocesora może być syntetyzer ABST, opisujący obróbkę żeber lotniczych, opracowany w ZDIMM [4].

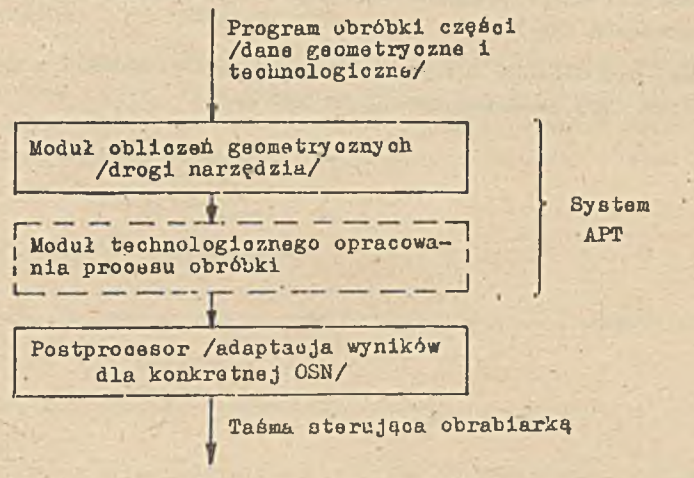
Z badań przeprowadzonych przez BAC (British Airoraft Corporation) wynika, że przy zastosowaniu preprocesorów objętość programu, który musi napisać programista wynosi ok. 20% objętości tego programu obróbki ozęści, który musiałby on napisać nie stosując preprocesora, przy ozym możliwe jest także nawet dziesięciokrotne skrócenie programu.

Głównymi zaletami stosowania preprocesorów jest więc znaczne skrócenie czasu potrzebnego na przygotowanie programu obróbki części, ze względu na mniejszą pracochłonność, jak również zmniejszenie ewentualności popełnienia błędu przez programistę, który musi napisać krótszy program.

Oczywiście, preprocesory takie muszą być najpierw napisane, ale przy dużym zapotrzebowaniu na programy opisujące obróbkę specjalnych części, włożony wkład pracy w napisanie preprocesorów jest z pewnością opłacalny.

Kierunki rozwoju systemu APT

System APT wymaga przeprowadzenia określonych prac nad jego rozszerzeniem w pewnych kierunkach, wynikających z zapotrzebowania użytkowników:przede wszystkim dobudowy modułów technologicznych do systemu. W swojej obecnej wersji system nie ma możliwości automatycznego projektowania procesu technologicznego (tak jak np. EXAPT), podobnie możliwości dotyczące opisu technologii procesu obróbki są bardzo skromne. Należałoby więc stworzyć strukturę przedstawioną na rys. 7.



Rys. 7. Możliwa przyszła struktura systemu APT

Należałoby również dobudować, bądź rozbudować istniejące mechanizmy kontroli poprawności programów obróbki części w systemie APT. Oczywiście prace takie mogą być prowadzone w zasadzie tylko przez twórców systemu.

Innym możliwym kierunkiem rozwoju tego systemu jest powstanie systemów konwersacyjnych.

Przypuszczalnie, prowadzenie tego typu prac nad systemem APT zwiększyłoby efektywność jego wykorzystania.

Uwagi dotyczące konfiguracji systemu cyfrowego

Trudno sobie wyobrazić skuteczną eksploatację systemu APT bez spełnienia pewnych wymagań dotyczących konfiguracji systemu cyfrowego, w którym jest on użytkowany. Należy tu przede wszystkim

zapewnić odpowiednio duży rozmiar pamięci operacyjnej ze względu na to, że system APT potrzebuje dużego obszaru pamięci, zaś pozostała jej część musi być wystarczająca dla wykonywania nawet dużych programów obróbki części. W przeciwnym wypadku należałoby nakładać ograniczenia objętości wykonywanych programów, co byłoby źródłem dodatkowych kłopotów dla programistów.

Należałoby również zapewnić odpowiednią liczbę stacji pamięci zewnętrznych (dyskowych lub taśmowych), w celu przechowywania bibliotek makroinstrukcji systemowych, postprocesorów, gotowych programów obróbki części, jak również zbiorów zawierających aktualnie uruchamiane programy obróbki części - co jest konieczne ze względu na ich dużą objętość.

Warto w tym miejscu jeszcze raz podkreślić użyteczność plotera w systemie cyfrowym. Ze względu na specyfikę systemu APT jest on bardzo pomocny podczas uruchamiania programów obróbki części, głównie do sprawdzania ich poprawności geometrycznej, oraz przy tworzeniu dokumentacji programów.

Literatura

[1] BONKOWICZ-SITTAUER S., GUTOWSKA H.: System APT. Biuletyn Informacyjny OSK 1977 nr 3-4, s. 39-53

[2] GUTOWSKA H., KWAŚNIEWSKA G., BOBIŃSKA I.: APT - opis języka. Warszawa: IMM 1977 Archiwum Opracowań IMM nr 18

[3] GUTOWSKA H., BOBIŃSKA I.: Zbiór programów testujących instrukcje języka APT. Warszawa: IMM 1977 Archiwum Opracowań IMM nr 56

[4] BERTHOLD A., TOPOLSKI S., GALICKI J.: System generowania programów w języku APT SYNTETYZER ADST. Warszawa: IMM 1979 Archiwum Opracowań IMM nr 66

mgr inż. Jan WRONA

Biuro Generalnych Dostaw CNPTKiP
Pracownia Projektowa

Próba przedstawienia wieloprocesorowej wewnętrznej organizacji wyspecjalizowanego minikomputera sterującego systemami wytwarzania

Wstęp

Budowa obiektów sterowanych z jednego ośrodka dyspozycyjnego jest zadaniem niezwykle istotnym dla dalszego rozwoju techniki wytwarzania. Powodzenie budowy takich systemów zależeć będzie przede wszystkim od niskich kosztów budowy i eksploatacji. Decydującym czynnikiem będą, oprócz ceny komputera, dodatkowo niezbędne urządzenia oraz liczba połączeń z poszczególnymi urządzeniami, a także możliwość szybkiego i prostego generowania programów zarządzających pracą systemu i programów wykonawczych dla poszczególnych automatów. Niemniej ważnym czynnikiem jest możliwość stałej rozbudowy, a więc dołączania nowych urządzeń do systemu.

Budowa dotychczas stosowanych systemów sterowania jest oparta na standardowych minikomputerach. Nieprzystosowanie ich do komunikowania się z dużą liczbą urządzeń, zazwyczaj bardziej oddalonych od komputera niż dopuszcza to organizacja interfejsu, narzuca konieczność stosowania specjalnych dodatkowych rozwiązań, np. adresowanie pierwszymi słowami przesyłanej informacji czy też stosowania transmisji cyklicznych do zbierania danych.

W artykule będzie omówiony sposób organizacji specjalizowanego minikomputera, przeznaczonego do sterowania zespołami automatów i urządzeń pomocniczych tworzących centra produkcyjne. Rozpatrzmy to zagadnienie na przykładzie centrum obróbki metali, w skład którego, oprócz obrabiarek sterowanych numerycznie, wchodzą stanowiska pomiarowe i sterowany system transportu.

Wymagania dotyczące komputera sterującego

W tym punkcie będą przedstawione funkcje, które powinien spełniać komputer sterujący.

Generalnie przyjęto trzy założenia:

- komputer jest przeznaczony do sterowania systemami wytwarzania i spełnia wszystkie niezbędne funkcje sterowania,
- sprzętowa rozbudowa systemu sprowadza się tylko do podłączania nowych urządzeń za pośrednictwem kabli,
- budowa komputera powinna umożliwiać bezpośrednie instalowanie go w halach produkcyjnych bądź w ich pobliżu.

Spełnienie trzeciego warunku wyklucza praktycznie stosowanie zewnętrznych pamięci magnetycznych z ruchomym nośnikiem.

Zadania komputera sterującego

Komputer sterujący systemem - w naszym przykładzie centrum obróbki metali - powinien realizować między innymi zadania:

- sterować procesem obróbki na poszczególnych automatach przez wysyłanie programów wykonawczych oraz prowadzić ich korektę na bieżąco;
- sterować transportem detali obrabianych oraz pracą automatów zgodnie z zadanym programem;
- sterować pracą stanowisk pomiarowych, zbierać z nich dane i na ich podstawie korygować programy wykonawcze obrabiarek i program sterujący transportem;
- nadzorować zużycie narzędzi oraz podejmować decyzje o ich wymianie;

- prowadzić statystykę pracy systemu oraz wykonywanych detali.

Przyjmujemy, że przygotowywanie programów wykonawczych dla automatów, sterowania pracą systemu i statystycznych będzie się odbywało na innym komputerze. Przygotowanie programów wykonawczych dla automatów uwzględniających optymalne wykorzystanie pracy systemu wymaga komputera o znacznie większych możliwościach obliczeniowych, niż to jest niezbędne do sterowania. Wskazane jest, żeby komputer przygotowujący programy wykonawcze był podłączony linią transmisji na stałe z komputerem sterującym.

Rodzaje informacji przesyłanej między komputerem a urządzeniami

Informację przesyłaną między urządzeniami a komputerem możemy podzielić na dwie grupy:

- programy wykonawcze sterujące pracą automatów charakteryzujące się przesyłaniem dużej ilości informacji, przesyłanej blokami, przy czym w blokach zawarte są polecenia np. o
  - rodzaju narzędzia,
  - głębokości obróbki,
  - szybkości przesuwu narzędzia i detalu,
  - ustawieniu powierzchni obrabianej detalu;
- polecenia do i dane od urządzeń systemu charakteryzujące się przesyłaniem małej ilości informacji w jednej sekwencji, np.
  - dane o stanie automatów,
  - polecenia do i dane ze stanowisk pomiarowych,
  - polecenia i dane z systemu transportu,
  - dane o stanie systemu i dane statystyczne przesyłane do stanowisk dyspozycyjnych,
  - polecenia ze stanowisk dyspozycyjnych.

Specjalne wymagania dotyczące komputera w systemie sterowania

Komputer sterujący systemem wytwarzania powinien spełniać specjalne wymagania, a mianowicie:

- mieć odpowiednio dużą liczbę urządzeń wejścia/wyjścia,
- umożliwiać przesyłanie i bezpośrednie czytanie informacji ze znacznych odległości,
- umożliwiać czasową kontrolę wykonywania rozkazów przez urządzenia,
- umożliwiać programowe testowanie transmisji oraz urządzeń,
- mieć system przesyłania informacji odporny na zakłócenia.

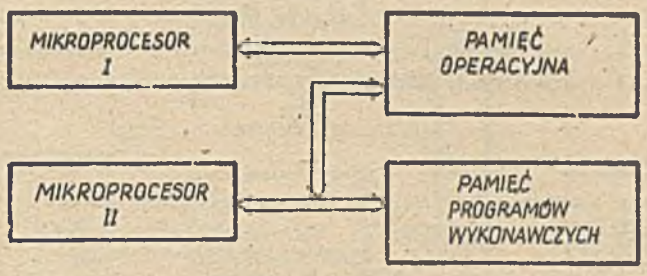
Budowa wewnętrzna komputera

Ogólne założenia

Dążenie do jak najniższych kosztów budowy komputera narzuca zastosowanie układów dużej skali integracji, a więc przede wszystkim mikroprocesorów.

Wymagania dotyczące instalacji komputera w halach produkcyjnych wykluczają możliwość stosowania zewnętrznych pamięci dyskowych do przechowywania programów wykonawczych dla automatów, a więc należy przewidzieć odpowiednio większą pamięć wewnętrzną komputera. W praktyce - w systemach wytwarzania nie prowadzi się jednoczesnego wykonywania więcej niż pięciu różnych detali. Większa ich liczba prowadzi, oprócz znacznych trudności w ułożeniu harmonogramu pracy z optymalnym wykorzystaniem maszyn, do utrudnień w obsłudze stanowisk i znacznie częstszego występowania błędów. Do przechowywania tylu programów wykonawczych (poza wyjątkami) zupełnie wystarczają 64 kbajty pamięci, jednak nic nie stoi na przeszkodzie, aby w miarę potrzeby wynosiła ona N x 64 kbajty. Jeżeli przyjmiemy, że będziemy stosowali rejestry dynamiczne (obecnie seryjnie są produkowane pojedyncze układy o pojemności 64k bity) koszty takiej pamięci będą niskie. Jest rzeczą oczywistą, że w pamięci będą przechowywane tylko programy źródłowe, zaś korekty i poprawki do nich wprowadzane będą na bieżąco. Wskazane jest w tym wypadku zastosowanie drugiego mikroprocesora, którego zada-

niem będzie sterowanie wysyłaniem programów wykonawczych oraz ich korektą. Schemat takiego rozwiązania pokazany jest na rys. 1.

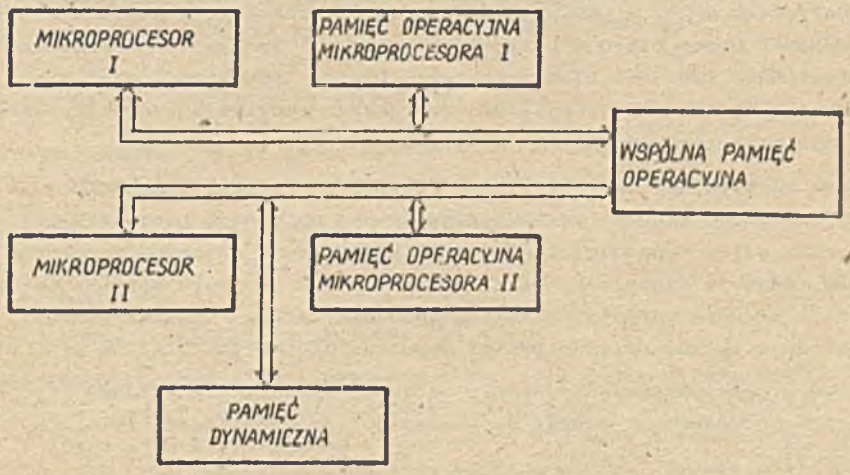


Rys. 1.

Przyjęcie dwóch mikroprocesorów ma tę zaletę, że pozwala rozdzielić w komputerze podstawowe dwie funkcje, tj. sterowanie systemem i sterowanie pracą automatów. Ułatwia to pisanie programów, a przy tym duże zapasy czasu przy obliczeniach pozwalają na sterowanie dużymi systemami.

Organizacja pamięci wewnętrznej

Przyjęcie dwóch mikroprocesorów wymaga rozbudowy pamięci wewnętrznej. Przyjmuje się, że każdy mikroprocesor jest wyposażony we własną pamięć operacyjną, oprócz tego jest podłączony do wspólnej pamięci operacyjnej. Pamięci te są zbudowane na układach RAM i ROM. Oprócz tego mikroprocesor drugi jest wyposażony w pamięć dynamiczną programów wykonawczych zbudowaną na rejestrach dynamicznych. Maksymalna wielkość pamięci operacyjnej może wynosić 64 kbajty, część adresów w tym wypadku będzie zajęta przez pamięci indywidualne, pozostała może być przeznaczona na pamięć wspólną. Schemat organizacji pamięci wewnętrznej pokazany jest na rys. 2.



Rys. 2.

Organizacja wspólnej pamięci operacyjnej

Praca dwóch mikroprocesorów z wspólną pamięcią operacyjną wymaga specjalnego rozwiązania układowego, które pokazane jest na rys. 3.

Szyny DANYCH i ADRESOWA MIKROPROCESORÓW (1,2) podłączone są poprzez PRZEŁĄCZNICE (7) do WSPÓLNEJ PAMIĘCI OPERACYJNEJ (8). Przełączaniem ich steruje KONTROLER DOSTĘPU DO PAMIĘCI (9) na podstawie sygnału "CZYTAJ", "PISZ" otrzymywanych z KONTROLERÓW PRACY MIKROPROCESORA (5,6).

żoli uwzględnimy, że w typowych rejestrach dynamicznych ŚREDNI CZAS DOSTĘPU do dowolnej komórki jest ok. 100 μs, straty czasu nie są zbyt duże.

Po wybraniu pierwszej komórki, o zadanym adresie, następne przesłania odbywają się już synchronicznie z programem przesyłania mikroprocesora, a więc nie różnią się od współpracy z pamięcią operacyjną. Jak to się dzieje, zobaczmy na przykładzie czytania informacji z pamięci dynamicznej i przesyłania jej do pamięci operacyjnej, przy czym zakładamy, że komórka pamięci dynamicznej już została wybrana.

Załóżmy, że program jest następujący:

- POBIERZDO REJ.M wlg adresu w REJ(AB) - z pamięci dynamicznej
- PRZESLIJ z REJ.M wlg adresu w REJ(CD) - do pamięci operacyjnej
- PORÓWNA zawartość REJ.M i N
  gdy M ≠ N SKOCZ do 1
  gdy M = N SKOCZ x (dalszy program)

przy czym w rejestrze N jest znak końca bloku.

Jeżeli teraz przyjmiemy, że zegar taktujący pamięcią dynamiczną napędzany jest impulsami zegara mikroprocesora, a w momencie pobierania informacji, częstotliwość impulsów taktujących będzie zmniejszona o liczbę taktów zegara mikroprocesora potrzebnych do wykonania przedstawionego wyżej programu, to pobranie informacji z następnej komórki wypadnie w chwili wybrania jej przez zegar taktujący pamięci. Oczywiście, pobranie informacji z pamięci operacyjnej i przesłanie jej do pamięci dynamicznej powinno wyglądać analogicznie.

Prostota konstrukcji wymaga, aby programy przesyłania między pamięciami były niezmienne, a więc - najlepiej - umieszczone w pamięci stałej mikroprocesora.

Przyjęta wyżej zasada odwoływania się bezpośrednio rozkazami do komórek obu pamięci wymaga możliwości ich rozróżniania, a także rozróżniania poszczególnych bloków po 64 kbajty pamięci dynamicznej. Stosowanie specjalnych nielegalnych rozkazów, pomijając trudności konstrukcyjne w układach z mikroprocesorami, jest niewskazane przede wszystkim dlatego, że wyklucza przygotowywanie programów na typowych komputerach zbudowanych na tych samych mikroprocesorach. Ponieważ przyjęliśmy, że programy dotyczące pracy z pamięcią dynamiczną zapisane są w pamięci stałej, możemy adresowi ich przyporządkować numer bloku w pamięci dynamicznej. Wymaga to umieszczenia w pamięci tylu identycznych programów, ilo jest bloków pamięci, biorąc jednak pod uwagę niski koszt układów ROM nie ma to większego znaczenia. Schemat blokowy układu umożliwiającego bezpośredni dostęp do komórek pamięci dynamicznej pokazany jest na rys. 4.

Praca układu jest następująca: gdy KONTROLER DOSTĘPU DO PAMIĘCI DYNAMICZNEJ (4) stwierdzi, że rozkaz przesłania pochodzi z komórki pamięci przypisanej do danego bloku PAMIĘCI DYNAMICZNEJ (6), powoduje: przyłączenie szyny adresowej i DANYCH MIKROPROCESORA do szyny adresowej i danych PAMIĘCI DYNAMICZNEJ, zdjęcie sygnału PRACA z MIKROPROCESORA do czasu odszukania przez KOMPARATOR (7) komórki pamięci o zadanym adresie. Z chwilą odszukania danej komórki pamięci, sygnał A=D z KOMPARATORA (7) powoduje zmianę częstotliwości impulsów ZEGARA TAKTUJĄCEGO PAM. DYNAMICZNA (9).

Wyjście z programem poza zastrzeżony obszar adresów, PAMIĘCI OPERACYJNEJ (5), dla komunikacji z PAMIĘCIĄ DYNAMICZNĄ (6) powoduje powrót do pierwotnej częstotliwości impulsów taktujących z ZEGARA TAKTUJĄCEGO (9).

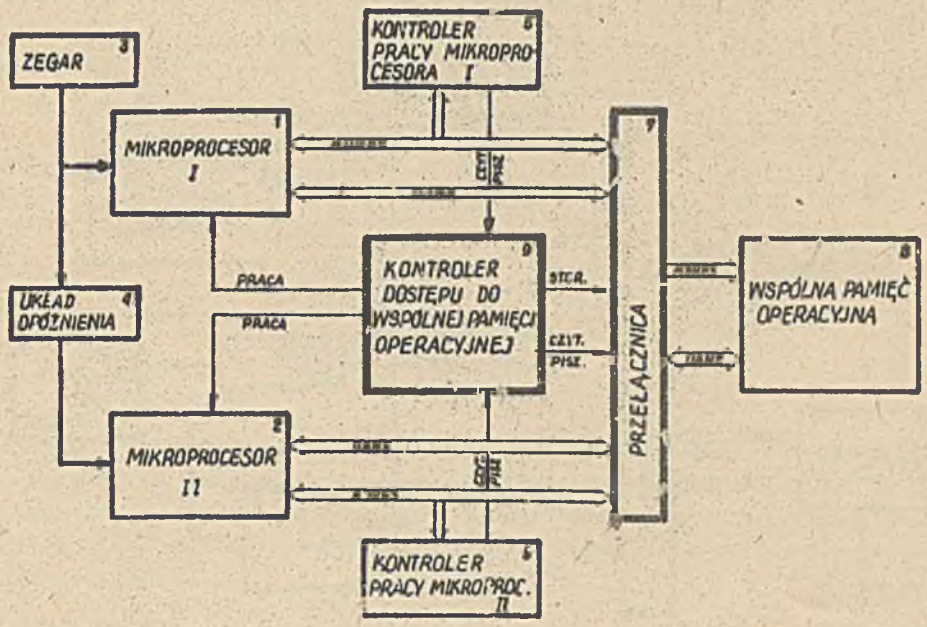
W komputerze dla zwiększenia elastyczności pracy z pamięcią dynamiczną wskazano jest przewidzieć oba przedstawione powyżej sposoby przesyłania.

Organizacja czytania informacji i sposób adresowania urządzeń zewnętrznych *)

Przyjęty sposób organizacji czytania informacji pozwala w jednym cyklu rozkazu CZYTAJ na bezpośrednie pobranie informacji od urządzeń oddalonych do 700 m od komputera. Komputer wysyła do

---

*) Dokładny opis organizacji czytania przedstawiony jest w artykule: Wrona J. "Organizacja bezpośredniego wprowadzania informacji do urządzeń systemów sterowania." BI OSK 1979 nr 2



Rys. 5.*

W wypadku gdy mikroprocesor zgłasza się do pamięci a jest ona zajęta przez drugi mikroprocesor, KONTROLER DOSTĘPU DO PAMIĘCI (9) zdejmuje z mikroprocesora sygnał "PRACA", co powoduje wstrzymanie jego pracy do końca przesłania słowa, w trakcie którego nastąpiło zgłoszenie. Warunkiem poprawnego działania układu jest, aby oba mikroprocesory traktowane były impulsami ze wspólnego ZEGARA (3). Dla uniknięcia występowania hazardów czasowych i związanych z tym przekłamań, impulsy zegarowe podawane są do MIKROPROCESORA II przez UKŁAD OPÓŹNIENIA (4) dający stałe opóźnienie czasowe w stosunku do impulsów zegarowych podawanych na MIKROPROCESOR I.

Organizacja dynamicznej pamięci programów wykonawczych

Jak już wcześniej było podane, pamięć dynamiczna programów wykonawczych zbudowana jest na rejestrach dynamicznych, charakteryzujących się dużą pojemnością i niską ceną.

Pojedynczy układ scalony składa się z pewnej liczby rejestrów o pojemności 256 bitów (produkuje się obecnie rejestry o pojemności 256 rejestrów 256 bitowych). Organizacja adresowania pozwala na wybieranie tylko rejestrów.

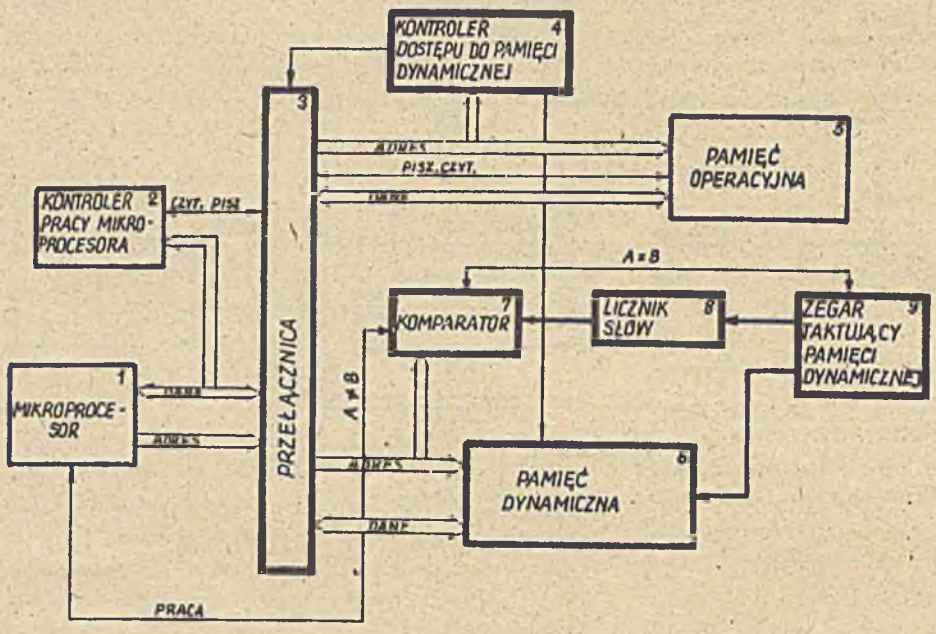
Można stosować zasadniczo dwa rodzaje pracy pamięci, tj.

- wykorzystując standardowy kontroler do współpracy z pamięciami dyskowymi, dzięki takiej samej organizacji rejestrów dynamicznych jak pamięci dyskowych; pozwala on przepisywać informację między pamięciami - operacyjną i dynamiczną - blokami po 256 słów z pominięciem procesora; przy większej liczbie automatów, do których wysyłane są różne programy, wymaga to znacznego zwiększenia pamięci operacyjnej a więc i kosztów komputera;
- wykorzystując blokowe przesyłanie programów wykonawczych do automatów, z których każdy jest zakończony słowem końca bloku. Ponieważ długość bloku nie przekracza kilkudziesięciu słów, pozwala to na znaczne zmniejszenie pamięci operacyjnej.

Przy takiej pracy komórki pamięci dynamicznej są adresowane tak samo, jak komórki pamięci operacyjnej; do czasu znalezienia odpowiedniej w rejestrze zostaje wstrzymana praca mikroprocesora, Jo-

* Linia pogrubiona oznacza układy niestandardowe

wybranego urządzenia sygnał CZYTAJ, odpowiedzią na sygnał jest przesłana do niego informacja. Jest ona wpisywana do specjalnego rejestru, wspólnego dla wszystkich (bądź grupy) urządzeń.



Rys. 4.

Sposób adresowania polega na tym, że poszczególnym adresom komórek pamięci przyporządkowane są adresy urządzeń zewnętrznych i przesłanie bądź pobranie informacji odbywa się wtedy, gdy z danej komórki pamięci wysyłany jest rozkaz WE/WY.

Oprócz opisanego sposobu adresowania, umożliwiającego dołączanie praktycznie nieograniczonej liczby urządzeń zewnętrznych, systemy mikroprocesorowe mają standardowe układy WE/WY, których pewną liczbę należy przewidzieć, np. do podłączania urządzeń peryferyjnych.

Sposób przesyłania informacji i interfejs

Dążenie do zmniejszania do minimum liczby połączeń, a także możliwość prostego dołączania nowych urządzeń do systemu, narzuca warunek sposobu przesyłania informacji po szynach, do których podłączone są równolegle urządzenia. Zasadniczo przyjmuje się dwie szyny:

- szynę A do przesyłania programów wykonawczych dla automatów
- szynę B do przesyłania do urządzeń rozkazów sterujących i czytania od nich informacji.

Schemat dołączania urządzeń zewnętrznych pokazany jest na rys. 5. W wypadku dużych systemów sterowania należy przewidzieć możliwość stosowania po kilka szyn A i B.

Przedstawiony sposób przesyłania informacji nie dotyczy komunikacji ze standardowymi urządzeniami peryferyjnymi, która powinna odbywać się za pomocą standardowych układów WE/WY.

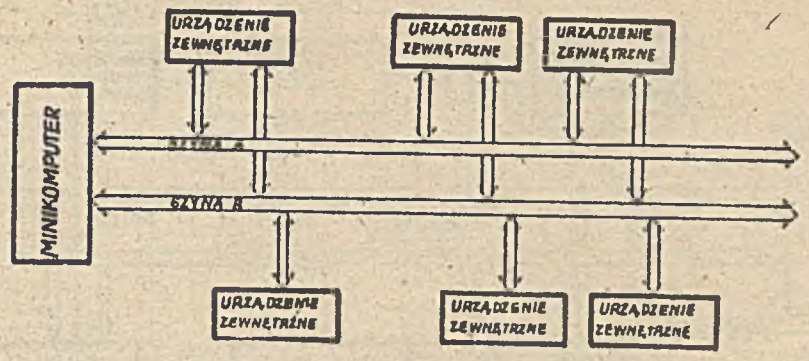
Przesyłanie informacji dwoma torami wynika z dwumikroprocesorowej organizacji wewnętrznej komputera.

Przyjęta organizacja wewnętrzna komputera oraz sposób przesyłania informacji pozwalają na zmniejszenie do minimum sygnałów indywidualnych przesyłanych między komputerem a urządzeniem, a mianowicie do sygnałów

"POWRÓT GOTOWOŚCI"

"TESTUJ"

Pierwszy informuje komputer o ponownej gotowości urządzenia na przyjęcie rozkazu. Sygnał "TESTUJ"



Rys. 5.

został wprowadzony ze względów bezpieczeństwa. Warunkiem poprawnej pracy systemów sterowania jest możliwość programowego testowania transmisji i urządzeń. Praktyka wykazała, że jeżeli brak indywidualnych sygnałów mówiących o testowaniu, istnieje możliwość interprotacji przez urządzenie rozkazu testującego jako rozkazu wykonawczego, co może prowadzić do znacznych awarii i niebezpieczeństwa dla osób pracujących.

Organizacja układów wejścia/wyjściu

Przy wysyłaniu rozkazów do urządzeń mogą wystąpić trzy warianty rozkazu PISZ.

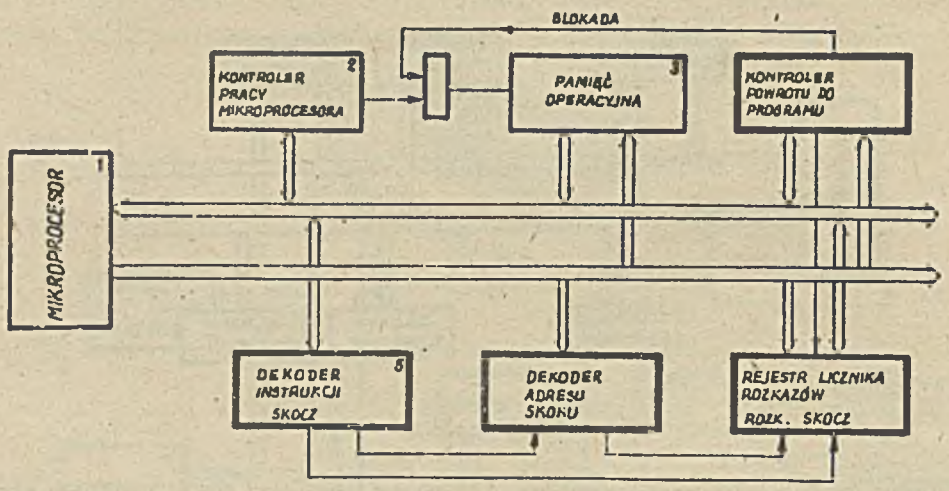
- Komputer wysyła rozkaz sterujący, zaś następny będzie wysłany po odpowiednio długim czasie, dłuższym niż wykonanie rozkazu, nie ma więc potrzeby informowania procesora o ponownej gotowości urządzenia.
- Komputer wysyła rozkaz sterujący i zaraz po jego wykonaniu należy wysłać nowy rozkaz. W tym wypadku po pojawieniu się sygnału POWRÓT GOTOWOŚCI należy zgłosić do procesora przerwanie programowe.
- Komputer wysyła rozkaz testujący, który powinien być wysłany z indywidualnym sygnałem TESTUJ.

Pomijając stosowanie nielegalnych rozkazów jako nie do przyjęcia, pozostają dwa sposoby: przez zapalanie specjalnych wskaźników bądź przez adres komórki pamięci operacyjnej, z której jest pobierany rozkaz PISZ. Pierwszy sposób jest o tyle kłopotliwy, że zespół wskaźników w obecnej strukturze systemów mikroprocesorowych można traktować wyłącznie jako układ WE/WY i przez rozkazy WE/WY gasić je lub zapalać.

Ponieważ już wcześniej przyjęliśmy, że adresowanie urządzeń zewnętrznych odbywa się przez numery komórek pamięci operacyjnej, wystarczy przyjąć, że rozkaz PISZ do każdego urządzenia jest napisany w trzech różnych komórkach pamięci i w zależności od ich numeru ma inne znaczenie. W praktyce, przy założeniu, że rozkazy WE/WY są zapisane w pamięci stałej, sprowadza się do wydzielenia czterech obszarów: trzech dla rozkazów PISZ i jednego dla rozkazów CZYTAJ.

Przyjęcie umieszczenia rozkazów WE/WY w pamięci stałej wymaga (dla ułatwienia pracy programistów i przyspieszenia działania programów) takiej organizacji sprzętowej, aby nastąpił powrót do dalszego ciągu programu. Przyjmuję, że po rozkazie WE/WY nastąpi skok do komórki pamięci o numerze większym o tyle, ile słów zajmował rozkaz skoku do rozkazu WE/WY. Schemat układu powodującego powrót do dalszego ciągu programu pokazany jest na rys. 6.

Praca układu jest następująca: przy każdym pobieranym z pamięci rozkazie SKOCZ (rozpoznanym przez DEKODER INSTRUKCJI SKOCZ (5)) i po stwierdzeniu przez DEKODER ADRESU SKOKU (6) adres rozkazu SKOCZ zostaje zapamiętany w REJESTRZE LICZNIKA ROZKAZÓW (7) i powiększony o tyle, ile słów zajmował rozkaz SKOCZ. KONTROLER POWROTU DO PROGRAMU (4) powoduje, że po rozkazie WE/WY bez względu na licznik rozkazów zawsze będzie wykonywany rozkaz SKOCZ do adresu zapisanego w REJESTRZE



Rys. 6.

LICZNIKA ROZKAZÓW (7). Powoduje on, że na czas pobrania rozkazu SKOCZ z KONTROLERA POWROTU DO PROGRAMU sygnałem "BLOKADA", pamięć operacyjna jest zablokowana.

System przerwań programowych

System przerwań programowy oparty jest na standardowych układach przyjmowania przerwań, przyjętych dla stosowanego typu mikroprocesorów. Nietypowa organizacja wewnętrzna komputera wymaga specjalnych wewnętrznych przerwań programowych, a mianowicie:

- przerwania zgłaszane od jednego do drugiego mikroprocesora,
- przerwania zgłaszane do mikroprocesorów przez UKŁADY KONTROLI CZASU WYKONANIA OPERACJI, w wypadku wykrycia przekroczenia przez dowolne urządzenie dopuszczalnego czasu wykonywania operacji,
- przerwania zgłaszane do mikroprocesorów przez układy POWRÓT GOTOWOŚCI,
- przerwania zgłaszane w wypadku nieprawidłowej realizacji instrukcji "CZYTAJ" przy pracy na odległość.

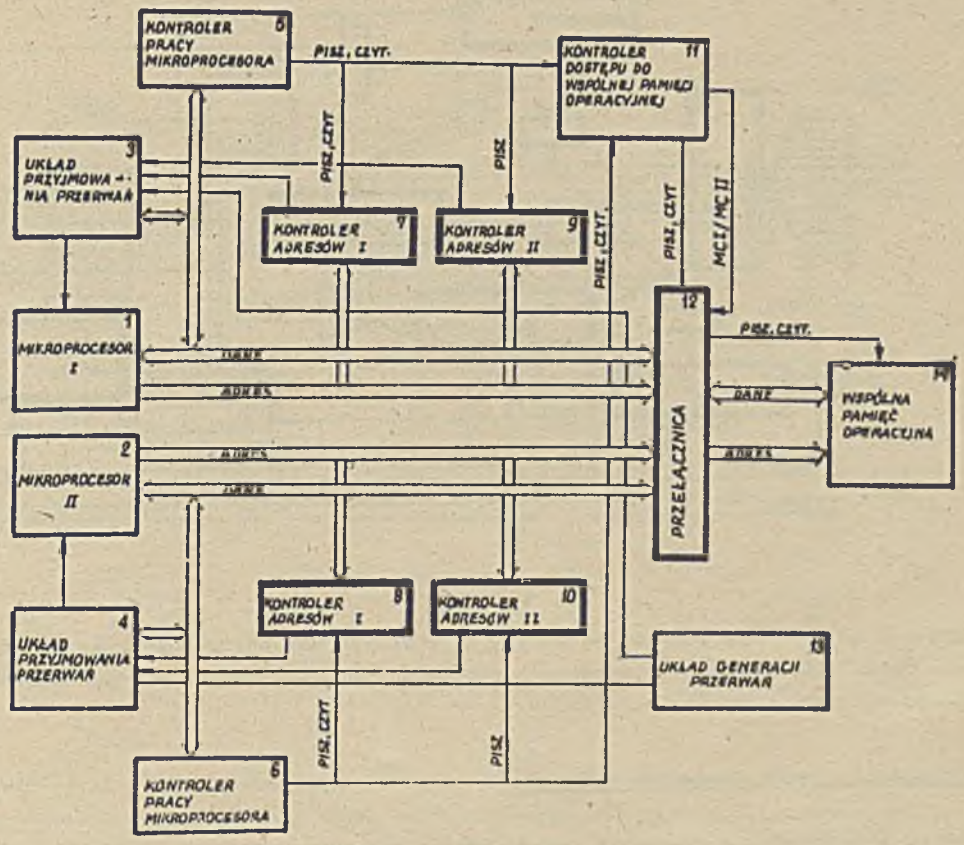
Dochodzą do nich typowe przerwania generowane w wypadku próby przesyłania informacji do którejś z komórek pamięci stałej bądź do nieistniejącej.

Do zgłaszania przerwań pomiędzy mikroprocesorami wykorzystana jest ich praca ze wspólną pamięcią operacyjną, a mianowicie, przesłanie informacji do określonego obszaru powoduje wygenerowanie przerwania do drugiego mikroprocesora. Ten sposób wymaga zabezpieczenia przed niepożądanymi przerwami biorącymi się z normalnej pracy mikroprocesorów z pamięcią operacyjną. W komputerze przyjęto, że generację przerwania powodować będzie kolejne przesłanie informacji do czterech kolejnych wydzielonych dla danego mikroprocesora komórek pamięci operacyjnej. Z przesłanych słów dwa będą stanowiły adres początku programu, do którego ma przejść mikroprocesor z wyniku przerwania, zaś pozostałe dwa będą negacją dwóch pierwszych słów i będą stanowiły zabezpieczenie przed fałszywymi zgłoszeniami przerwań.

Schemat układu generującego przerwania pokazany jest na rys. 7.

Przedstawiony układ powoduje wygenerowanie przerwań:

- przy próbie wpisania informacji do obszaru pamięci stałej za pomocą KONTROLERA ADRESÓW II (9, 10)
- przy próbie przesłania informacji do komórki o nieistniejącym adresie za pomocą KONTROLERA ADRESU I (7,8)



Rys. 7.

UKŁAD GENEROWANIA PRZERWAŃ (13) powoduje wygenerowanie przerwań zgłaszanych od jednego mikroprocesora do drugiego.

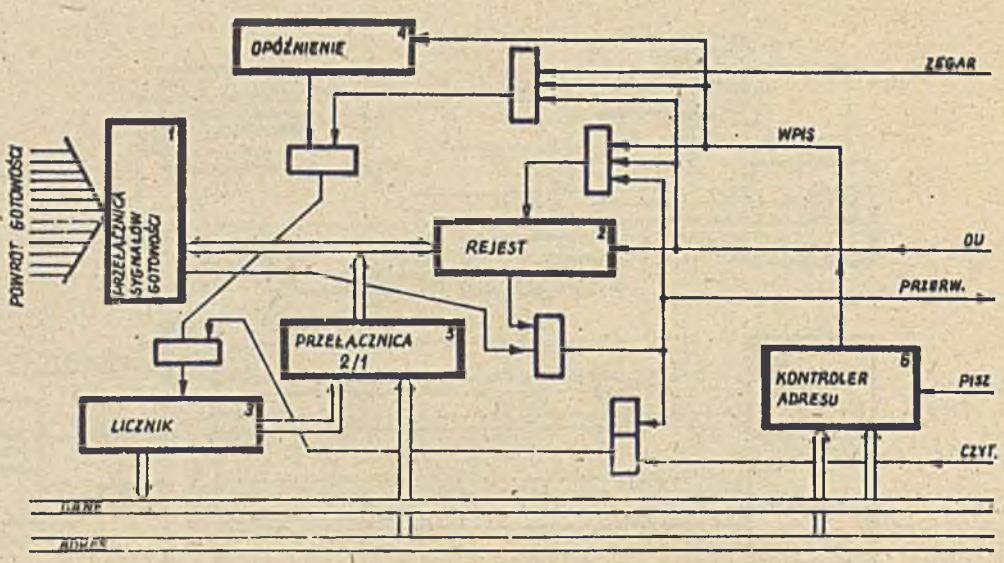
Układ sygnalizacji ponownej gotowości

Wysyłanie rozkazów sterujących do urządzeń może mieć trzy warianty. Jednym z nich jest żądanie zgłoszenia przerwania programowego po ponownym pojawieniu się sygnału POWRÓT GOTOWOŚCI od wybranego urządzenia. Schemat układu pokazany jest na rys. 8.

D z i a ł a n i e u k ł a d u

Jednobitowy REJESTR (2) służy do zapamiętywania konieczności wysłania przerwania przy pojawieniu się sygnału POWRÓT GOTOWOŚCI. Adres komórki w REJESTRZE odpowiada adresowi urządzenia. Jeżeli komputer wysyła rozkaz do urządzenia a KONTROLER ADRESU (6) stwierdzi, że pochodzi on z odpowiedniego obszaru pamięci operacyjnej (przeznaczonego do pracy z sygnalizacją powrotu gotowości), przesyła sygnał "WPIS", powodujący wpisanie do komórki pod adresem urządzenia sygnału "OU" z KONTROLERA PRACY MIKROPROCESORA, będącego jedynką.

Sygnały "POWRÓT GOTOWOŚCI" podawane są na PRZEŁĄCZNICĘ SYGNAŁU GOTOWOŚCI (1); wybieraniem poszczególnych sygnałów steruje LICZNIK (3), na który podawane są impulsy zegarowe. LICZNIK steruje także wybieraniem komórek w REJESTRZE. Jeżeli z wybranej komórki będzie odczytana jedynka a z PRZEŁĄCZNICY SYGNAŁU GOTOWOŚCI (1) przyjdzie sygnał POWRÓT GOTOWOŚCI, to zostanie wygenerowane przerwanie programowe. Sygnał przerwania powoduje odcięcie sygnałów zegarowych od LICZNIKA (3) oraz wpisanie zera do wybranej komórki REJESTRU (2). Przez mikroprocesor LICZNIK traktowany jest jako komórki pamięci stałej, z niego czytany jest numer urządzenia zgłaszającego przerwaniem gotowości. Do czasu przeczytania stanu LICZNIKA (3) podawanie na niego impulsów zegarowych jest wstrzymane.



Rys. 8.

Ponieważ od wysłania rozkazu do zaniku sygnału "POWRÓT GOTOWOŚCI" musi minąć jakiś czas wprowadzono układ OPÓŹNIENIE (4) powodujący podawanie impulsów zegarowych na LICZNIK po określonym czasie od wpisania jedynki do REJESTRU.

Układ kontroli czasu wykonania operacji

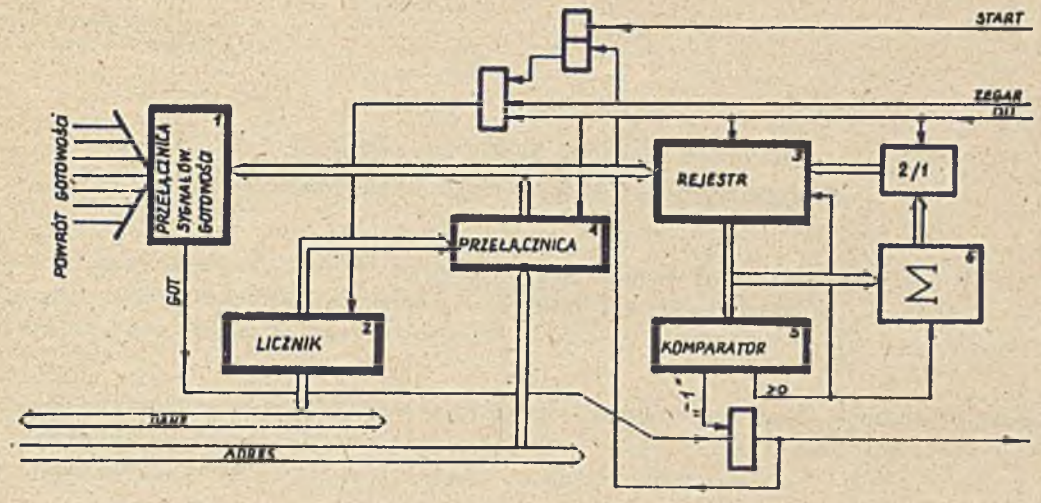
Jednym z najbardziej istotnych zagadnień w systemach sterowania jest odpowiednio wczesne wykrywanie przez system uszkodzeń czy też zacięć mechanizmów. Podawanie z każdym rozkazem czasu jego wykonania powoduje rozbudowę programów wykonawczych a przede wszystkim poważnie utrudnia przygotowywanie programów.

Doświadczenia z budowy systemów wskazują, że wystarczy przyjąć jako kontrolny, czas najdłużej wykonywanej instrukcji przez dane urządzenie, zaś dla uproszczenia budowy układów kontrolnych wystarczy przeprowadzać kontrolę w kilku przedziałach czasowych dla poszczególnych grup urządzeń. W przedstawionym na rys. 9 układzie kontrola czasu odbywa się poprzez kontrolę czasu upływającego od wysłania rozkazu do ponownego pojawienia się sygnału "POWRÓT GOTOWOŚCI". Każde wysłanie rozkazu do urządzenia powoduje odliczanie czasu od początku. Przekroczenie czasu sygnalizowane jest przerwaniem programowym do mikroprocesora.

Praca układu jest następująca: do zliczania impulsów czasowych służy REJESTR (3), będący układem pamięci RAM o takiej liczbie komórek, ile jest urządzeń, przy czym adres urządzenia jest zarazem adresem komórki; sygnał "OU" z KONTROLERA PRACY MIKROPROCESORA powoduje wpisanie jedynki do najmniej znaczącego bitu do komórki o adresie urządzenia.

Przeglądanie sygnałów "POWRÓT GOTOWOŚCI" odbywa się za pomocą PRZEŁĄCZNICY (1), adres kolejnych wejść jest wybierany przez LICZNIK (2) napędzany przez impulsy "ZEGAR" z generatora. Po wybraniu nowego adresu przez LICZNIK, KOMPARATOR (5) bada zawartość wybranej komórki w REJESTRZE. Jeżeli wpisane są do niej same jedynki i jednocześnie brak jest sygnału "POWRÓT GOTOWOŚCI" wysłano jest przerwanie programowe. Numer urządzenia, które było przyczyną wysłania przerwania jest odczytywany z LICZNIKA (2), który traktowany jest jako komórka pamięci operacyjnej stałej. Do czasu przeczytania stanu LICZNIKA zostaje wstrzymana jego praca. Ponowne przeszukiwanie następuje po sygnale "START".

Jeżeli w wybranej komórce REJESTRU nie występują same jedynki bądź zera w układzie następuje dodanie jedynki do liczby pobranej i ponownie wpisanie jej do tej samej komórki REJESTRU.



Rys. 9.

Zakończenie

Celem artykułu jest próba pokazania możliwości budowy specjalizowanego minikomputera opartego na typowym systemie mikroprocesorowym INTEL 8080 do sterowania procesami wytwórczymi.

Jeżeli uwzględnimy, że pewne układy powtarzają się i przy odpowiedniej organizacji sygnałów taktujących mogą być wspólne, realizacja specjalnych funkcji komputera związanych ze sterowaniem nie wymaga dużej liczby elementów elektronicznych. Dotyczy to układów przełącznicy powrotu gotowości, kontrolerów adresów pamięci operacyjnej, przełącznic szyn itp.

Realizacja specjalnych funkcji sterowania w wewnętrznej strukturze minikomputera jest znacznie łatwiejsza i wymaga mniej układów elektronicznych niż dobudowanie ich do standardowego komputera. Ocenia się, że koszt specjalizowanego minikomputera będzie zbliżony do kosztu dodatkowych układów niezbędnych do wyposażenia standardowego komputera w celu przystosowania go do sterowania procesami wytwarzania w czasie rzeczywistym.

dr inż. Tomasz JANNSON
doc.dr hab.inż. Zygmunt ZAWISŁAWSKI
Instytut Fizyki
Politechniki Warszawskiej

Niektóre zagadnienia prędkości wydruku optoelektronicznych drukarek kserograficznych

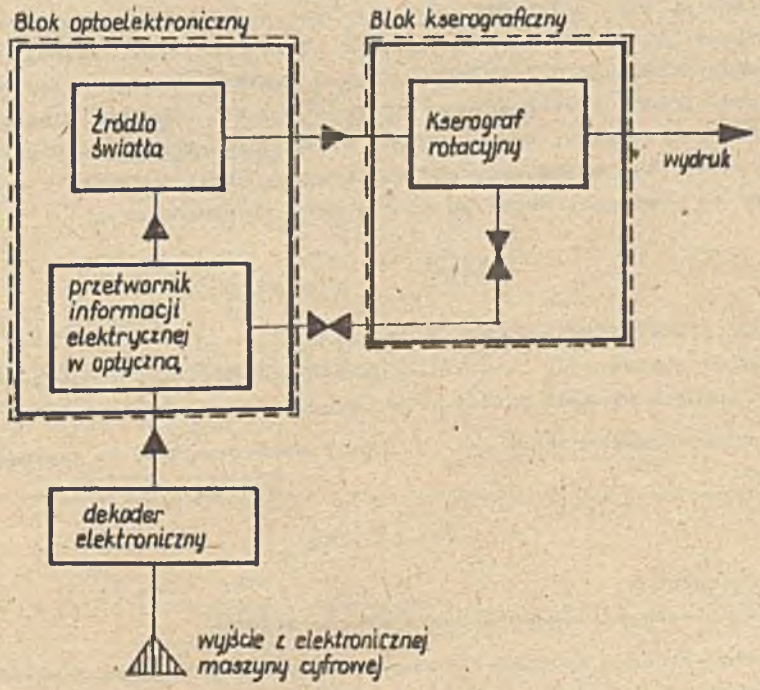
W ostatnich latach pojawiły się na rynku drukarki tzw. bezuderzeniowe, charakteryzujące się, w porównaniu z drukarkami uderzeniowymi, niezwykłą prędkością wydruku, dochodzącą do 21 tys.wierszy na minutę, tzn. około 10 razy większą od prędkości uzyskiwanych przez szybkie drukarki uderzeniowe [1].

Tak dużą szybkość wydruku osiągnięto przez zastosowanie techniki laserowej i procesu elektrofotograficznego (kserografii). Drukarka taka, obok wielu innych zalet, charakteryzuje się szczególnie cichą pracą, przy możliwości wydruku na zwykłym papierze.

Schemat blokowy optoelektronicznej drukarki kserograficznej przedstawiony jest na rys. 1. Składa się ona z dwóch podstawowych bloków:

- bloku optoelektronicznego, w którym znajduje się przetwornik optoelektroniczny i źródło światła
- bloku kserograficznego, w którym znajduje się warstwa fotoprzewodząca naniesiona na metaliczny bęben

i z urządzeń pomocniczych: urządzenie ładowania warstwy, wywoływaniu utajonego obrazu na warstwie kserograficznej powstałego po naświetleniu, przenoszenia obrazu proszkowego z warstwy kserograficznej na papier, czyszczenia warstwy kserograficznej z resztek obrazu proszkowego, utrwalania przeniesionego obrazu proszkowego na papier.



Rys. 1.

Praca poszczególnych urządzeń oparta może być na różnych sposobach i różnych zasadach działania [2], [3], [4], [5], [6].

Zadaniem bloku optoelektronicznego jest przetworzenie sygnałów elektrycznych z elektronicznej maszyny cyfrowej na sygnały optyczne, natomiast zadaniem bloku kserograficznego, który jest zmodyfikowanym kserografem rotacyjnym - jest utrwalanie sygnałów optycznych na papierze.

Duże prędkości drukowania nakładają niezwykle ostre wymagania dotyczące parametrów materiałów kserograficznych, w szczególności czasu regeneracji warstwy fotopółprzewodnika.

Prędkość wydruku zależy przede wszystkim od następujących parametrów: czułość i czas regeneracji materiału światłoczułego, sposób obróbki kserograficznej, moc źródła światła, przepustowość kanału optycznego, standard wydruku, średnica bębna kserograficznego. Parametry te są od siebie zależne i dlatego można je - w pewnych granicach - optymalizować. Natomiast wielkość czasu regeneracji wpływa bezpośrednio na prędkość wydruku i dlatego opracowanie technologii materiału światłoczułego o czasie regeneracji praktycznie ok. paru sekund, ma istotne znaczenie.

Z układowego punktu widzenia, proces kserograficzny jest równoważny z układem optycznym odwzorowującym obraz optyczny zarejestrowany na warstwie światłoczułej w obraz wydrukowany na papierze. Istotną rzeczą wobec tego jest określenie parametrów charakteryzujących układ jako całość. Poza podstawową zależnością między ekspozycją warstwy światłoczułej a gęstością optyczną otrzymanego wydruku, istotne znaczenie mają trzy parametry układowe: globalna czułość procesu, całkowity czas procesu i funkcja przenoszenia układu.

• Do określenia czułości kserograficznej jak dotąd brak jednoznacznych standardów. Najczęściej jako czułość przyjmuje się odwrotność naświetlenia (ekspozycji) wywołującego spadek potencjału płyty kserograficznej do połowy jej wartości [7]:

$$S = \left(\frac{1}{H}\right)_{\Delta V = \frac{V_0}{2}} \tag{1}$$

gdzie V_0 - potencjał początkowy płyty kserograficznej
H - ekspozycja płyty

Pomiar czułości przy stałym naświetlaniu sprowadza się w tym wypadku do pomiaru czasu spadku potencjału płyty kserograficznej do połowy. Przy tak przyjętej mierze czułości, pojawiają się trudności wynikające stąd, że czas połowicznego zaniku potencjału płyty kserograficznej zależy zarówno od wielkości ekspozycji, jak i od potencjału początkowego płyty. Należy jednak zaznaczyć, że czułość procesu kserograficznego zależy nie tylko od czułości samego materiału światłoczułego, lecz także od sposobu obróbki kserograficznej, aż do momentu wydruku. Dlatego też, przez analogię do teorii procesu fotograficznego, wygodnie jest wprowadzić globalną czułość procesu kserograficznego, traktowanego tutaj jako układ optyczny odwzorujący obraz optyczny z warstwy światłoczułej na obraz wydrukowany na papierze. Można ją zdefiniować za pomocą wzoru:

$$C \stackrel{def}{=} \frac{1}{H_c} \tag{2}$$

gdzie: C - czułość globalna procesu kserograficznego
H_c - minimalna ekspozycja, dla której jakość wydruku jest zadowalająca z punktu widzenia specyficznych wymagań określonego układu optycznego

• Całkowity czas procesu kserograficznego T_c przy obrotowym bębnie kserograficznym możemy zdefiniować następująco:

$$T_c = \max\left[\tau, T\right] \tag{3}$$

gdzie: τ - czas regeneracji
T - czas najdłuższego procesu obróbki kserograficznej

a więc czas całkowitego procesu kserograficznego będzie równy większej spośród dwóch wartości czasów podanych w nawiasie wzoru (3).

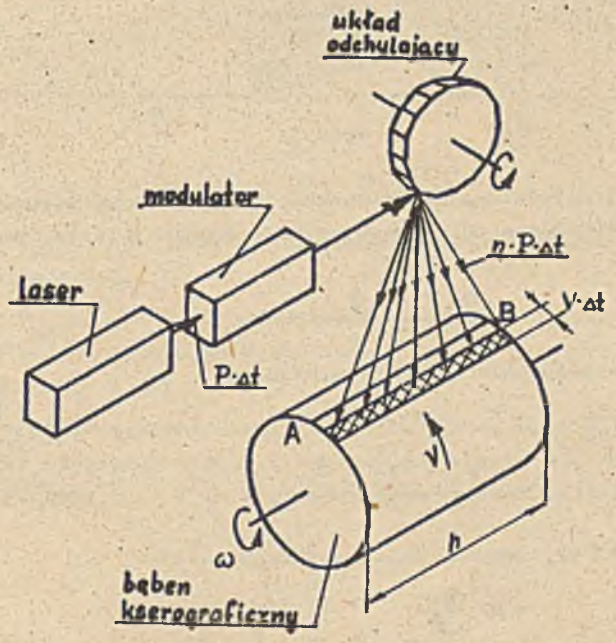
Czas regeneracji zależy przede wszystkim od czasu relaksacji, który jest bezpośrednio związany z czasem życia pułapek w materiale fotoprzewodzącym. Dodatkowymi czynnikami wpływającymi na czas regeneracji jest technologia otrzymywania warstwy, warunki ładowania i oświetlenia.

Funkcja przenoszenia układu charakteryzuje jakość odwzorowywania obiektów, które w zasadniczy sposób wpływają na jakość wydruku. Funkcja ta zależy przede wszystkim od sposobu wywoływania obrazu proszkowego na warstwie światłoczułej, rozmiarów ziaren proszku wywołującego i metody przenoszenia obrazu proszkowego na papier. Np. firma CANON preferuje wywoływanie cieczowe, dające bardzo dużą zdolność rozdzielczą obrazu.

Zakładając, że bęben ksorograficzny z naniesioną warstwą światłoczułą ma kształt walca kołowego o średnicy d i obraca się ze stałą prędkością ω, to prędkość liniowa punktów na powierzchni walca wynosi $v = \frac{\omega d}{2}$. Czas regeneracji warstwy światłoczułej musi być nie większy od okresu obiegu bębna, co prowadzi do podstawowej relacji wiążącej prędkość wydruku z czasem regeneracji

$$\tau \leq \frac{\pi d}{v} \qquad (4)$$

Jeśli czas naświetlania wynosi Δt, to na bęben ksorograficzny pada energia (rys. 2):



Rys. 2.

$$E = \eta_o \cdot P \cdot \Delta t$$

gdzie: P - moc źródła światła

η_o - przepustowość układu optycznego (sprawność)

na obszar o powierzchni

$$S = v \cdot h \cdot \Delta t$$

która w czasie Δt minie odcinek AB. W związku z tym ekspozycja H na powierzchni walca wyniesie

$$H' = \frac{E}{S}, = \frac{\eta_o \cdot P \cdot \Delta t}{v \cdot h \cdot \Delta t} = \frac{\eta_o \cdot P}{v \cdot h} \qquad (5)$$

Jeśli $[P] = W$; $[h] = cm$, $[v] = \frac{cm}{S}$; $\eta_o = [1]$

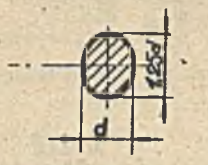
to $[H] = \frac{J}{cm^2}$

Ze względu na dużą szybkość pracy urządzenia przyjęto, że walec ksorograficzny obraca się ruchem

jednostajnym. Powoduje to, że kropki, z których składają się litery, trzeba zapisywać impulsowo. Przyjmując taki czas naświetlania wiersza kropek T_K, aby był on pełnym ułamkiem czasu T_o potrzebnego do obrócenia się walca o odległość między wierszami kropek, można ograniczyć rozmycie kropek do dopuszczalnych granic. Jeśli założy się np., że:

$$\frac{T_k}{T_o} = \frac{1}{5}$$

to wymiary kropek o średnicy d będą miały wymiary jak na rys. 3.



Rys. 3.

W związku z tym relację (5) należałoby zmodyfikować, uwzględniając współczynnik wypełnienia impulsu świetlnego, który zależeć będzie od rodzaju użytego źródła światła. Ostatecznie wzór (5) przyjmie postać:

$$H = \frac{\eta_o \cdot P \cdot \eta_w}{v \cdot h} \qquad (6)$$

gdzie: η_w - współczynnik wypełnienia impulsu świetlnego

Uwzględniając wzór (2) dla $H = H_o$ otrzymamy

$$v = \frac{\eta_o P \cdot \eta_w \cdot C}{h} \qquad (7)$$

Biorąc pod uwagę warunek (4), jednocześnie musi zachodzić

$$v \leq \frac{\pi d}{\tau} \qquad (8)$$

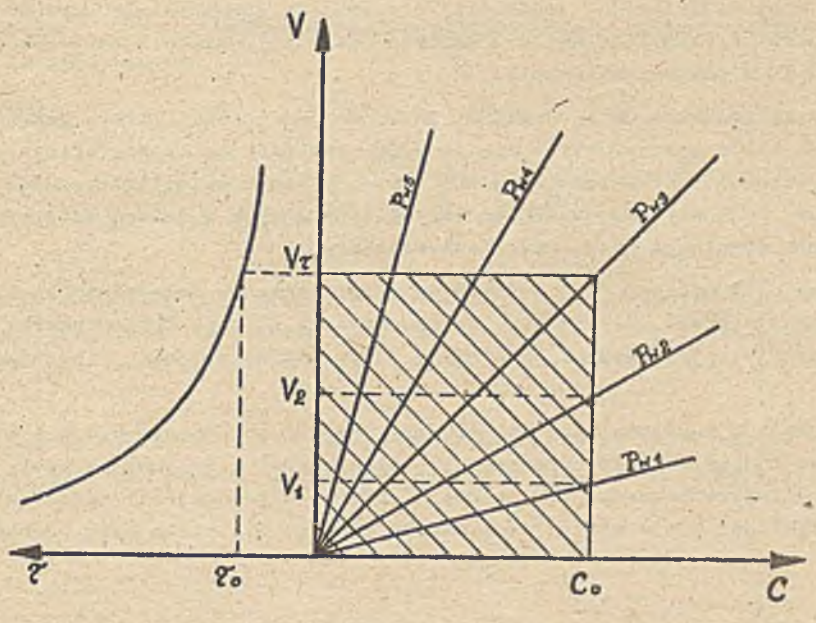
Ostatecznie otrzymamy

$$v_{max} = \min \left[\frac{\eta_o \cdot P \eta_w \cdot C}{h} ; \frac{\pi d}{\tau} \right] \qquad (9)$$

a więc maksymalna prędkość wydruku v_{max} kserograficznej drukarki optoelektronicznej jest równa mniejszej spośród dwóch wartości podanych w nawiasie wyrażenia (9). Ponieważ parametry h, i d są ustalone, więc relacja (9) określa zależność prędkości wydruku od następujących parametrów bloku optoelektronicznego: mocy źródła, przepustowości kanału optycznego, rodzaju użytego źródła światła, a także od parametrów bloku kserograficznego, tj. czułości i czasu regeneracji materiału światłoczułego (pośrednio także od parametrów obróbki kserograficznej).

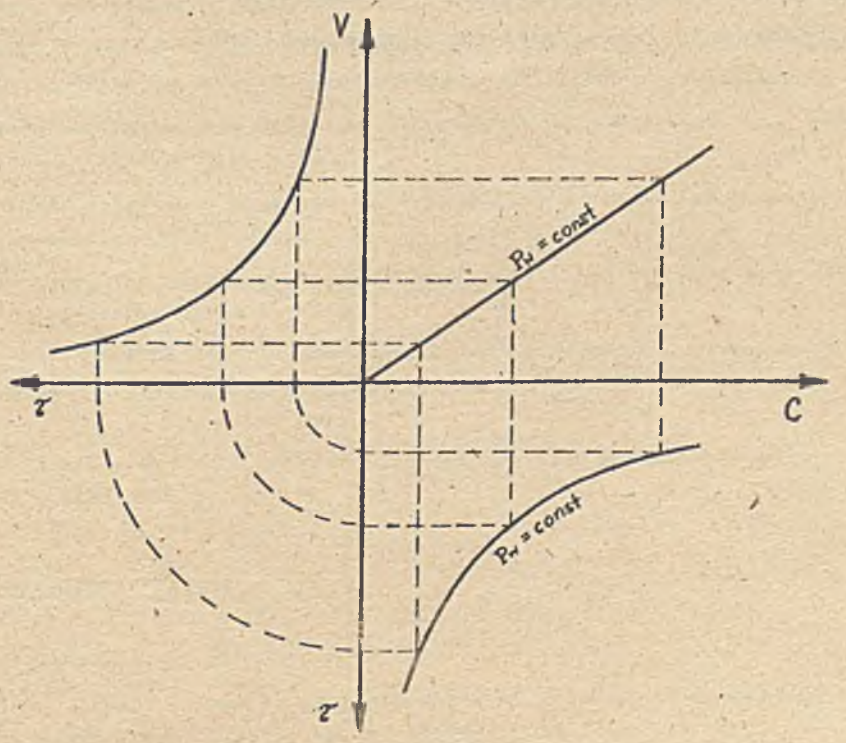
Na rys. 4 zilustrowana jest relacja (9); w pierwszej ćwiartce przedstawiona jest relacja (7), a w drugiej relacja (8). Dla czasu regeneracji $\tau = \tau_o$ otrzymujemy maksymalną prędkość wydruku równą v_τ. Pozioma prosta o równaniu $v = v_\tau$ ogranicza od góry dopuszczalny zakres prędkości. Jednocześnie prosta pionowa $c = Co$ daje nam ograniczenie ze względu na czułość procesu kserograficznego. W ten sposób otrzymujemy dopuszczalny obszar prędkości obszar zakreskowany dla różnych mocy wejściowych $P_w = \eta_o P$ bezpośrednio padającej na warstwę światłoczułą bębna kserograficznego.

Na podstawie wykresu (rys. 4) można wyciągnąć następujące wnioski:



Rys. 4.

- dla $P_w \geq P_{w3}$ mamy $v_{max} = v_\tau$ i na ograniczenie prędkości wydruku wpływa wartość czasu regeneracji warstwy światłoczułej,
- dla $P_w < P_{w3}$ mamy $v_{max} < v_\tau$ i na ograniczenie prędkości wpływa czułość procesu kserograficznego, np. dla mocy wejściowych zawartych między P_{w1} i P_{w2} prędkość maksymalna jest zawarta między prędkościami v_1 i v_2,
- interesujący jest przypadek optymalnego wykorzystania własności materiału (i całego procesu kserograficznego), gdy czas regeneracji i czułość procesu są tak dobrane, że jednocześnie ograniczają v_{max} tzn. gdy $P_w = P_{w3}$. Sytuacja taka jest zilustrowana na rys. 5.



Rys. 5.

Dla określonej mocy wejściowej i prędkości, otrzymujemy jednocześnie wartość na czas regeneracji τ i czułość kserograficzną c (czwarta ćwiartka). Zmieniając prędkość otrzymujemy zależność $c = c(\tau)$ w postaci hiperboli.

Obecnie na świecie produkowane są drukarki laserowe przez takie firmy, jak Cannon, IBM; należy przypuszczać, że tendencja wykorzystania laserów oraz procesu kserograficznego będzie nadal utrzymywana. W przyszłości bez względu na wybór wariantu bloku optoelektrycznego, niezbędny będzie blok kserograficzny, nadający się do zapisu danych wyjściowych z maszyn matematycznych, przy zastosowaniu w zasadzie dowolnego bloku optoelektronicznego.

Opierając się na dotychczasowych krajowych doświadczeniach w dziedzinie elektrofotografii(kserografii) i techniki laserowej wydaje się, że opracowanie komputerowej drukarki laserowo-elektrofotograficznej, zgodnie z tendencjami światowymi w tym zakresie, jest możliwe w naszych warunkach.

Pewne wstępne prace w zakresie elektrofotografii są już wykonane w kraju i stanowi to już dobry punkt wyjścia do budowy szybkiego roracyjnego elektrografu. Podobnie zresztą postąpili producenci drukarek optoelektronicznych (IBM, Cannon), którzy wyprodukowali optoelektroniczne drukarki laserowe wykorzystując swoje własne specyficzne rozwiązanie bloku elektrograficznego.

Literatura

[1] KUCHENBECKER H., UNGER H.: Data Raport (firmy Siemens) 1976 nr 6

[2] Elektrofotografia. WNT: Warszawa 1965

[3] SAVETA N.N.: Bystrodejstvujuščie pečatajuščie ustrojstva. Mašinostroenie: Moskva 1965

[4] SCHAFFERT R.M.: Elektrophotography. The Focal Press: London 1965

[5] ZAWISŁAWSKI Z.: Badanie wybranych zagadnień procesu elektrofotograficznego. Politechnika Warszawska: Warszawa 1971 Elektryk nr 22

[6] ZAWISŁAWSKI Z., DWORAKOWSKI T.: Urządzenie do wywoływania obrazów kserograficznych za pomocą chmury pyłowej. Patent PRL nr 95380, Warszawa 1978

[7] JAENICKE W., LORENTZ B.: Zeitschrift für Elektrochemie 1961 nr 65

Biuletyn Informacyjny OBIEKTOWE SYSTEMY KOMPUTEROWE

3/79

mgr inż. Jerzy MOCAŁA
Instytut Maszyn Matematycznych

Komputeryzacja projektowania inżynierskiego

Automatyzacja procesu projektowania
Program KATLG dla mc MERA 400

Program KATLG przewidziany jest do wspomagania projektantu przy sporządzaniu opisów przedmiotu projektowanego. Projektant za pomocą KATLG może zapisywać cechy i elementy składowe projektowanego obiektu, a także modyfikować je dokonywać poprawek w zależności od rozwoju procesu projektowego.

Program KATLG można traktować jako "zminiaturyzowany" system informacyjny lub program tworzący bazę danych małego banku danych dla potrzeb projektowania technicznego.

Wprowadzenie

Proces projektowania można interpretować jako sporządzanie opisów przedmiotu projektowanego [2], [3]. W każdym opisie zawarty jest obraz obiektu będący częścią semantyczną informacji.

Obraz składa się z następujących grup informacji:

- cech obiektu,
- elementów składowych,
- relacji na elementach składowych, pokazującej jak elementy łączą się dając obiekt.

W dokumentacji projektowej występują obrazy:

- obraz składający się z samych cech (obraz prosty),
- obraz składający się z elementów składowych (specyfikacje),
- obraz składający się z elementów składowych i relacji (obraz morfologiczny),
- obraz składający się z cech, elementów i relacji (obraz zupełny).

Taka interpretacja procesu projektowania sprowadza stosowanie maszyn cyfrowych w procesie projektowania do dwóch podstawowych zagadnień (podobnie jak w przetwarzaniu danych [5]):

- do przetwarzania obrazów w celu ułatwienia projektantowi podjęcia decyzji projektowych,
- do przetwarzania informacji decyzyjnych.

Ingerencja maszyny cyfrowej w zagadnienie 1 nazywa się wspomaganiem procesu projektowego, w zagadnienie 2 - automatyzacją procesu.

Niezależnie od tego, czy się chce wspomagać czy automatyzować proces projektowania, musi istnieć maszynowa reprezentacja opisu obiektu.

Program KATLG dla EMC MERA 400 pozwala magazynować opisy obiektów tworzone w czasie procesu projektowania, a także przeprowadzać modyfikacje opisów. Program ten jest realizowany w języku FORTRAN IV [4].

Realizacja na maszynie MERA 400 poprzedziły implementacje programów o podobnych funkcjach na maszynach RIAD 32 i WANG 2200B.

Opis obiektu - morfologia opisu

Opis obiektu składa się:

- z identyfikatora obiektu - jest to przyjęta w ramach systemu nazwa obiektu (można interpretować także jako nazwę opisu obiektu) np. 11-80501-14,
- z nazwy potocznej obiektu - jest to nazwa w języku naturalnym, np. izolator stojący LSP-20,

- z listy cech obiektu - cecha jest to pewna wielkość opisująca (charakteryzująca) obiekt, np. długość, ciężar,
- z listy elementów składowych.

Ze względu na realizację maszynową rozróżnia się cechy o wartości typu tekst od cechy o wartości typu liczba. Cechy scharakteryzowane są więc w następujący sposób:

- dla cech o wartości tekstowej podaje się nazwę cechy i wartości cechy, np. producent (nazwa cechy) - Zakłady Mechaniczne "URSUS" (wartość cechy),
- dla cech o wartości liczbowej podaje się nazwę cechy, jednostkę i wartość (liczbę jednostek) np. ciężar, kg, 10.0.

Podobnie wygląda sprawa z elementami składowymi. Elementy składowe dzielą się na takie, które wyrażają się liczbą całkowitą i liczbą rzeczywistą.

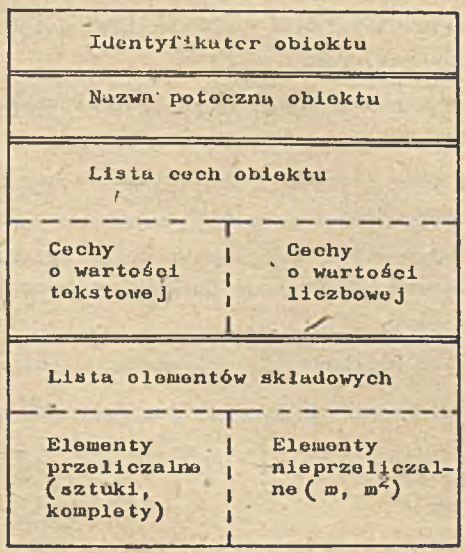
Elementy składowe wyrażające się liczbą rzeczywistą są elementami nieprzeliczalnymi. Takie elementy składowe można opisywać dwojako.

Niech A będzie obiektem, który składa się z elementów B,C,D. Element składowy C jest elementem nieprzeliczalnym, np. jest to przewód. Możemy ten element opisać podając liczbę odpowiadającą długości przewodu lub podając, że takich elementów w obiekcie jest szt. - 1, a następnie opisać element C podając cechę - długość. Elementy składowe wyrażalne liczbami całkowitymi są elementami przeliczalnymi, np. w układach elektronicznych - kondensatory, oporniki, itp.

Na rys. 1 przedstawiono zawartość opisu obiektu. Jest rzeczą oczywistą, że nie wszystkie obiekty muszą być opisywane w taki sposób. Dla niektórych wystarczy podać tylko identyfikator, a dla innych listę cech itd.

Ze względu na zawartość opisy dzielą się na:

- opis typu N - zawiera identyfikator obiektu i nazwę potoczną (może być pusta), jest to minimalny opis obiektu,
- opis typu P - zawiera identyfikator, nazwę potoczną, listę cech obiektu,
- opis typu S - zawiera identyfikator, nazwę potoczną, listę elemontów składowych,
- opis typu K - zawiera wszystkie części opisu z rys. 1, jest to maksymalny opis obiektu.



Rys. 1. Opis obiektu

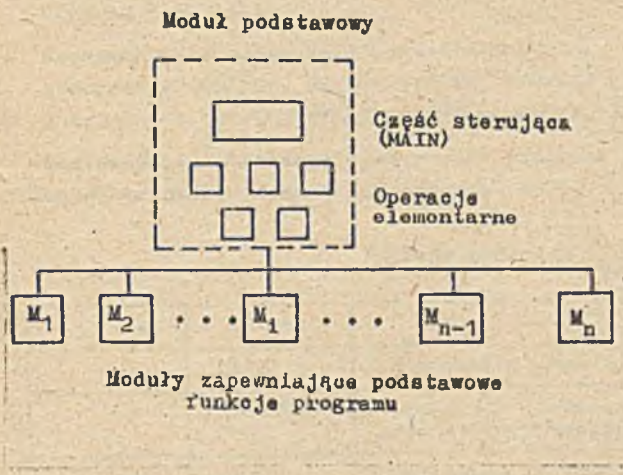
Idea rozwiązania programu

Struktura programu

Ze względu na małą pamięć maszyny MERA 400 (32 K bajtów) program będzie składał się z segmentów overlay. Przyjęto następujący podział programu na moduły (rys. 2):

- moduł podstawowy - w skład tego modułu będą wchodziły segment główny (MAIN) i procedury, które realizują operacje elementarne wspólne dla wszystkich modułów;
- moduły funkcyjne - w skład tych modułów wchodzić będą procedury realizujące przyjęte funkcje programu, tzn. jeden moduł realizuje jedną funkcję programu, np. usuwanie opisu obiektu ze zbioru opisów.

Segment główny - MAIN steruje wykonaniem poszczególnych modułów.

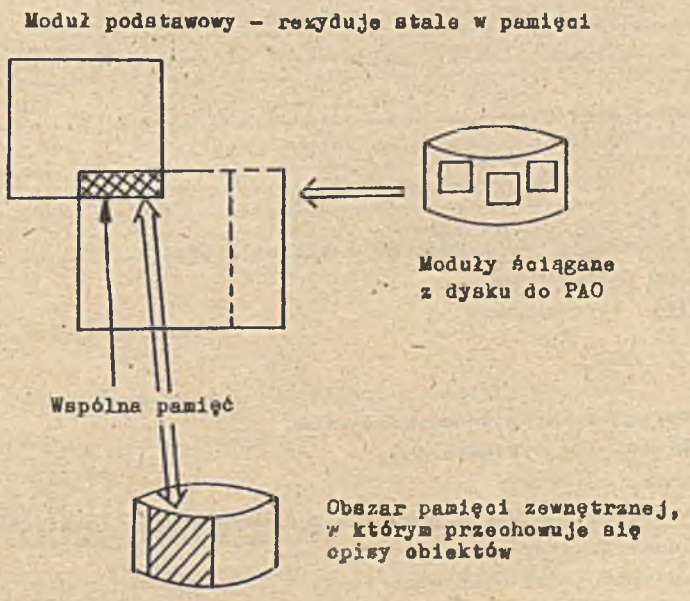


Rys.2. Idea programu - struktura

Organizacja programu

Moduł podstawowy umieszczony jest na stałe w PAO. Moduły funkcyjne przechowywane są w dysku i ściągane do PAO gdy projektant życzy sobie wykonania określonej pracy przez program(rys.3).

Moduły komunikują się za pośrednictwem wspólnego obszaru pamięci. Z tego obszaru przesyłany jest skompletowany opis obiektu (przedmiotu projektowanego) do zbioru opisów. Moduł funkcyjny nie może wywołać innego modułu funkcyjnego ani odwołać się do procedur umieszczonych w innych modułach funkcyjnych. Może jedynie wywołać procedury umieszczone w module podstawowym. Po zakończeniu pracy modułu funkcyjnego sterowanie wykonywaniem programu zawsze przekazywane jest do segmentu głównego - MAIN.



Rys. 3. Idea programu - organizacja

Algorytm przeszukiwania zbioru opisów

Magazynowanie i odszukiwanie opisów odbywa się według schematu: identyfikatory obiektów tworzą uporządkowany liniowo wektor, z każdą pozycją wektora związany jest adres miejsca, w którym przechowuje się listę cech i listę elementów składowych. Dołączenie opisu wymaga sprawdzenia, czy w wektorze nie występuje już identyfikator i ustalenia pozycji wektora, w której powinien znajdować się identyfikator, aby zachować porządek liniowy.

Odszukanie opisu wymaga przejrzenia wektora w celu znalezienia pozycji zawierającej podany identyfikator i odczytania adresu miejsca, gdzie przechowywany jest opis. Przeglądanie wektora identyfikatorów realizuje się na podstawie następującego algorytmu, przedstawionego w formie procedury algolowskiej:

```
procedure  LOOK (M,n,v,index,flag)
    value  n,v; array M;  integer  n, index;
begin  integer  i up, idown, ir;
    iup:=n;
    idown:=1;
    if  M [iup]  ≤ v  then
        begin  if  M [iup] = v  then
                    begin  index:= iup; flag:=1;
                           goto  output
                    end;
        end;
                    else  begin  index :=1: flag:=ø
                                 goto  output
                          end:
    if  M [idown]   ≥ v  then
        begin  if M [idown] =  v  then
                        begin  index:=idown; flag:=1;
                               goto output
                        end;
        end;
                        else  begin  index:=n+1; flag:=ø
                                     goto output
                              end;
loop:  ir:= iup - idown
     if  ir ≤ 1  then
        begin  index:=idown; flag:=ø;
               goto output
        end;
     index:= (iup+idown) ÷ 2;
     if  M [index] = v  then  begin  flag:=1;  goto output  end;
     if  M [index] < v  then  iup:=index
                        else  idown:=index
     goto loop;
output: end LOOK;
```

gdzie: M – wektor uporządkowanych identyfikatorów,
n – liczba opisów (identyfikatorów),
v – identyfikator obiektu,
INDEX – pozycja wektora, pod którą powinien być umieszczony identyfikator lub znajduje się już identyfikator v,
flag – ø, nie ma opisu; 1, jest opis.

Implementacja program

Funkcje programu

Przyjęto następujące podstawowe funkcje realizowane przez program:

- wykonanie i magazynowanie opisu obiektu w zbiorze opisów,
- usuwanie opisu obiektu ze zbioru opisów,
- poprawianie przechowywanego opisu,
- dostarczanie żądanego opisu obiektu,
- drukowanie zawartości zbioru opisów.

Struktura programu

Program KATLG składa się:

- z segmentu głównego MAIN (część sterująca)
- z procedur LOOKFOR, ORDER, FIRSTA, COMPRES (operacje elementarne),
- z segmentów overlay FILLER, GETOUT, MODYF, INFO, COPY (moduły funkcyjne).

Zastosowanie poszczególnych części programu przedstawiono w tab. 1, na rys. 4 zobrazowano realizację sterowania wykonaniem programu KATLG.

Tab. 1.

Funkcje procedur i segmentów programu KATLG

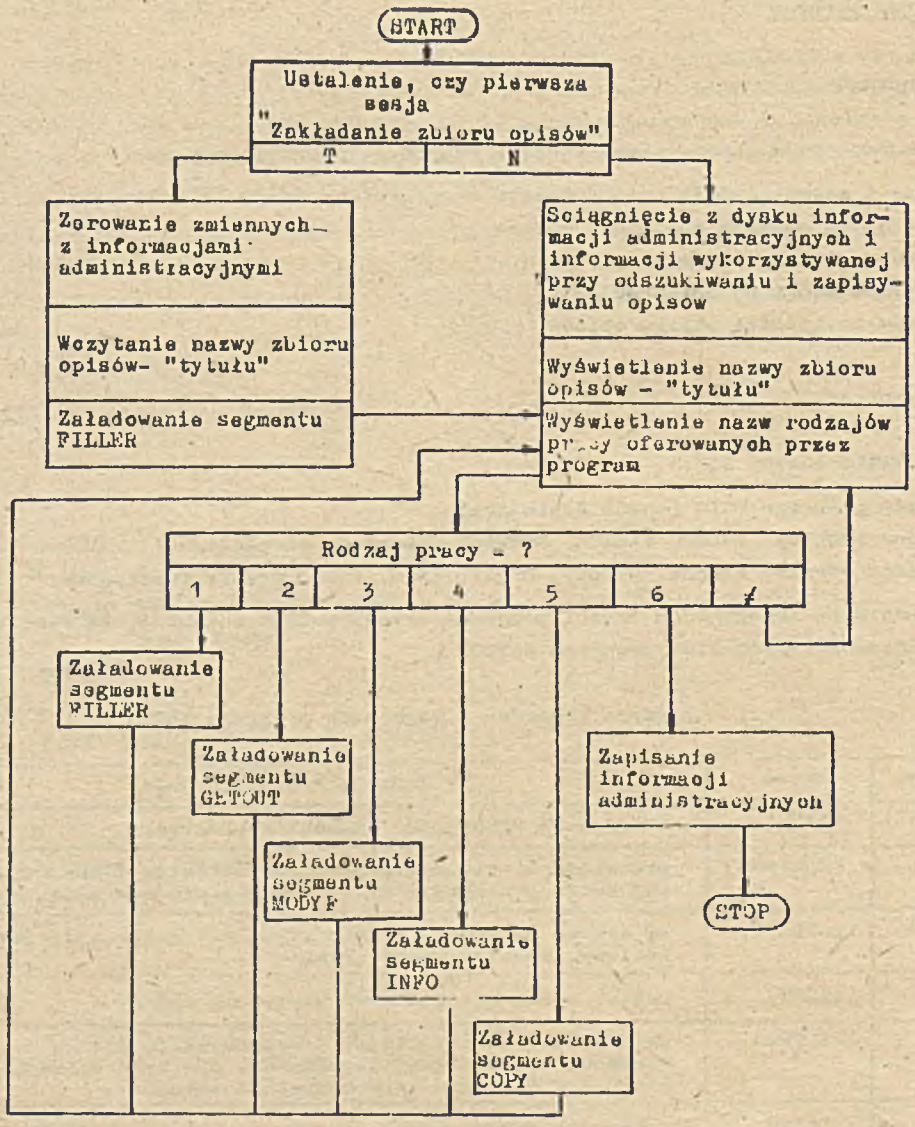
Lp.	Nazwa	Funkcja - zastosowanie
1	MAIN	Sterowanie wykonaniem segmentów programu
2	LOOKFOR	przegląda listę identyfikatorów obiektów i znajduje pozycję, pod którą przechowywany jest adres miejsca opisu
3	ORDER	umieszcza na liście identyfikatorów, zachowując porządek liniowy, identyfikator obioktu
4	FIRSTA	ustala adres miejsca przechowywania opisów
5	COMPRES	ustala nowe adresy miejsc przechowywania opisów i przemieszcza opisy w taki sposób, aby nie było wolnych miejsc (dziur) w obszarze przechowywania opisów
6	FILLER	wykonanie opisu obiektu i dołączenie opisu do zbioru opisów
7	GETOUT	usuwanie opisu ze zbioru opisów
8	MODYF	modyfikacja opisu umieszczonego w zbiorze opisów
9	INFO	dostarczanie wybranego opisu obiektu
10	COPY	drukowanie zbioru przechowywanych opisów

Organizacja przechowywania opisów

Opis projektowanego obiektu został podzielony na cztery części:

- identyfikator obiektu i nazwa potoczna,
- lista cech tekstowych (C_t),
- lista cech liczbowych (C_l),
- lista elementów składowych składająca się z elementów przeliczalnych i nieprzeliczalnych (E).

Identyfikator obiektu i nazwa potoczna pamiętane są w części zbioru nazwanej SPISEM TREŚCI, lista cech tekstowych i lista cech liczbowych magazynowane są w części zbioru nazwanej obszarem opisów "C", a lista elementów składowych w części zbioru - obszar opisów "E" (rys. 5). SPIS TREŚCI oprócz identyfikatorów zawiera dodatkowo trzy adresy miejsc przechowywania C_t, C_l, E.

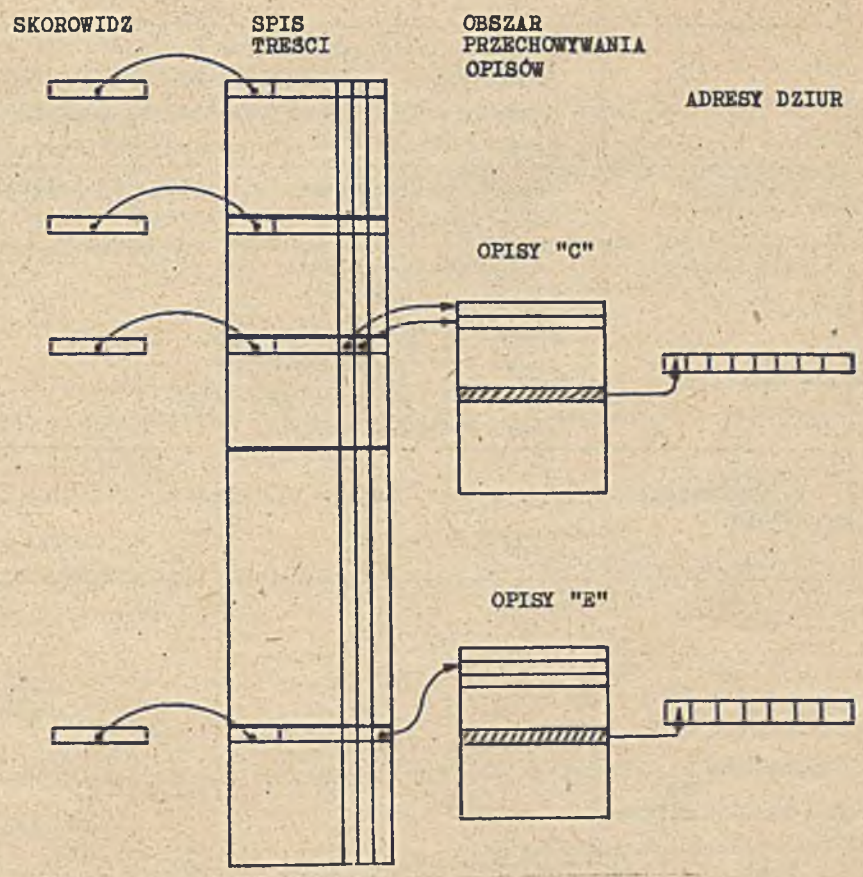


Rys. 4. Ogólny schemat blokowy segmentu MAIN

Wymazanie części opisu (C_1, C_1, E) polega na wpisaniu ø w miejsce adresu w SPISIE TREŚCI. Powoduje to powstanie wolnego miejsca (dziury) w obszarze przechowywania opisów. Adres tego miejsca pamiętany jest w wektorze dziur, gdy potrzeba zapamiętać opis w zbiorze sprawdza się, czy są dziury; gdy są, przydzielane jest miejsce z wektora dziur, a gdy nie ma - przydzielane jest kolejne miejsce w obszarze przechowywania opisów ("C" lub "E").

Odszukanie żądanego opisu polega na przeglądaniu identyfikatorów w SPISIE TREŚCI według algorytmu z punktu "Algorytm przeszukiwania zbioru opisów". SPIS jest podzielony na mniejsze jednostki nazwane stronami. Podczas przeglądania do PAO zostaje ściągana zawsze jedna strona SPISU, żeby zminimalizować liczbę dostępów do dysku tworzony jest wektor nazwany SKOROWIDZ. Zawiera on identyfikatory obiektów, które występują na pierwszych pozycjach stron SPISU.

Przeglądanie odbywa się więc dwustopniowo - najpierw ustala się numer strony, na której może znajdować się identyfikator (na podstawie wektora SKOROWIDZ), ściąga się ją do pamięci i przegląda się stronę, aby ustalić pozycję na stronie z szukanym identyfikatorem.



Rys. 5. Zasady działania programu KATLG

Organizacja informacji przechowywanej na dysku

Pamięć zewnętrzna podzielona jest na 4 części (rys. 6). Pojemność tych części jest następująca:

- ADMINISTRACJA - 3 sektory,
- SKOROWIDZ - 3 sektory,
- SPIS TREŚCI - 384 sektory,
- OBSZAR PRZECHOWYWANIA OPISÓW - 4000 sektorów.

Zawartość (funkcje) poszczególnych części przedstawiono w tab. 2.

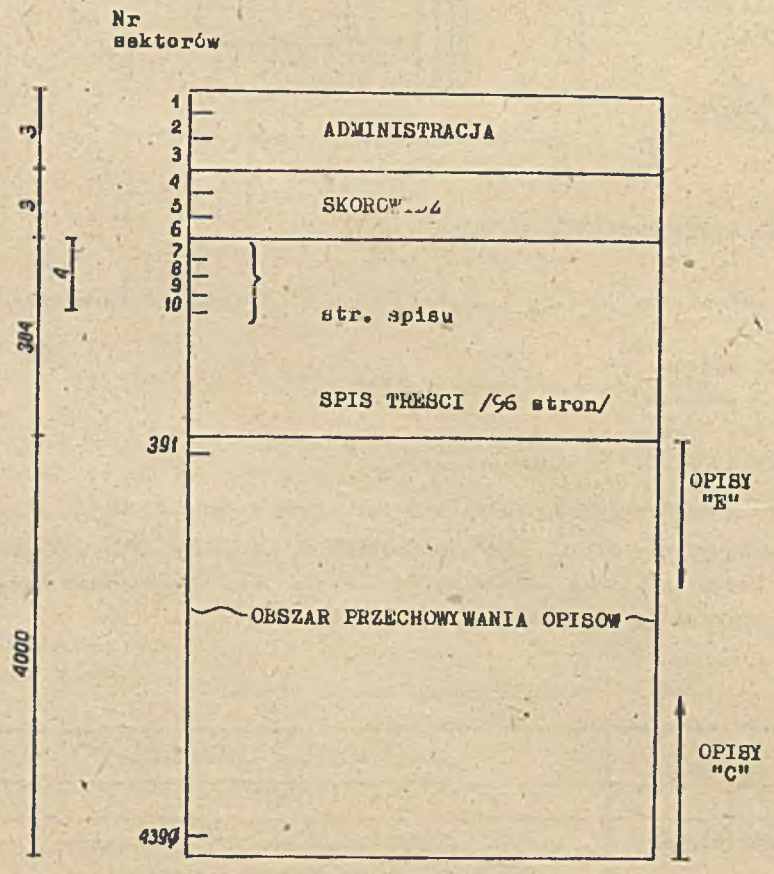
Listę cech tekstowych (C_t) i listę cech liczbowych (C_l) wpisuje się do obszaru opisów począwszy od sektora 4000, natomiast listę elementów od sektora 391. W programie istnieje zabezpieczenie uniemożliwiające pokrywanie się obszarów opisów "C" i "E".

Tab. 2.

Funkcje części zbioru opisów

Nazwa części	Przechowywana informacja
1	2
ADMINISTRACJA	Liczba opisów zgromadzonych w katalogu Zapełnienie SPISU TRESCI - liczba stron, numer pozycji na ostatniej stronie "Tytuł" zbioru opisów

1	2
	Liczba dziur w podobszarze E, adresy dziur, Liczba dziur w podobszarze C, adresy dziur Adres ostatniego zajętego miejsca w części E, Adres ostatniego zajętego miejsca w części C
SKOROWIDZ	Identyfikatory obiektów z pierwszej pozycji stron spisu
SPIS TREŚCI	Identyfikator obiektu Typ opisu (N,P,S,K) Adres "C_1" (cechy liczbowe) Adres "C_t" (cechy znakowe) Adres "E" (elementy składowe) Nazwa potoczna obiektu
OBSZAR PRZECHOWYWANIA OPISÓW	"C_1", "C_t", "E"



Rys. 6. Organizacja zbioru opisów obiektu

Ograniczenia programu

Podczas realizacji programu przyjęto następujące wielkości:

- liczba opisów - 3072 (pojemność SPISU TREŚCI),
- identyfikator obiektu - 16 znaków,
- nazwa potoczna obiektu - 32 znaki,
- liczba cech - 16 cech liczbowych,
  10 cech znakowych,
- nazwa cechy liczbowej - 10 znaków,
- jednostka cechy liczbowej - 14 znaków,
- wartość cechy liczbowej - liczba rzeczywista,
- nazwa cechy znakowej - 20 znaków,
- wartość cechy znakowej - 30 znaków,
- liczba róznych elementów składowych - 32 znaki,
- rodzaj elementu - 2 znaki,
- typ opisu - 2 znaki.

Przyjęte wartości są wypadkową przewidywanych potrzeb projektanta, możliwych typów zmiennych, występujących w języku FORTRAN IVE oraz dążeniom do możliwie maksymalnego wypełnienia sektorów w pamięci dyskowej.

Poszczególne typy zmiennych występujące w języku mają następujące pojemności:

- integer - 1 słowo (2 bajty),
- real - 3 słowa (6 bajtów),
- double precision - 6 słów (12 bajtów),
- logical - 1 słowo (2 bajty).

Sektor zawiera 256 słów (512 bajtów).

Praca z programem KATLG

Program jest przeznaczony do pracy na zestawie sprzętu, w skład którego wchodzi:

- pamięć dyskowa MERA 9425,
- konsola operatorska DZM-180-KSR lub Videoton 340,
- drukarka znakowo-mozaikowa DZM-180,
- czytnik taśmy papierowej CT 1001A,
- dziurkarka taśmy papierowej DT 105S.

Wywołanie programu KATLG odbywa się według zasad opracowanych dla maszyny MERA 400 [4].

Po załadowaniu programu do PAO sterowanie przejmuje program i na drodze konwersacyjnej ustala rodzaj pracy, jaką chce wykonać projektant, a także wartości danych. Dialog maszyny z projektantem można zaliczyć według klasyfikacji z [1] do dialogu hybrydowego.

Ustalanie rodzaju pracy odbywa się według zasad "wyboru z repertuaru", natomiast wprowadzanie opisu odbywa się według metody "proste pytania" i "wyświetlanie wzorów danych".

Przykład dialogu maszyna cyfrowa - projektant.

Wyświetla się (lub drukuje) napis:

<*>< > PROGRAM KATLG < > <*>

RODZAJ PRACY: 1 - WPROWADZANIE 4 - OPIS
2 - WYMAZANIE 5 - KOPIA
3 - MODYFIKACJA 6 - KONIEC PRACY

NUMER?

Należy podać numer wybranej pracy, np. 1. Na ekranie (od nowej strony) wyświetli się napis:

<*>< >WPROWADZANIE < > <*>

(16 kropek)
..... IDENTYFIKATOR

Należy podać (pod kropkami) 16-znakowy identyfikator obiektu.

Wyświetla się napis:

(32 kropki)
..... NAZWA POTOCZNA

Należy podać nazwę obiektu (może być pusta).
Po wprowadzeniu nazwy wyświetli się napis:

LICZBA CECH ZNAKOWYCH:

Gdy liczba cech większa od zera pojawia się napis:

(20 kropek)
..... NAZWA CECHY (Z) - XX?

Pod kropkami należy podać co najwyżej 20-znakową nazwę (XX - oznacza kolejny numer wprowadzonej cechy znakowej).

Zastosowanie programu

Projektant za pomocą programu KATLG może tworzyć zbiór opisów projektowanego obiektu, zawierających informacje o cechach i elementach składowych. Informacje te mogą posłużyć do:

- sprawdzaniu poprawności proponowanego rozwiązania projektowego, np. czy elementy składowe faktycznie dają obiekt o podanych cechach,
- automatycznego rozwijania opisów projektowanego obiektu, np. projektant podaje jedynie pierwszy poziom elementów składowych, a maszyna uzupełnia opis przez podanie niższych poziomów struktury obiektu (elementy składowe elementów składowych).

Program można wykorzystać także do tworzenia katalogu elementów gotowych, wykorzystywanych przez projektanta w procesie projektowym. Realizacja programu na maszynie WANG 2200B posłużyła do założenia katalogu elementów napowietrznych linii elektroenergetycznych. Na podstawie opisów typu S zawartych w katalogu sporządza się wykazy montażowe, przeznaczone dla wykonawcy. Automatyzacja tej działalności projektowej pozwoliła uwolnić projektanta od nadmiaru informacji i skróciła czas potrzebny na otrzymanie dokumentacji projektowej.

Literatura

[1] MARTIN J.: Dialog człowieka z maszyną cyfrową. WNT: Warszawa 1976

[2] MOCAŁA J.: Pewien warunek realizacji komputerowego wspomagania projektowania. Ogólnopolska Konferencja Projektowania III, Wrocław, 20-22.09.1978

[3] MOCAŁA J.: Zagadnienie modelu procesu projektowania w komputeryzacji projektowania technicznego. IMN: Warszawa 1977 Archiwum Opracowań nr 35

[4] FORTRAN IV E. Procesory systemowe systemu SOM-3. Opis użytkowy OBRSM MERA-ZSM. S-OF-4-0000/-OOD. Warszawa 1977

[5] TARGOWSKI A.: Automatyczne przetwarzanie danych. Systemy. Technika. Metody. PWN: Warszawa 1973

BRANŻOWY OŚRODEK INFORMACJI NAUKOWEJ TECHNICZNEJ I EKONOMICZNEJ
INSTYTUTU MASZYN MATEMATYCZNYCH
02-078 Warszawa, ul. Krzywickiego 34, tel. 21-84-41 w. 391

BOINTE udziela informacji

z zakresu techniki komputerowej

BOINTE wydaje

informacja ekspresowa

OBIEKTOWE SYSTEMY KOMPUTEROWE

przegląd dokumentacyjny

OBIEKTOWE SYSTEMY KOMPUTEROWE

biuletyn informacyjny

OBIEKTOWE SYSTEMY KOMPUTEROWE

Materiały konferencyjne, szkoleniowe, prospekty

BOINTE gromadzi

wydawnictwa zwarte, czasopisma krajowe i zagraniczne, katalogi i prospekty, sprawozdania z prac naukowo-badawczych oraz inne materiały informacyjne

BOINTE wykonuje usługi reprodukcyjne i poligraficzne

fotokopie, mikrofilmy, kserokopie z zakresu posiadanych zbiorów

Cena zł. 140. -

WARUNKI PRENUMERATY

Prenumeratę na kraj przyjmują Oddziały RSW "Prasa-Książka-Ruch" oraz urzędy pocztowe i doręczyciele w terminie do dnia 25 listopada na rok następny.

Cena prenumeraty rocznej zł 840.

Jednostki gospodarki uspołecznionej, instytucje, organizacje i wszelkiego rodzaju zakłady pracy zamawiają prenumeratę w miejscowych Oddziałach RSW "Prasa-Książka-Ruch", w miejscowościach zaś, w których nie ma Oddziałów RSW - w urzędach pocztowych.

Czytelnicy indywidualni opłacają prenumeratę wyłącznie w urzędach pocztowych i u doręczycieli.

Prenumeratę ze zleceniem wysyłki za granicę przyjmuje RSW "Prasa-Książka-Ruch", Centrala Kolportażu Prasy i Wydawnictw, ul. Towarowa 28, 00-958 Warszawa, konto PKO Nr 1153-201045.

Prenumerata ze zleceniem wysyłki za granicę jest droższa od prenumeraty krajowej o 50% dla zleceniodawców indywidualnych i o 100% dla zlecających instytucji i zakładów pracy.

INDEKS 35362