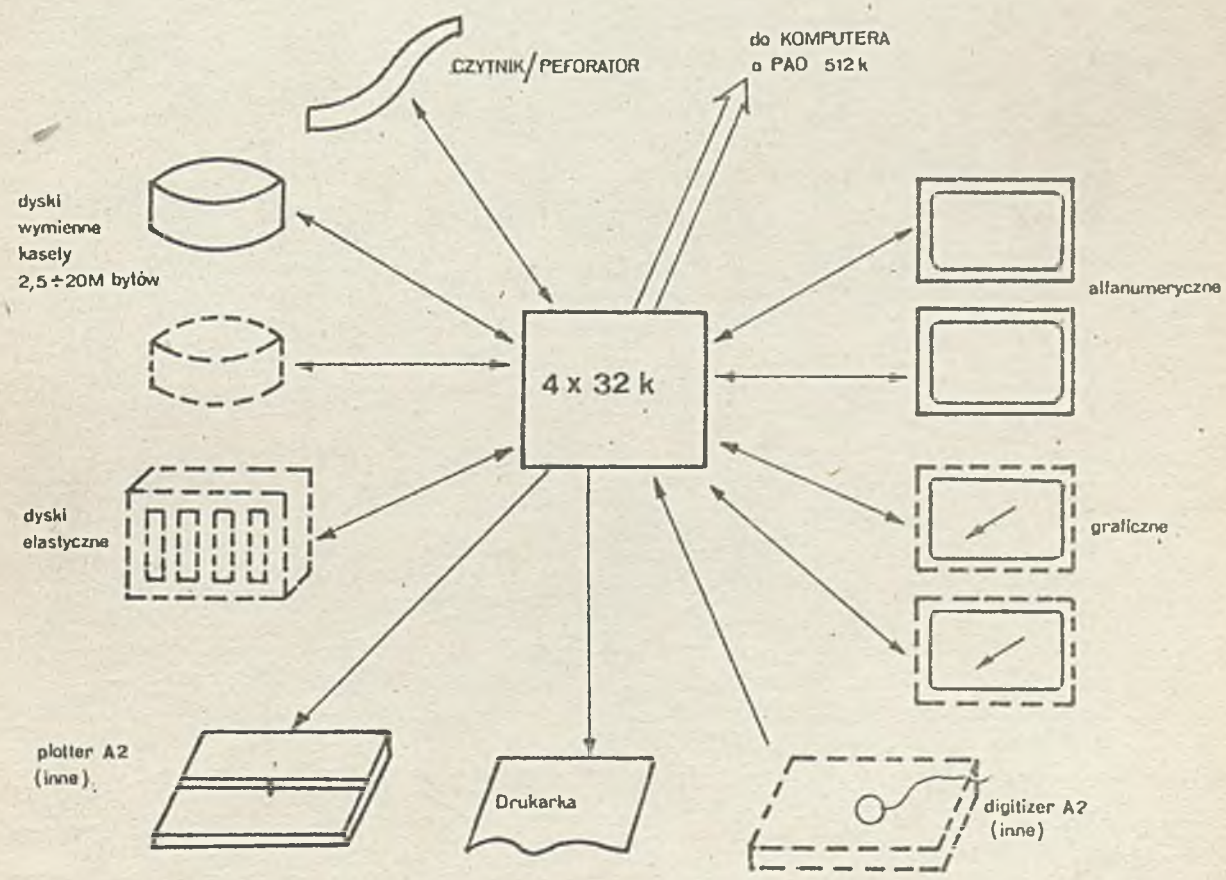
biuletyn informacyjny

4 '79

MERA IMM

OBIEKTOWE SYSTEMY KOMPUTEROWE



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

Na okładce: Modułowa struktura lokalnego systemu minikomputerowego dla KWP

Druk okładki IMM zam. 64/80 nakł. 820

Biuletyn Informacyjny OBIEKTOWE SYSTEMY KOMPUTEROWE

Rok XVII Nr 4 1979

Spis treści

Содержание

Contents

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.21-84-41 w.431 lub 28-37-29

doc. dr hab. inż. Jan BIAŁASIEWICZ
Instytut Badań Jądrowych

Systemy operacyjne czasu rzeczywistego dla perspektywicznych systemów minikomputerowych

Wprowadzenie

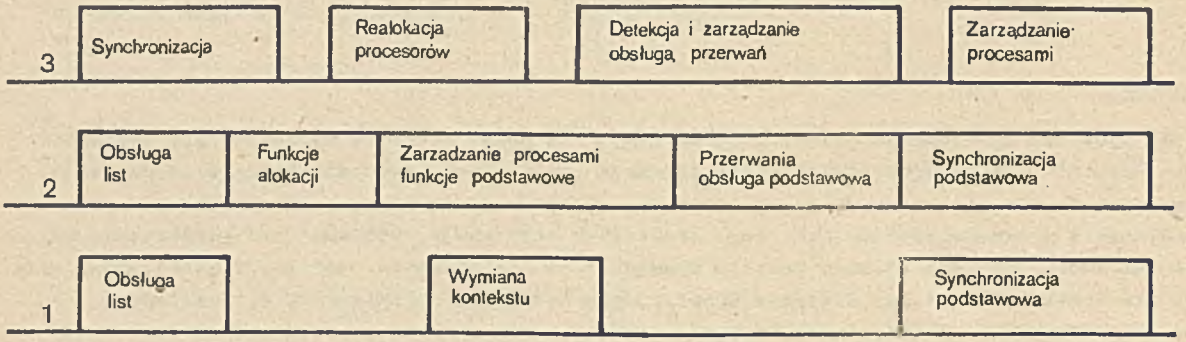
Rozwój systemów minikomputerowych, przeznaczonych do pracy w czasie rzeczywistym, zmierza w kierunku systemów wieloprocesorowych z inteligencją rozproszoną. Taka struktura systemu komputerowego oraz realizacja niezbędnego oprogramowania użytkowego musi znaleźć odpowiednie odzwierciedlenie w systemie operacyjnym. Jego prawidłowa realizacja, zapewniająca odpowiednią efektywność działania, narzuca różnego rodzaju wymagania na architekturę systemu. Zagadnieniami tymi zajmuje się Komitet Techniczny Systemów Operacyjnych Czasu Rzeczywistego TC 8, działający w ramach European Workshop on Industrial Computer Systems. Niniejsza praca relacjonuje opracowany przez ten Komitet model systemu operacyjnego czasu rzeczywistego oraz sygnalizuje jego wpływ na strukturę i rozwój systemów minikomputerowych. Kompletny opis tego modelu zawiera praca "The up-to-date TC 8 Report" [1].

Model systemu operacyjnego czasu rzeczywistego

W większości systemów czasu rzeczywistego występuje zbiór komunikujących się ze sobą procesów realizowanych przez zbiór zawierający jeden lub więcej procesorów. Maksymalna liczba procesorów, które mogą być realizowane fizycznie równolegle jest oczywiście równa liczbie sprzętowo równoważnych procesorów. Natomiast system operacyjny musi dostarczyć pseudo-procesora każdemu procesowi, który może być realizowany niezależnie od innych procesów. Wybór struktury danych i funkcji wewnętrznych, zdefiniowanego dalej jądra systemu operacyjnego, musi być taki, aby synchronizacja i komunikacja procesów i procesorów była realizowana w sposób najbardziej efektywny. Zatem obiekty danych wspólnych oraz te części funkcji, które nie mogą być równocześnie realizowane przez różne procesory, powinny być jak najmniejsze w celu zapewnienia optymalnych warunków realizacji procesów w sytuacjach współdostępu.

Realizacja zasady współbieżności wymaga posiadania przez system operacyjny funkcji umożliwiających synchronizację oraz komunikację procesów asynchronicznych, jak również funkcji przydzielających procesory procesom.

Komitet Techniczny Systemów Operacyjnych Czasu Rzeczywistego TC 8, biorąc pod uwagę wymagania niezawodnościowe oraz aspekty języków wysokiego poziomu, opracował otwarty wielowarstwowy model systemu operacyjnego czasu rzeczywistego. Przez termin otwarty rozumie się możliwość rozbudowy przez dodawanie coraz to nowych warstw modelu. Zdefiniowany model jest modelem logicznym, to znaczy funkcje umieszczone na poszczególnych poziomach mogą być realizowane przez odpowiednią kombinację układów i oprogramowania. Pierwsze trzy warstwy tworzą jądro systemu przedstawione na rys. 1.



Rys. 1. Jądro systemu operacyjnego

Wprowadzona struktura warstwowa systemu operacyjnego zakłada możliwość przenoszenia funkcji znajdujących się w danej warstwie do warstwy następnej w postaci niezmienionej lub rezygnacji z dostępu do nich w warstwie następnej. Nowe funkcje wprowadzane w danej warstwie są implementowane przy wykorzystaniu funkcji dostępnych w warstwie poprzedniej.

Obsługa list

Proponowane funkcje obsługi list ujmuje tab. 1.

Tab. 1

Funkcja	Działanie
INSERT (el,list)	Wprowadza element 'el' do listy 'list'
REMOVE (el,list)	Zgodnie z jakąś strategią wybiera i usuwa jakiś element z listy 'list' i umieszcza go według 'el'
LISTSIZE (list,length)	Umieszcza według 'length' długość listy 'list'
NEXT (el,list)	Wybiera z listy 'list' element następny w stosunku do 'el'
SEARCHFOR (el,list, id_attribute)	Usuwa z listy 'list' jeden element o identyfikatorze 'id_attribute' i umieszcza go według 'el'
CHANGE (list,id_attribute, change_attribute)	Dla wszystkich elementów z listy 'list' zmienia jego wartość na 'change_attribute'

Synchronizacja podstawowa

W systemie operacyjnym czasu rzeczywistego muszą istnieć na różnych poziomach różne mechanizmy synchronizacyjne, przy czym zawsze wyższy mechanizm synchronizacyjny powstaje przy wykorzystaniu mechanizmu bardziej prymitywnego. Najprymitywniejszy mechanizm synchronizacyjny musi dać możliwość wykluczenia jednoczesnego dostępu przez różne procesy do danych wspólnych. Wszystkie sekwencje instrukcji, umożliwiające dostęp do danych wspólnych, powinny mieć przy-

dzieloną tę samą zmienną zamykającą v. Daje to możliwość uczynienia każdej z tych sekwencji nieprzerywalną. W tym celu należy zrealizować funkcje synchronizujące, operujące na zmiennej v, przy czym wykonanie funkcji LOCK(v) otwiera wybraną do wykonania i nieprzerywalną sekwencję instrukcji, a wykonanie funkcji UNLOCK(v) zamyka tę sekwencję. Dopiero po UNLOCK(v) możliwe jest ponowne wykonanie tej samej sekwencji instrukcji lub też innej sekwencji instrukcji opatrzonej tą samą zmienną zamykającą v. Dzięki takiemu mechanizmowi mamy możliwość wykluczenia jednoczesnego korzystania przez różne procesy z tej samej sekwencji instrukcji, jak również równoległego (na różnych procesorach) lub pseudorównoległego (na tym samym procesorze) wykonywania sekwencji instrukcji (procesów) wzajemnie wykluczających się (opatrzonych tą samą zmienną zamykającą).

W systemach z jednym procesorem fizycznym funkcje LOCK i UNLOCK mogą być zrealizowane przez blokowanie przerwań. Funkcje LOCK i UNLOCK powinny być implementowane układowo.

Funkcja LOCK zawiera mechanizm czekania w pętli. Jednakże w pewnych sytuacjach może być konieczne wykonanie sekwencji alternatywnej zamiast po prostu oczekiwania. W tym celu wprowadzono funkcję LOCK_OR_BRANCH(v,etykieta). Oczywiście argument `etykieta' identyfikuje sekwencję alternatywną.

Wprowadzono również, podobne do LOCK i UNLOCK, funkcje

EXCLUDE(v,w) i ALLOW(v,w)

umożliwiające wykluczenie współczesnego wykonania grupy operacji opatrzonej zmienną zamykającą v i grupy operacji opatrzonej zmienną zamykającą w. W wielu wypadkach (wyłącznie w omawianych dalej systemach z realokacją) może wystąpić potrzeba zawieszenia, na czas wykonywania pewnej sekwencji instrukcji, możliwości asynchronicznego wykonywania innych funkcji przez ten procesor. W tym celu zdefiniowano funkcje

INHIBIT_NOTIFY i ENABLE_NOTIFY

Wymiana kontekstu

Kontekst procesu, umożliwiający jego start lub wznowienie, musi być zachowany i zastąpiony kontekstem innego procesu zawsze wtedy, gdy procesor zostanie przydzielony innemu procesowi lub nawet wówczas, gdy przydział pozostaje niezmieniony, ale procesor zaczyna realizację jednej z funkcji jądra. Wymianę kontekstu procesów realizują funkcje SAVE_CONTEXT i RESTORE_CONTEXT. Inną parę funkcji definiuje się dla wymiany kontekstu funkcji jądra.

Funkcje alokacji

W wieloprocesorowym systemie komputerowym można wydzielić podzbiory procesorów, zwane polami, charakteryzujące się tym, że z każdym z nich związany jest ściśle określony podzbiór procesów. Z kolei każdy proces ma deskryptor specyfikujący

- pole procesorów, przy użyciu którego proces ten może być realizowany,
- pamięć do przechowywania jego kontekstu.

Natomiast każde pole procesorów zaopatrzone jest

- w listę procesów realizowanych przez poszczególne procesory,
- w listę procesów gotowych (READYLIST), które są gotowe do realizacji przez to pole.

Alokacja procesorów jest realizowana za pomocą następujących funkcji:

ASSIGN - przekazuje sterowanie procesorem wykonującym tę funkcję procesowi znajdującemu się na liście procesów gotowych. Mówiąc inaczej zmienia stan procesu z GOTOWOŚĆ na BIEG.

BLOCK - odłącza proces wykonujący tę funkcję od procesora, który został mu poprzednio przydzielony. Mówiąc inaczej, zmienia stan procesu z BIEG na ZAWIESZENIE.

READY - zmienia stan procesu z ZAWIESZENIE na GOTOWOŚĆ.

Dodatkowe funkcje alokacji są potrzebne w systemach, w których przewiduje się zabranie sterowania procesowi w stanie BIEG w celu przekazania sterowania tym procesorem procesowi innemu, w danej chwili ważniejszemu. Są to funkcje następujące:

RESIGN - zmienia stan procesu z BIEG na GOTOWOŚĆ, to znaczy umieszcza go na liście procesów gotowych,

NOTIFY - zmienia stan procesora z PRZYDZIELONY na ZAWIADOMIONY i powoduje wykonanie przez ten procesor, opisanej dalej, funkcji REDISPATCH,

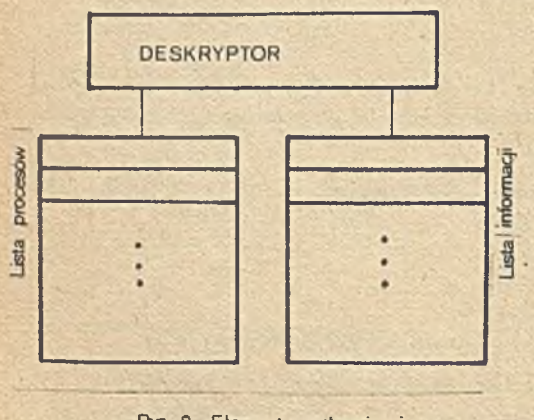
RESUME - zmienia stan procesora wykonującego tę funkcję z ZAWIADOMIONY na PRZYDZIELONY, to znaczy powoduje ponowne przejęcie sterowania przez proces poprzednio przez ten procesor realizowany.

Realokacja procesorów

W systemach z możliwością realokacji jej wykonanie może być potrzebne w chwili, gdy na liście procesów gotowych zostanie umieszczony nowy proces. W tym celu wprowadzono funkcję REDISPATCH. Procesor wykonujący tę funkcję znajduje się w stanie ZAWIADOMIONY lub WOLNY. Jeśli znajduje się on w stanie WOLNY, to REDISPATCH sprowadza się do wykonania ASSIGN. Natomiast jeśli stan procesora jest ZAWIADOMIONY, to REDISPATCH sprowadza się do sprawdzenia przez procesor czy ważniejsze jest wykonanie RESUME, to znaczy wznowienie realizowanego przez niego procesu, czy też należy wykonać RESIGN, a następnie przez wykonanie ASSIGN przekazać sterowanie innemu procesowi.

Funkcje synchronizacji

W omawianym modelu systemu operacyjnego czasu rzeczywistego trzecia warstwa jądra zawiera funkcje synchronizacji umożliwiające koordynację procesów na poziomie, na którym procesory fizyczne są odłączane od procesów przechodzących w stan ZAWIESZENIE. Zaproponowano element synchronizacji zilustrowany na rys. 2. Na elemencie tym działają dwie funkcje synchronizacji INC i DEC. Zdefiniujemy teraz te funkcje.



Rys. 2. Element synchronizacji

Funkcja INC (element_synchronizacji,info) zdefiniowana jest następująco:

lista procesów
- = pusta — umieść informację na liście informacji
- ≠ pusta — usuń jeden proces z listy procesów elementu synchronizacji; przekaż mu informację (info) i umieść go na liście procesów gotowych

Funkcja DEC (element_synchronizacji,info) zdefiniowana jest następująco:

lista informacji
- = pusta — umieść proces wykonujący tę funkcję na liście procesów elementu synchronizacji i zawieś go
- ≠ pusta — usuń jeden element z listy informacji elementu synchronizacji i przekaż go procesowi wykonującemu tę funkcję.

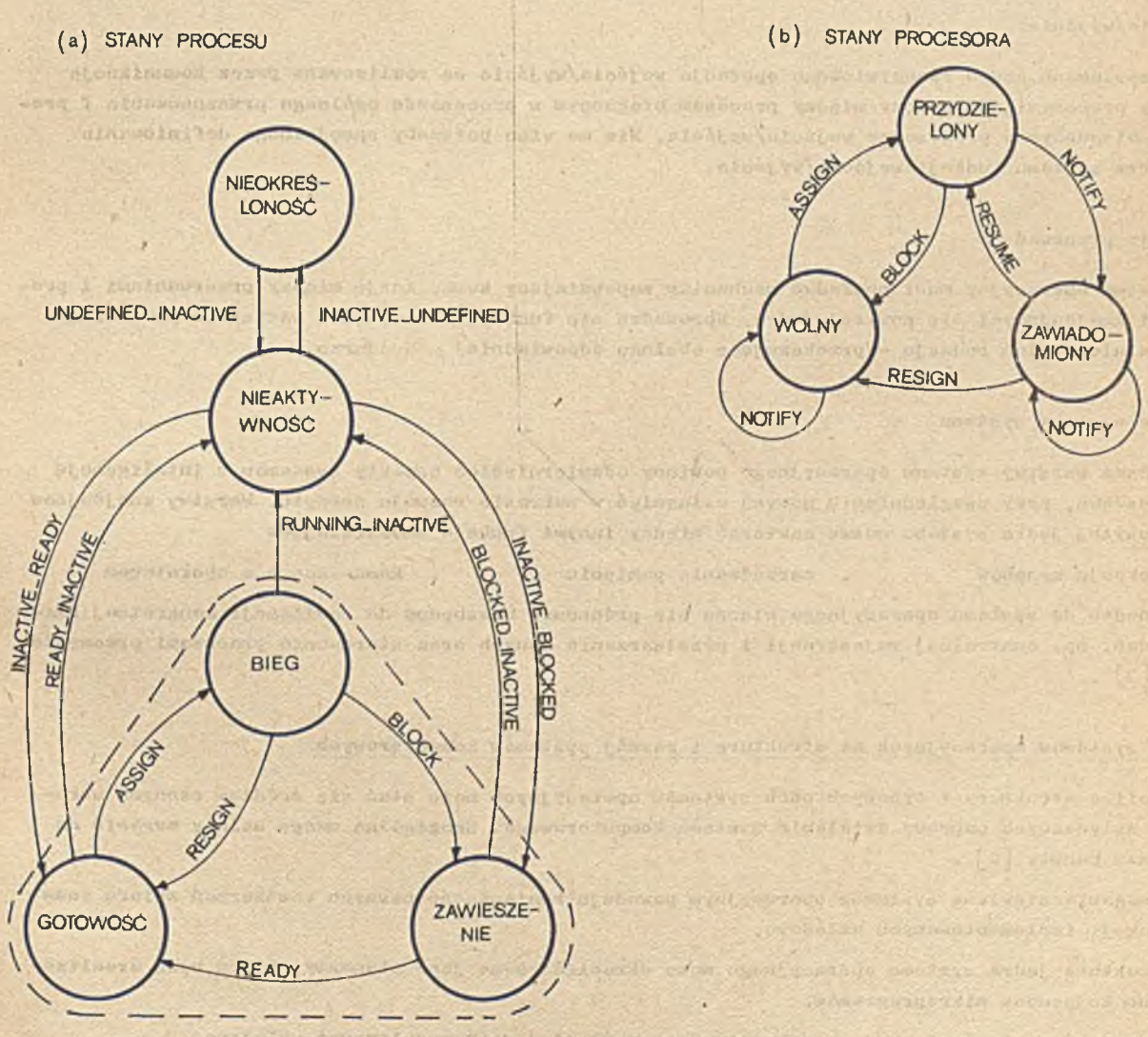
Zdefiniowany element synchronizacji oraz operujące na nim funkcje INC i DEC mogą być wykorzystane do realizacji innych mechanizmów synchronizacyjnych, takich jak semafory, komunikaty i monitory.

Stany procesów i procesorów. Funkcje podstawowe zarządzania procesami

W celu uzyskania możliwości asynchronicznego zawieszania procesu przez inny proces, co stanowi warunek dynamicznej alokacji zasobów (w tym również pamięci operacyjnej) oraz właściwej obsługi przerwań, wprowadza się dwa nowe stany procesów, a mianowicie

stan NIEAKTYWNOŚĆ i stan NIEOKREŚLONOŚĆ.

To uzupełnienie zmienia model stanów procesu w stosunku do modelu obowiązującego dotychczas. Model ten przedstawiony jest na rys. 3a z tym, że linią przerywaną obwiedziono model wprowadzo-



Rys. 3 Stany procesu i procesora

ny poprzednio. Model stanów procesora nie ulega zmianie i jest przedstawiony na rys. 3b. Uzupełnienie modelu stanów procesu dwoma nowymi stanami prowadzi do konieczności zdefiniowania funkcji podstawowych zarządzania procesami, umożliwiających zmiany stanów. Rys. 3a zawiera ich nazwy (znajdujące się na zewnątrz linii przerywanej) i w zasadzie wyjaśnia cel ich działania. Funkcje te uzupełniają drugą warstwę jądra systemu operacyjnego.

Zarządzanie procesami

W trzeciej warstwie jądra systemu operacyjnego, w której znajdują się funkcje INC i DEC wprowadza się funkcje zarządzania procesami. Przede wszystkim wzajemna dezaktywizacja (niezależnie od rodzaju stanu pierwotnego) lub aktywizacja jednego procesu przez drugi dokonywana jest przez odpowiednio zdefiniowane funkcje DEACTIVATE i ACTIVATE. Na tym poziomie wprowadza się też inne funkcje zarządzania procesami.

Wejście/wyjście

W systemach czasu rzeczywistego operacje wejścia/wyjścia są realizowane przez komunikację między procesami, to znaczy między procesem biegnącym w procesorze ogólnego przeznaczenia i procesem biegnącym w procesorze wejścia/wyjścia. Nie ma więc potrzeby specjalnego definiowania w jądrze systemu funkcji wejścia/wyjścia.

Obsługa przerwań

System operacyjny musi posiadać mechanizm zapewniający komunikację między przerwaniami i procesami znajdującymi się powyżej jądra. Wprowadza się funkcję odbierającą wszystkie przerwania i zależnie od ich rodzaju - przekazującą obsługę odpowiedniej procedurze.

Wyższe warstwy systemu

Wyższe warstwy systemu operacyjnego powinny odzwierciedlać aspekty systemów z inteligencją rozproszoną, przy uwzględnieniu nowych osiągnięć w zakresie rozwoju sprzętu. Warstwy znajdujące się powyżej jądra systemu muszą zawierać między innymi funkcje umożliwiające

. alokację zasobów . zarządzanie pamięcią . komunikację z operatorem

Ponadto do systemu operacyjnego włącza się procedury niezbędne do realizacji konkretnej klasy zadań, np. centralnej rejestracji i przetwarzania danych oraz sterowania procesami przemysłowymi [3] .

Wpływ systemów operacyjnych na strukturę i rozwój systemów komputerowych

Analiza struktury i typowych cech systemów operacyjnych może stać się źródłem cennych wskazówek dotyczących poprawy działania systemu komputerowego. Szczególną uwagę należy zwrócić na poniższe punkty [2] .

- Wymagania stawiane systemom operacyjnym powodują konieczność pewnych rozszerzeń zbioru instrukcji implementowanych układowo.
- Struktura jądra systemu operacyjnego może określić pewne jego elementy, które będą zrealizowane za pomocą mikroprogramów.
- Zawarte w systemie funkcje synchronizujące najlepiej jest zrealizować układowo.
- Niezbędne mechanizmy obsługi przerwań powinny być odpowiednio uwzględnione w rozwiązaniach układowych.
- Implementację funkcji obsługi list ułatwia zastosowanie pamięci asocjacyjnej.

- Analiza pożądanego działania systemu operacyjnego dostarcza wskazówek dotyczących właściwego doboru rozwiązań układowych i programowych przy realizacji układu priorytetów.
- Przedstawiony model systemu operacyjnego może służyć jako narzędzie oceny architektury systemów komputerowych.

Literatura

[1] The up-to-date TC8 Report, International Purdue Workshop on Industrial Computer Systems

[2] Bauman R., Lalive d'Epinay T., Schrott G. : Design of Distributed Multiprocessor Operating Systems. Computers in Industry 1979 vol. 1 nr 2

[3] Białasiewicz J., Adamek A., Maliszewski K. : Oprogramowanie podstawowe komputerowych systemów sterowania. Warszawa: WNT 1979

mgr inż. Jerzy MOCAŁA
Instytut Maszyn Matematycznych

System BASIC MERA 400 i WANG 2200. Porównanie

Wstęp

Rozwój produkcji zestawu MERA-400 i wprowadzenie go do biur projektowych spowodował rozpoczęcie prac nad przenoszeniem oprogramowania z innych maszyn. Bogate oprogramowanie maszyny cyfrowej WANG 2200 w BASIC-u oraz łatwość korzystania z programów na tej maszynie przez użytkowników-projektantów spowodowała, że użytkownicy mc MERA-400 chcą uzyskać podobne wygodne narzędzie do prac projektowych. Przed rozpoczęciem prac implementacyjnych programów z mc WANG na mc MERA warto zdać sobie sprawę z różnic tych systemów, aby założenie, że wystarczy zmienić kod programów nie okazało się błędne. Różnice mogą występować nie tylko w formie pewnych instrukcji, ale także w reprezentacji maszynowej liczb i wyrażeń, lub w interpretacji instrukcji.

W artykule przedstawiono podstawowe różnice systemów BASIC dla minikomputerów MERA-400 i WANG 2200. Przyjmując BASIC na WANG za system wzorcowy - przedstawiamy encyklopedyczny opis systemu BASIC dla mc MERA-400.

Ponadto opisując dyrektywy i instrukcje języka przyjęto następujące zasady:

- parametry w nawiasach trójkątnych < > muszą być zastąpione wartościami podanymi przez użytkownika zgodnie z jego intencją wykonania prac przez mc,
- nawiasy kwadratowe [ ] oznaczają, że parametry w nich zawarte są opcjonalne,
- parametry zawarte w klamrach { } umieszczone jeden nad drugim oznaczają, że jeden z nich jest wymagany,
- nawiasy z wielokropkiem oznaczają, że parametr w nich zawarty może powtarzać się wielokrotnie,
- prostokąty oznaczają klawisze zwrotnego urządzenia wejścia/wyjścia np. CR oznacza klawisz "powrót karetki".

Ogólny opis systemu BASIC

System BASIC jest systemem konwersacyjnym umożliwiającym użytkownikowi pracę w trybie natychmiastowym i w trybie programowym.

W trybie natychmiastowym wykonuje się dyrektywy typu rezerwacji obszaru na dysku, wyświetlania wartości zmiennych programu, różnych obliczeń itp. W trybie programowym wykonuje się przygotowany program w języku programowania BASIC.

Te dwa tryby wzajemnie się przeplatają. Należy przy tym zwrócić uwagę, że część dyrektyw ma swoje odpowiedniki w postaci instrukcji języka BASIC.

Program w języku BASIC zbudowany jest z wierszy składających się z numeru wiersza i z instrukcji.

Numer wiersza określa kolejność wykonywania instrukcji. Instrukcje języka BASIC są analogiczne do innych języków algorytmicznych, np. do FORTRAN-u.

Program w języku BASIC wprowadza się do pamięci maszyny wiersz po wierszu, przy czym kolejność wprowadzania wierszy jest dowolna, nie musi się pokrywać z kolejnością wynikającą z porządku ustalonego przez wzrastające numery wierszy. Jeśli wiersz programu jest syntaktycznie poprawny, zostanie zapamiętany, stanowiąc część tworzonego programu.

Poprawność całego programu sprawdza się podczas realizacji dyrektywy systemowej wykonania programu.

Konwersacyjność(użytkownik komunikuje się z systemem przez zwrotne urządzenie wejścia/wyjścia) umożliwia przerwanie wykonania programu, zlecanie wykonania dyrektyw i następnie powracanie do zawieszonego programu i kontynuowanie jego pracy od momentu przerwania. Ułatwia to uruchamianie programu, a także pozwala uzyskać wygodną formę użytkową programu.

Dyrektywy systemu BASIC. Tryb natychmiastowy

Dyrektywy są to zlecenia użytkownika wykonywane w trybie natychmiastowym. System przechodzi do pracy w trybie natychmiastowym, jeśli wprowadzony wiersz informacji nie ma numeru.

Zakres prac możliwych do realizacji za pomocą dyrektyw jest następujący:

- listowanie programu przechowywanego w pamięci operacyjnej,
- oczyszczenie pamięci operacyjnej,
- załadowanie programu do pamięci operacyjnej z pamięci zewnętrznej (dysk, taśma),
- przesłanie programu z pamięci operacyjnej do pamięci zewnętrznej,
- przenumerowanie programu,
- zatrzymanie i kontynuacja programu,
- nadanie wierszowi kolejnego numeru.

Formalny zapis tych dyrektyw dla systemów WANG i MERA przedstawia tab. 1.

Tab. 1. Podstawowe dyrektywy systemu BASIC

Opis ogólny	WANG 2200	MERA 400
Listowanie programu	LIST [S] [⟨nr1⟩ [, ⟨nr2⟩]]	LIST [# ⟨n⟩ [⟨nr1⟩ - - [⟨nr2⟩]]]
Czyszczenie pamięci operacyjnej	CLEAR [{P / V / N}] [⟨nr1⟩ [, ⟨nr2⟩]]	{NEW / SCRATCH}
Załadowanie programu do PAO z pamięci zewnętrznej	LOAD [DC {F / R / T}] [{#n / XXX}] ["nazwa"] [⟨nr1⟩] [, ⟨nr2⟩]]	{LOAD #⟨n⟩ / OLD ⟨z⟩}
Przesłanie programu z PAO do pamięci zewnętrznej	SAVE [DC {F / R / T}] [S] [(wyrażenia) / "nazwa"] [{#n / XXX}] [P] ["nazwa"] [⟨nr1⟩] [, ⟨nr2⟩]]	SAVE
Przenumerowanie programu	RENUMBER [⟨nr1⟩] [, ⟨nr2⟩] [, ⟨LICZBA⟩]	RESEQUENCE [⟨nr1⟩] [, ⟨nr2⟩] [, ⟨liczba⟩]

Opis ogólny	WANG 2200	MERA 400
Zatrzymanie programu	{ HALT STEP / RESET }	CONTROL i T
Wznowienie programu	{ CONTINUE / GOTO <nr> / RUN [<nr>] }	RUN [# nr1]. [,# nr2] [, <nr1> - <nr2>]
Nadanie wierszowi kolejnego numeru	STMT NUMBER	
Czyszczenie dysku	SCRATCH { F / R / T } [{ #n, / /XXX, }] "<nazwa1>" [,"<nazwa2>"] ...	-
Listowanie dysku	LIST DC { F / R / T } [{ #n / /XXX }]	-
Przepisywanie zawartości talerzy dyskowych	MOVE [{ #n, / /XXX, }] { RF / FR }	-
Sprawdzanie poprawności zapisu na dysku	VERIFY { F / R / T } [{ #n, / /XXX, }] [(<wyrażenie1>,<wyrażenie2>)]	-
Rezerwacja obszaru na dysku	DATA SAVE DC OPEN { F / R / T } [$] [#n] { { "<nazwa>" / "<wyrażenie>" }, '<nazwa> / TEMP, '<wyrażenie1>' , '<wyrażenie2>' }	-

#n - numer logiczny, do którego jest przypisany adres urządzenia zewnętrznego,
/XXX - adres w kodzie szesnastkowym urządzenia zewnętrznego
R - talerz wymienny,
F - talerz stały

W podanym zestawieniu podano jedynie podstawowe dyrektywy. Ilustruje ono różnice formalne. Dokładniejszy opis poszczególnych dyrektyw przedstawi różnice w działaniu i sytuacje, w których korzysta się z tych dyrektyw.

- Listowanie programu

Dyrektywa LIST służy do listowania programu znajdującego się w pamięci operacyjnej. Urządzenie, na którym będzie wyświetlany drukowany listing programu, zależy od dyrektywy:

SELECT LIST <nr> [(<długość>)]

gdzie: nr - numer urządzenia,
długość - maksymalna długość wiersza na wydruku.
Dyrektywa ta wybiera odpowiednie urządzenie wyjściowe.

Przykład

SELECT LIST 215 (116) - specyfikuje drukarkę z maksymalną długością wiersza na wydruku 116 znaków,

SELECT LIST ØØ5 (5Ø) - specyfikuje monitor ekranowy z maksymalną długością wyświetlanego wiersza 50 znaków (ekran ma standardową długość 64 znaki).

Parametr S w dyrektywie LIST przeznaczony jest do użycia przy wyświetlaniu programu na monitorze ekranowym. Powoduje on listowanie programu stronami. Strona zawiera tyle wierszy programu, ile mieści się na ekranie. Po każdej stronie wyświetlanie się kończy. W celu kontynuacji listingu trzeba nacisnąć klawisz CR/LF, co spowoduje wyświetlenie następnej strony.

Parametry nr 1, nr 2 oznaczają część programu zawartą między tymi numerami, która ma być listowana.

Dyrektywa MERA-400 działa podobnie. Parametr n określa urządzenie, na którym chcemy otrzymać listing. Brak w niej parametru S, co w sytuacji, gdy system korzysta z monitorów ekranowych jako urządzeń WE/WY powoduje, że zawsze listing programu należy wyprowadzać na drukarkę. W przeciwnym wypadku dostępne będą na ekranie jedynie ostatnie wiersze programu. Może to być utrudnieniem, gdy monitory są zainstalowane w innym pomieszczeniu niż drukarka.

Czyszczenie pamięci operacyjnej

Dyrektywa CLEAR na WANG-u ma różnorodne zastosowanie w zależności od użytych parametrów:

CLEAR - usuwa z pamięci program wraz ze zmiennymi oraz zwalnia wszystkie przypisane urządzenia WE/WY,

CLEAR P - usuwa tylko tekst programu,

CLEAR V - zeruje wszystkie zmienne występujące w programie,

CLEAR N - zeruje zmienne, które nie występują w obszarze COMMON.

Parametry nr 1 i nr 2 określają granice działania dyrektywy.

Podanie numerów wierszy ogranicza działanie dyrektywy do części programu zawartej między tymi wierszami.

Dyrektywy MERA-400 działają jak dyrektywa CLEAR bez parametrów w systemie BASIC WANG:

NEW ≡ SCRATCH ≡ CLEAR

Ładowanie programu do pamięci operacyjnej

Dyrektywa LOAD umożliwia wprowadzenie programu przechowywanego w pamięci zewnętrznej do pamięci operacyjnej. Jeżeli w PAO znajduje się program, to tworzona jest suma programów tego, który był w pamięci i nowo ściąganego.

Przykład

LOAD DC R "CDPRO" 1ØØ,2ØØ - w miejsce od numeru wiersza 100 do 200 zostanie umieszczony program przechowywany na dysku wymiennym pod nazwą CDPRO (por. instrukcja LOAD),

LOAD "PROG1" - z pamięci taśmowej (kasetowej) zostanie ściągnięty program przechowywany pod nazwą PROG1.

W systemie BASIC MERA-400 dyrektywa ładowania jest zróżnicowana w zależności od nośnika, na którym przechowywany jest program.

Dyrektywa ładowania programu z dysku ma postać:

OLD < z >

Oznacza ona ładowanie do wyczyszczonej pamięci operacyjnej programu ze zbioru dyskowego o numerze z.

Dyrektywa

LOAD #<n>

o parametrze n - numer logiczny urządzenia wejściowego (dysk, czytnik) ściąga do pamięci program bez czyszczenia pamięci. Działa więc podobnie jak dyrektywa WANG-owska. Ponieważ podaje się w niej jedynie numer logiczny urządzenia WE potrzeba wcześniej "ustawić" na nim potrzebny program.

Przesłanie programu z pamięci operacyjnej

Przesłanie całego programu lub jego części do pamięci zewnętrznej można zrealizować za pomocą dyrektywy SAVE.

Przykład

SAVE DC F ("STARY") "NOWY" - oznacza przesłanie programu NOWY z pamięci na dysk stały na miejsce programu STARY,

SAVE "PGM" 1ØØ,5ØØ - oznacza przesłanie części programu PGM (od wiersza 1ØØ do 5ØØ) na taśmę magnetyczną.

W systemie BASIC MERA-400 SAVE jest dyrektywą bezparametryczną. Jej zastosowanie ogranicza się jedynie do możliwości przepisania programu z pamięci na zbiór dyskowy. Program dopisywany jest na końcu pliku zbiorów. Aby móc się do niego odwołać należy zapamiętać miejsce zapisu (zob. dyrektywę OLD).

Przenumerowanie programu

System BASIC umożliwia aktualizację programu np. przez wstawienie nowych wierszy. Może się zdarzyć, że sąsiednie wiersze mają numery nie pozwalające wstawić dodatkowego ciągu wierszy. Niezbędne jest przenumerowanie programu nadanie nowych numerów wierszom oraz aktualizację skoków itp. . Do tego celu służy dyrektywa RENUMBER.

Parametr nr 1 oznacza numer wiersza, od którego ma być prowadzona renumeracja, nr 2 oznacza numer, na jaki ma być zamieniony numer nr 1, liczba określa krok numeracji.

Dyrektywa RESEQUENCE spełnia podobne funkcje. Różnice występują przy pomijaniu parametrów dyrektywy.

Dla WANG-a dyrektywa postaci:
RENUMBER - spowoduje nadanie całemu programowi numerów co 10 zaczynając od 10.

Natomiast dla MERA-400 dyrektywa:
RESEQUENCE - spowoduje nadanie całemu programowi numerów co 10 zaczynając od 100.

Zatrzymanie programu

W systemie BASIC istnieje możliwość zatrzymania wykonywanego programu przez naciśnięcie klawisza [HALT/STEP] lub [RESET].

Klawisz [HALT/STEP] przeważnie używany jest do wykonywania programu krok po kroku. Odnosi się on do jednej instrukcji programu. Jeśli w jednym wierszu programu było kilka instrukcji, to program zatrzymuje się po wykonaniu przetwarzanej instrukcji. Dyrektywa [RESET] jest używana, gdy w wypadku zakłócenia przerwanie wykonania programu przez [HALT/STEP] nie skutkuje.

Dyrektywa ta nie pozwala na kontynuację wykonywania programu od miejsca, w którym nastąpiło przerwanie, chociaż tekst programu z aktualnie obliczonymi wartościami zmiennych przechowywany jest w pamięci operacyjnej.

W systemie BASIC MERA zatrzymanie działania programu jest możliwe przez wciśnięcie klawiszy [CONTROL] i [T]. Ponieważ nie można wznowić programu od miejsca przerwania, dyrektywa ta jest odpowiednikiem dyrektywy [RESERT].

Wznowienie działania programu

Rozpoczęcie wykonywania programu dokonuje się za pomocą dyrektywy RUN. Jeśli nie użyto parametru, program wystartuje od wiersza o najniższym numerze. Przed przekazaniem sterowania programowi, zerowane są zmienne spoza obszarów COMMON (por. Instrukcje deklaracji). Dyrektywa tej postaci służy do rozpoczęcia wykonania programu załadowanego do pamięci. Gdy przerwano działanie programu (np. dyrektywą HALT/STEP), a następnie program ma kontynuować obliczenia od miejsca przerwania, należy użyć dyrektyw:

CONTINUE , GOTO < nr >, RUN < nr > .

Programu nie można kontynuować, gdy po przerwaniu dołączono nowy wiersz, przenumerowano program, wykonano dyrektywę CLEAR. Dyrektywa CONTINUE powoduje wznowienie wykonywanego programu od następnej instrukcji, przy której nastąpiło przerwanie.

Dyrektywy GOTO < nr >, RUN < nr > wznawiają wykonanie od numeru wiersza podanego jako parametr dyrektywy. Nie może to być numer wiersza występującego w pętli FORT-NEXT lub podprogramie GOSUB-RETURN.

System BASIC MERA nie ma tak bogatego aparatu wznawiającego działanie programu.

Dyrektywa RUN zeruje zawsze zmienne proste, więc jedynie może służyć do wystartowania całego lub części programu.

Znaczenie parametrów dyrektywy jest następujące:

n1 - numer logiczny wejścia,
n2 - numer logiczny wyjścia,
nr 1 - nr 2 - numery wierszy, które ograniczają wykonanie programu od nr 1 do nr 2.

Kolejny numer wiersza

Numery wierszy mogą być wprowadzane bądź z klawiatury numerycznej, bądź za pomocą klawisza STMT NUMBER .

Naciśnięcie klawisza powoduje nadanie wierszowi, który ma być wprowadzany, numeru większego o 10 od numeru ostatnio wprowadzonego wiersza.

W systemie BASIC MERA numery wierszy wprowadza się tylko z klawiatury numerycznej.

Operacje dyskowe

Ze względu na to, że w obecnej postaci system BASIC MERA-400 nie ma możliwości korzystania z dysku, nie wydaje się celowe omawianie szerzej dyrektyw związanych z pamięcią dyskową.

W systemie BASIC oprócz dyrektyw, można w trybie natychmiastowym wykonywać pewne prace (nazywane dalej zleceniami) typu: wydrukować wartość zmiennych, nadać nowe wartości zmiennym, obliczyć proste wyrażenie arytmetyczne, itp.

W systemie MERA-400 jest możliwe do wykonania jedynie zlecenie drukowania wartości zmiennych lub wyrażenia arytmetycznego, jeśli nie będą to zmienne indeksowane lub funkcje definiowalne.

Postać zlecenia jest następująca:

PRINT { < wyrażenie arytmetyczne > / < zmienna prosta > }

Przykład

PRINT A*B - na urządzeniu zwrotnym WE/WY wydrukowana zostanie wartość iloczynu, jeśli poprzednio zmienne A i B występowały w programie,

PRINT A (3) - zlecenie niepoprawne, zawiera element tablicy.

Dla porównania zlecenie tej klasy na WANG-u może mieć postać:
<instrukcja> : <instrukcja> : ..... : PRINT <wyrażenie arytmetyczne>

Pozwala to wykonywać proste programy mieszczące się w jednym wierszu. Praca na maszynie cyfrowej podobna jest wtedy do obliczeń wykonywanych na kalkulatorze elektronicznym.

Opis języka BASIC. Tryb programowy

W systemie BASIC w trybie programowym wykonuje się programy napisane w języku BASIC. Elementami składowymi programu w języku BASIC są instrukcje, stałe, zmienne proste i tablice, wyrażenia arytmetyczne, numery wierszy, funkcje standardowe, operatory arytmetyczne i logiczne oraz spacje.

Poszczególne te elementy zostaną omówione poniżej.

Podstawowe symbole języka

Litery: od A do Z (26 znaków), w systemie MERA do liter dołączono znak @ (27 znaków).
Cyfry: od Ø do 9.
Operatory arytmetyczne: + - * / ↑
Operatory relacji: = > < => =< < >
Ograniczniki: , ; : " ' (w systemie MERA nie ma ogranicznika ')
Kropka dziesiętna: .
Nawiasy: ( )
Znak zapytania: ?
Znak numeru: #
Znaki specjalne: $ % (w systemie MERA znaki te można używać w komentarzach i łańcuchach alfanumerycznych).

Typy stałych

W programach w języku BASIC można używać stałych liczbowych i alfanumerycznych. Stałe liczbowe mogą być przedstawione w postaci liczb całkowitych, ułamków dziesiętnych lub w postaci z wykładnikiem, ale zawsze są traktowane przez maszynę jako liczby zmiennoprzecinkowe. Stała liczbowa może mieć co najwyżej 13 cyfr (w systemie MERA - 11 cyfr). Dokładność i zakres liczb reprezentowanych przez maszynę cyfrową WANG wynosi:

1E-99 < <liczba> < 1E+99

W systemie MERA istnieją dwie funkcje standardowe, które informują o dokładności i zakresie liczb.

Funkcja EPS jest najmniejszą liczbą, która spełnia w arytmetyce maszyny nierówność:

1+EPS > 1 i 1-EPS < 1
EPS=1.81899 E-12

Funkcja INF jest największą liczbą, dla której +INT oraz -INF da się przedstawić w maszynie cyfrowej:

INF=1.70141 E+38

Dla wygody programistów w BASIC-u wprowadzono standardowo stałą liczbową oznaczoną # PI, której wartość, to liczba π - na mo MERA stałą tę oznacza się następująco: @ PI.

Oprócz stałych liczbowych można używać stałych alfanumerycznych - są to łańcuchy znaków alfanumerycznych ograniczonych cudzysłowem np.: "STAŁE ALFANUMERYCZNE".

Zmienne

Zmienna jest nazwą, mającą w systemie wartość, która w trakcie wykonywania programu może się zmieniać. Ze względu na wartość rozróżnia się zmienne liczbowe i zmienne alfanumeryczne. Wartościami zmiennych liczbowych są liczby, wartościami zmiennych alfanumerycznych są łańcuchy znaków alfanumerycznych.

Ze względu na strukturę rozróżnia się zmienne proste i tablice. Możliwe rodzaje zmiennych w systemie BASIC przedstawiają tab. 2 i 3 (+ oznacza występowanie zmiennej rodzaju określonego przez podział ze względu na wartość i strukturę).

Tab. 2 Zmienne w systemie WANG

Typ zmiennej	liczbowa	alfanumeryczna
prosta	+	+
tablica	+	+

Tab. 3 Zmienne w systemie MERA

Typ zmiennej	liczbowa	alfanumeryczna
prosta	+	-
tablica	+	-

Z przedstawionego zestawienia wynika, że system BASIC MERA-400 nie ma zmiennych o wartościach alfanumerycznych.

Identyfikator (nazwa) zmiennej prostej liczbowej ma postać:

< litera > [< cyfra >]

Program może zatem rozróżnić 286 różnych zmiennych w systemie WANG i 297 zmiennych w systemie MERA.

Identyfikator tablicy liczbowej ma postać:

< litera > [< cyfra >] - w systemie WANG

< litera > - w systemie MERA

Zatem różnych nazw tablic może być 286 (WANG) i 27 (MERA).

W tab. 4 przedstawiono możliwe identyfikatory zmiennych.

Tab. 4. Identyfikatory zmiennych w języku BASIC

Rodzaj zmiennej	WANG	MERA
zmienna prosta liczbowa	< litera > [< cyfra >]	< litera > [< cyfra >]
zmienna prosta alfanumeryczna	litera [< cyfra >] $	-
tablica liczbowa	litera [< cyfra >]	< litera >
tablica alfanumeryczna	litera [< cyfra >] $	-

Tablice mogą być co najwyżej dwuwymiarowe. W systemie WANG indeksy tablic mogą mieć wartość od 1 do 255, natomiast na mc MERA od 0 do 32767.

Wyrażenia arytmetyczne

Wyrażenie arytmetyczne zbudowane jest ze zmiennych, stałych, operatorów arytmetycznych, funkcji standardowych, funkcji użytkownika oraz nawiasów ustalających porządek wykonywania działań.

<wyrażenia arytmetyczne> ::= <zmienna liczbowa>|<stała liczbowa>|<funkcja>|<kombinacja tych elementów połączonych znakami działań arytmetycznych>

<zmienna liczbowa> ::= { <zmienna liczbowa prosta> | <tablica liczbowa> }

Kolejność wykonywania działań (gdy nie ma nawiasów) jest następująca:

1) potęgowanie
2) mnożenie i dzielenie
3) dodawanie i odejmowanie

Dla operatorów z tej samej grupy priorytetowej kolejność wykonania ustala pierwszeństwo występowania w wyrażeniu "czytając" go od lewej do prawej.

Przykład

Wyrażenie postaci: A * B/C * D
jest równoważne: ((A * B) /C) * D

Oznaczenie operatorów arytmetycznych przedstawiono w tab. 5.

Tab. 5. Operatory arytmetyczne

Znaczenie	Symbol operatora WANG	Symbol operatora MERA
dodawanie	+	+
odejmowanie	-	-
mnożenie	*	*
dzielenie	/	/
potęgowanie	↑	^

Instrukcja podstawienia

Instrukcja podstawienia LET pozwala zmienić wartości poszczególnym zmiennym programu. Postać ogólna tej instrukcji w systemie WANG jest następująca:

[LET] <zmienna> [{ ,<zmienna> } ...] = { <wyrażenie arytmetyczne> | <zmienna> | <stała> }

Przykład

100 LET A,B,C=100*A/D

Instrukcja ta oznacza, że zmiennym prostym A,B,C nadano wartość wyrażenia 100*A/D.

W systemie MERA instrukcja podstawienia odnosi się tylko do zmiennych liczbowych. Postać ogólna w tym systemie jest następująca:

$$[\text{LET}]\ \langle\text{zmienna liczbowa}\rangle\ [\{=\langle\text{zmienna liczbowa}\rangle\}\ \ldots]\ =\ \left\{\begin{array}{l}\langle\text{wyrażenie arytmetyczne}\rangle\\ \langle\text{zmienna liczbowa}\rangle\\ \langle\text{stała liczbowa}\rangle\end{array}\right\}$$

Przykład

Instrukcja z przykładu poprzedniego w systemie MERA ma postać:

100 LET A=B=C=100*A/D.

Słowo LET może być pominięte dla oszczędności miejsca w pamięci i z reguły podczas pisania programu bywa pomijane.

Instrukcje skoku

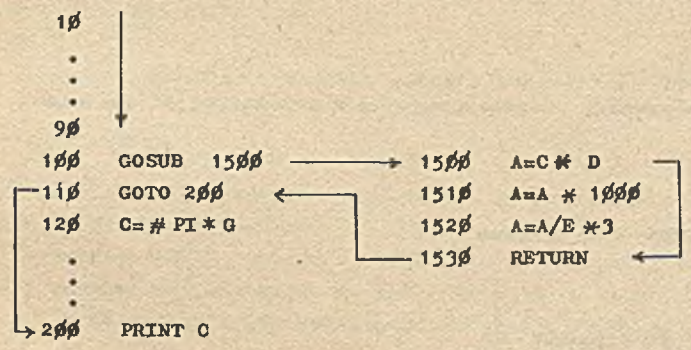
Instrukcje skoku pozwalają zmieniać kolejność wykonywania wierszy programu w czasie jego realizacji. Do tej klasy instrukcji należą:

- instrukcja skoku GOTO <nr wiersza>
- instrukcja skoku ze śladem GOSUB <nr wiersza>
- instrukcja skoku liczonego ON <wyrażenie arytmetyczne> $\left\{\begin{array}{l}\text{GOSUB}\\ \text{GOTO}\end{array}\right\}$ <nr1> $[\{,\langle nr2\rangle\}\ \ldots]$

Instrukcja skoku służy do zmiany normalnej kolejności wykonywania programu. Określa, że następnym wykonywanym wierszem ma być wiersz podany w instrukcji skoku.

Instrukcja skoku ze śladem umożliwia wykonanie skoku do wiersza o podanym numerze, wykonanie instrukcji począwszy od tego wiersza, aż do napotkania instrukcji RETURN i powrót do wykonywania następnej instrukcji po GOSUB. Sytuację taką obrazuje poniższy przykład.

Przykład



Program wykonuje się w formie realizacji wierszy kolejno od 10 do 90, instrukcja GOSUB zmienia kolejność wykonywania. Następuje skok do wiersza 1500, teraz wykonują się wiersza od 1500 do 1530, aż do instrukcji RETURN, która powoduje powrót do wykonywania instrukcji wiersza o numerze 110. Ponieważ w tym wierszu jest instrukcja skoku, następnym realizowanym wierszem jest wiersz o numerze 200.

Instrukcja skoku liczonego jest odmienną instrukcji GOSUB lub GOTO. Podczas wykonywania instrukcji oblicza się wartość wyrażenia arytmetycznego. Część całkowita wskazuje na liście numerów wierszy pozycję, do której przekazane zostanie sterowanie.

Przykład

```
10   A=2
20   B=1.2
30   ON A*B   GOTO   200,300,400
```

Wartość wyrażenia A * B=2.4. Część całkowita wynosi 2. Na liście wierszy druga pozycja, to numer wiersza 300. Sterowanie zostaje przekazane do wiersza o numerze 300.

Jeśli część całkowita wyrażenia jest większa lub mniejsza od liczby wyspecyfikowanych numerów wierszy w instrukcji ON, to wykonywana jest następna instrukcja po ON. W systemie MERA sytuacja taka traktowana jest jako błąd i przerywane jest wykonywanie programu z wydrukiem komunikatu o błędzie.

Instrukcja pętli

Instrukcja o postaci:

FOR <liczbowa zmienna prosta> = <wyrażenie1> TO <wyrażenie2> [STEP <wyrażenie 3>]

wspólnie z instrukcją:

NEXT <liczbowa zmienna prosta>

służy do cyklicznego wykonywania instrukcji zawartych między tymi instrukcjami.

Instrukcje będą się wykonywały tyle razy, aż <liczbowa zmienna prosta>, nazywana zmienną sterującą, nie przekroczy wartości określonej przez <wyrażenie2>. Za każdym przebiegiem pętli zmienna sterująca zwiększa swoją wartość o wartość kroku ustaloną przez <wyrażenie3>. Gdy opcja STEP <wyrażenie3> jest pominięta, przyjmuje się wartość kroku za +1.

W systemie WANG pętla wykona się przynajmniej jeden raz, nawet w sytuacji poniższej:

```
FOR I=1 TO-1
  :
NEXT I
```

Wartość zmiennej sterującej przekroczyła ograniczenie, ale pętla wykona się jeden raz zanim zostanie sprawdzony warunek przerywający wykonanie pętli. W systemie MERA w takiej sytuacji pętla się nie wykona, będzie pominięta. Wykona się wiersz następny za NEXT I.

Instrukcja warunkowa

Instrukcja warunkowa zmienia kolejność wykonywania wierszy programu w zależności od spełnienia relacji.

Postać ogólna instrukcji jest następująca:

IF <operand> <operator relacji> <operand> THEN <nr wiersza>

$$\text{<operand>} ::= \left\{ \begin{array}{l} \text{<zmienna>} \\ \text{<stała>} \\ \text{<wyrażenie arytmetyczne>} \end{array} \right\}$$

Jeśli relacja w instrukcji jest spełniona, wykonywany jest wiersz o numerze podanym w instrukcji, jeśli nie, wykonywany jest wiersz następny za instrukcją warunkową.

W systemie MERA instrukcja warunkowa ma postać:

$$\text{IF <operand> <operator relacji> <operand>} \left\{ \begin{array}{l} \text{THEN} \\ \text{GOTO} \\ \text{GOSUB} \end{array} \right\} \text{<nr wiersza>}$$

$$\text{<operand>} ::= \left\{ \begin{array}{l} \text{<zmienna liczbowa>} \\ \text{<stała liczbowa>} \\ \text{<wyrażenie arytmetyczne>} \end{array} \right\}$$

Jest rozszerzona o możliwość wykonania skoku za śladem (GOSUB). Postać IF - THEN jest równoważna postaci IF - GOTO.

Instrukcja wejścia

Wprowadzanie danych w czasie działania programu realizuje instrukcja INPUT. Postać ogólna instrukcji:

INPUT [<stała alfanumeryczna>,] < zmienna1 > [,{ < zmienna2 >} ...]

Podczas wykonywania instrukcji na końcówce wyświetlona zostanie <stała alfanumeryczna> i znak ?. Następnie program czeka na podanie wartości dla zmiennych występujących na liście zmiennych.

Przykład

INPUT "WPROWADZ DANE", A
Na ekranie wyświetli się napis:
WPROWADZ DANE ?

Gdy napiszemy 12.5 [CR], na zmienną A wczytana zostanie wartość 12.5.

W systemie MERA-400 instrukcja wejścia ma postać:

INPUT < zmienna liczbowa > [,{ < zmienna liczbowa >} ...]

Nie ma w niej parametru <stała alfanumeryczna> , zatem gdy chcemy zaznaczyć, że program czeka na wprowadzenie danych, należy użyć sekwencji instrukcji:

{ PRINT
{ PRINTUSING }
INPUT .....

Przykład

PRINT "WPROWADZ DANE"
INPUT A
Te dwie instrukcje są równoważne jednej instrukcji z przykładu poprzedniego.

Instrukcje wyjścia

Wyniki obliczeń programu wyprowadza się na urządzenie wyjściowe (monitor, drukarka) instrukcjami PRINT lub PRINTUSING.

Instrukcja PRINT realizuje drukowanie według przyjętego w systemie formatu. Gdy proponowane przez system rozmieszczanie wyników obliczeń na wydruku jest niezadowalające, można określić swój format wydruków korzystając z instrukcji PRINTUSING.

Postać ogólna instrukcji PRINT jest następująca:

PRINT < element>[{{ ; } < element>}....] [;]
                 { , }   [,]

<element> ::= { < stała >
              { < zmienna >
              { < wyrażenie arytmetyczne >
              { TAB (< wyrażenie>)

Użycie separatora , między elementami powoduje rozpoczęcie drukowania elementu od początku 15-znakowej strefy, na jakie dzieli się pole przeznaczone do wydruku.

Separator ; likwiduje podział wydruku na strefy. Można korzystać wtedy z funkcji TAB, która ustala początek wydruku elementu na pozycji wyznaczonej przez argument funkcji.

Wartości liczbowe drukowane są w dwóch formatach:

- dla wartości < 0.1 lub $> 10^{13}$

  $\pm$ M,MMMMMMME$\pm$XX

- dla wartości z przedziału $[0.1, 10^{13}]$

  $\pm$MMMMMM.MMMMMMM

  Liczba może zajmować maksymalnie 14 znaków.

W systemie BASIC MERA-400 instrukcja PRINT działa podobnie. Jedyna różnica polega na tym, że drukowane liczby zawierają mniej cyfr (maksymalnie sześć cyfr). Postać drukowanej liczby jest następująca:

$\pm$M.MMMMME$\pm$XX lub $\pm$MMM.MMM

Postać ogólna instrukcji PRINTUSING jest następująca:

PRINTUSING < nr wiersza > [, { < element > [;] } ...]

Instrukcja drukowana używa instrukcji formatującej znajdującej się w < nr wiersza > . Instrukcja formatująca (wzorca) ma postać:

% [< łańcuch alfanumeryczny >] [{ [{ + / - / $ }] [# [,] ...] [. [# ...]] [↑↑↑↑]

< łańcuch alfanumeryczny > } ....]

Znak # oznacza pozycję cyfry.

Instrukcja wzorca może zawierać teksty umieszczone przed, między i po specyfikacjach formatu. Przykłady formatów dla liczb są następujące:

- liczby całkowite # # # (drukowanie liczb co najwyżej trzycyfrowych)
- liczby rzeczywiste ## . ### (drukowanie liczb o dwóch cyfrach przed kropką dziesiętną i trzech po kropce)
- liczby rzeczywiste w postaci wykładniczej # . # # ## ↑↑↑↑ (strzałki oznaczają pole przeznaczone na wydruk E$\pm$XX)

Instrukcja drukowania (PRINTUSING i wzorzec) systemu MERA-400 różni się następującymi szczegółami:

- po < nr wiersza > w PRINTUSING występuje separator ;
- instrukcja wzorca rozpoczyna się znakiem ; , a nie %,
- pole wykładnika zajmuje 5 znaków
- przy drukowaniu następuje zaokrąglanie liczby

Przykład

Niech zmienna A ma wartość 12.7886.

Drukując instrukcję PRINTUSING ... ,A, dla której we wzorcu wyspecyfikowano format # #. ### otrzyma się wynik:

- w systemie MERA 12.789,
- w systemie WANG 12.788.

W systemie BASIC WANG należy używać instrukcji:

PRINTUSING ... , A+Ø.ØØØ5

Instrukcja komentarza

Instrukcja komentarza jest używana w celu zwiększenia czytelności programu. Zawiera dowolny tekst objaśniający działanie programu, wskazówki dla konserwatora programu itp. Pisząc komentarze, należy pamiętać, że instrukcja ta, obocnie jest ignorowana podczas wykonywania programu zajmuje miejsce w pamięci. Dla właściwego programu zostaje zatem mniej miejsca w PAO.

Postać ogólna instrukcji komentarza jest następująca:

REM < łańcuch alfanumeryczny >

Przykład

```
1Ø  REM PROGPAM KATLG
2Ø  REM ZAKLADA KATALOG ELEMENTOW WSPORCZYCH
```

Instrukcje deklaracji

Instrukcje należące do tej klasy są instrukcjami niewykonywalnymi, tzn. z punktu widzenia programisty nie powodują żadnej akcji. Odgrywają one zasadniczą rolę z punktu widzenia systemu, dostarczając mu opisu obiektów, na których będą wykonywane operacje.

Na podstawie opisu system rezerwuje odpowiednie obszary w pamięci operacyjnej i przy odwoływaniu do tych obszarów odpowiednio interpretuje jego zawartość.

Postać ogólna instrukcji jest następująca:

DIM < lista zmiennych >
COM < lista zmiennych >

Instrukcja DIM służy do rezerwowania obszarów w PAO dla tablic jedno- i dwuwymiarowych, a także dla zmiennych alfanumerycznych.

Przykład

```
1ØØ   DIM N1$16, D(36,12)
```

Instrukcja powyższa spowoduje zarezerwowanie odpowiedniej liczby bajtów na zmienną alfanumeryczną N1$ o długości 16 znaków i tablicę liczbową o 432 elementach - zarezerwuje 3872 bajtów.

Instrukcja COM umożliwia przenoszenie wartości danych między segmentami dużego programu. Każdy segment musi zaczynać się od takiej samej instrukcji COM, która listą zmiennych określa obszar wspólny (COMMON) dla segmentów. Zmienne zadeklarowane w COM nie będą wymazywane z PAO przy ładowaniu do niej nowego segmentu programu.

Przykład

```
1Ø  COM  T$(5)44,A1(26),A1
```

Instrukcja deklaruje tablicę alfanumeryczną T (5 elementów po 44 znaki każdy), tablicę liczbową (wektor) A1 o 26 elementach i liczbową zmienną prostą A1. Zmienne te nie będą wymazywane przy ściąganiu segmentu programu do pamięci operacyjnej.

W systemie BASIC MERA-400 dopuszcza się używanie jedynie instrukcji DIM. Instrukcja COM jest niedozwolona.

Instrukcja zatrzymania

Postać ogólna instrukcji zatrzymania realizacji programu jest następująca:

STOP ["<łańcuch alfanumeryczny>"]

Instrukcja powoduje zatrzymanie programu - możliwe jest wtedy wykonanie pewnych czynności manualnych, np. zmiana kasety, włączenie drukarki lub wykonanie dyrektyw systemowych, np. rezerwacja obszaru na dysku. Aby wznowić program od następnej instrukcji po STOP, używa się dyrektywy CONTINUE.

Zatrzymanie się programu sygnalizowane jest wyświetleniem słowa STOP i łańcucha alfanumerycznego. W programie może występować kilka instrukcji zatrzymania, parametr instrukcji znakowicie ułatwia rozpoznanie, w którym miejscu wstrzymano działanie programu.

Przykład

```
35Ø  STOP "POTRZEBNA KASETA PB4"
```

Na ekranie zostanie wyświetlony napis: STOP POTRZEBNA KASETA PB4 i program zawiesi swoją działalność.

W systemie MERA-400 instrukcja STOP służy jedynie do zakończenia wykonywania programu. Jest instrukcją bez parametru o postaci: STOP

Przykład

```
2ØØ   STOP
```

Działanie programu jest zakończone, na urządzeniu zwrotnym WE/WY system wydrukuje informację w postaci:

```
FINISH 2ØØ
```

Instrukcja końca programu

Instrukcja końca programu zatrzymuje odpowiednią ścieżkę realizacji programu. Zatrzymanego tą instrukcją programu nie można wznowić dyrektywą CONTINUE.

Postać ogólna dyrektywy jest następująca:

END

Przykład

```
1Ø    REM PGM TP1'
 .
 .
 .
8Ø    IF A=Ø THEN 1ØØ
9Ø    END
 .
 .
 .
1ØØ/  LET A=B
...
2ØØØ  END
```

Program zakończy się na wierszu o numerze 9Ø lub o numerze 2ØØØ w zależności od wyników relacji A=Ø (wiersz 8Ø).

W systemie BASIC MERA-400 instrukcja END jest logicznym końcem programu. Wiersze o numerach wyższych od wiersza z instrukcją END są przez translator pomijane.

Przykład

```
1Ø   REM PGM1
 .
 .
9Ø   END
1ØØ  REM PGM2
 .
 .
19Ø  END
```

Mając powyższy program w PAO, wywołując go od wiersza 1ØØ system zachowa się następująco:

```
RUN 1ØØ          (wywołanie programu)
RUNNING XXX      (XXX - zajętość pamięci)
FINISH 8Ø        (zatrzymanie na wierszu 8Ø zamiast na 18Ø)
```

Instrukcja END działa tak samo, jak instrukcja STOP.

Instrukcje współpracy z pamięciami zewnętrznymi

W systemie BASIC MERA nie przewidziano możliwości korzystania z pamięci zewnętrznych w takiej postaci, jak na maszynie cyfrowej WANG 2200. Będzie więc jedynie przedstawiona instrukcja umożli-

wiająca segmentowanie programu w systemie BASIC WANG w celu ilustracji rozwiązania problemu wykonywania dużych programów.

Postać ogólna instrukcji jest następująca:

$$\text{LOAD}\left[\ \text{DC}\ \left\{\begin{matrix}\text{R}\\ \text{F}\end{matrix}\right\}\right]\left[\ \left\{\begin{matrix}\#\ \text{n}\\ /\text{XXX}\end{matrix}\right\}\right]\left[\ \text{"<nazwa>"}\right]\left[\ \text{<nr1>}\right]\left[\ \text{,<nr2>}\right]$$

Znaczenie parametrów instrukcji:

DC $\left\{\begin{matrix}\text{R}\\ \text{F}\end{matrix}\right\}$ - ładowanie programu z dysku (R - talerz wymienny, F - stały),
n - numer logiczny zbioru, urządzenia z którego ładowany będzie program,
/XXX - adres urządzenia, z którego ładowany będzie program,
nazwa - nazwa programu, pod którą przechowywany jest program,
nr 1, nr 2 - numery wierszy określające część programu rezydującego w pamięci operacyjnej, która ma być usunięta a na jej miejsce załadowany nowy segment programu.

Instrukcja LOAD automatycznie zrealizuje sekwencję czynności:

- STOP - zatrzymanie programu z możliwością wznowienia,
- CLEAR P<nr1>,<nr2> - usunięcie tekstu programu od nr 1 do nr 2,
- CLEAR N - wyzerowanie zmiennych spoza obszarów COMMON,
- LOAD "<nazwa>" - załadowanie nowego programu,
- RUN <nr1> - wznowienie programu od wiersza nr 1.

Przykład

```
  1Ø   REM PRZYKLAD SEGMENTACJI PROGRAMU
  2Ø   REM JEDNO Z WIELU ROZWIAZAN
   :
 17Ø   PRINT "RODZAJ PRACY:","1-MODYFIKACJA","3-KOPIA","5-OPIS"
 175   PRINT,"2-WYMAZANIE","4-DOLACZENIE","6-KONIEC
 18Ø   INPUT "NUMER",Z
 185   ON Z GOTO 21Ø,22Ø,23Ø,24Ø,25Ø,26Ø
 2ØØ   PRINT "ZLY NUMER": GOTO 17Ø
 21Ø   LOAD DC R "MODYF" 15ØØ,2ØØØ
   :
 25Ø   LOAD DC R "INFO" 15ØØ,2ØØØ
 26Ø   PRINT"!KONIEC PROGRAMU"
   :
 3ØØ   END
 31Ø   REM PROCEDURY I FUNKCJE
   :
149Ø   REM KONIEC PROC I FUNK
15ØØ   REM OBSZAR WYMIANY SEGMENTOW PROGRAMU
2ØØØ   REM KONIEC OBSZARU WYMIANY
2Ø1Ø   GOTO 17Ø
2Ø2Ø   REM KONIEC PRZYKLADU
```

W zależności od wybranego rodzaju pracy z dysku ściągany jest odpowiedni program. Po wykonaniu go przechodzi się do wiersza o numerze 17Ø. Każdy program przechowywany na dysku w tym wypadku musi mieć numerację z przedziału [15ØØ, 2ØØØ].

Instrukcje definicji

Instrukcja należące do tej klasy pozwalają zaprogramować (jednorazowo, a wykorzystywać dowolną liczbę razy) pewnych [illegible]zalnych w programie operacji.

Znaczenie	WANG 2200	MERA-400
entier(x)	INT(X)	INT(X)
signum(x)	SGN(X)	SGN(X)
ln x	LOG(X)	LOG(X)
e^x	EXP(X)	EXP(X)
$\sqrt{x}$	SQR(X)	SQR(X)
	# PI	@ PI
1+EPS > 1 1-EPS < 1	-	EPS
największa liczba w mo	-	INF
część ułamka x	-	FRA(X)
N parzyste Ø N nieparzyste 1	-	ODD(N)

W zasadzie nie ma znacznych różnic w zbiorach funkcji standardowych omawianych systemów.

W systemie BASIC MERA funkcje trygonometryczne sin, cos, tg, ctg wymagają argumentów w radianach. W systemie BASIC WANG jednostki argumentów dla tych funkcji ustala się instrukcją:

SELECT D - argumenty w stopniach,
SELECT R - argumenty w radianach,
SELECT G - argumenty w gradach.

Przykład

```
 1Ø  REM FUNKCJE TRYGONOMETRYCZNE
  :
 5Ø  SELECT R
 6Ø  LET B=A1/#PI
  :
1ØØ  A=SIN B
  :
2ØØ  SELECT G
  :
22Ø  G=SIN E
```

Od numeru wiersza 5Ø do 2ØØ odwołania do funkcji trygonometrycznych (np. wiersz 1ØØ) wymagają, aby argument funkcji był przeliczony na radiany. Od wiersza 2ØØ argumenty funkcji trygonometrycznych muszą być wyrażane w gradach, aby wynik obliczeń był poprawny.

Następną zasadniczą różnicą w funkcjach standardowych omawianych systemów występuje w generatorze liczb losowych.

Funkcja RND generuje liczby losowe z przedziału [0,1]. W systemie WANG jest funkcją z argumentem RND(X). Argument X ma następujące znaczenie:

X = Ø - wydawana jest pierwsza liczba losowa z "pseudolisty",
X ≠ Ø - wydawana jest kolejna liczba losowa z "pseudolisty".

Różnica ta powoduje, że w systemie MERA otrzymuje się zawsze taki sam ciąg liczb losowych, natomiast w systemie WANG można otrzymać taki sam lub różny ciąg liczb losowych (w zależności od potrzeb) za każdym przebiegiem programu.

Rozróżnia się dwie instrukcje:

- instrukcja definiowania funkcji,
- instrukcja definiowania podprogramu i klucza funkcyjnego.

Instrukcja definiowania funkcji ma postać:

DEFFN <identyfikator>(<argument>)=<wyrażenie>

<identyfikator> ::= { <litera> / <cyfra> }

<argument> ::= <liczbowa zmienna prosta>

W systemie MERA identyfikatorem może być tylko <litera>.

Aby skorzystać z funkcji, należy odwołać się do niej z różnych miejsc programu z aktualnym argumentem, np.:

A=FN <identyfikator>(<wyrażenie>)

Instrukcja definiowania podprogramu ma postać:

DEFFN'<identyfikator> { <stała alfanumeryczna> / (<zmienna>[,<zmienna> ...]) }

<identyfikator> ::= liczba całkowita ØØ - 255

Po tej instrukcji występują wiersze zawierające instrukcje realizujące operację, która ma być wielokrotnie używana w programie.

Ostatnią instrukcją podprogramu musi być instrukcja RETURN.

Podprogram może być wywołany albo przez naciśnięcie klawisza funkcyjnego o numerze odpowiadającym identyfikatorowi podprogramu, albo w programie instrukcją:

GOSUB '<identyfikator> (<parametry aktualne>)

Parametry aktualne muszą odpowiadać typom i liczbą parametrów formalnych.

W systemie MERA-400 nie ma możliwości definiowania wyżej wymienionego typu podprogramu.

Funkcje standardowe

Dla ułatwienia pisania programów w języku BASIC dostępne są często używane funkcje (np. trygonometryczne) w postaci standardowej opracowanej przez producenta. Zestaw funkcji standardowych przedstawiono w tab. 6. Podano w niej znaczenie i nazwy funkcji dla obu systemów: BASIC WANG i BASIC MERA.

Tab. 6. Typowe funkcje-standardowe w systemie BASIC

Znaczenie	WANG 2200	MERA-400
sin(x)	SIN(X)	SIN(X)
cos(x)	COS(X)	COS(X)
tg(X)	TAN(X)	TAN(X)
ctg(X)	-	COT(X)
arc sin(x)	ARC SIN(X)	-
arc cos(x)	ARC COS(X)	-
arc tg(x)	ARCTAN(X), ATN(X)	ATN(X)
generator liczb losowych	RND(X)	RND
|X|	ABS(X)	ABS(X)

Znaczenie	WANG 2200	MERA-400
entier(x)	INT(X)	INT(X)
signum(x)	SGN(X)	SGN(X)
ln x	LOG(X)	LOG(X)
e^x	EXP(X)	EXP(X)
$\sqrt{x}$	SQR(X)	SQR(X)
	# PI	@ PI
1+EPS > 1 1-EPS < 1	-	EPS
największa liczba w mo	-	INF
ozpśó ułamka x	-	FRA(X)
N parzyste ø N nieparzyste 1	-	ODD(N)

W zasadzie nie ma znacznych różnic w zbiorach funkcji standardowych omawianych systemów.

W systemie BASIC MERA funkcje trygonometryczne sin, cos, tg, ctg wymagają argumentów w radianach. W systemie BASIC WANG jednostki argumentów dla tych funkcji ustala się instrukcją:

SELECT D - argumenty w stopniach,
SELECT R - argumenty w radianach,
SELECT G - argumenty w gradach.

Przykład

```
 1ø  REM FUNKCJE TRYGONOMETRYCZNE
 :
 5ø  SELECT R
 6ø  LET B=A1/#PI
 :
1øø  A=SIN B
 :
2øø  SELECT G
 :
22ø  G=SIN E
```

Od numeru wiersza 5ø do 2øø odwołania do funkcji trygonometrycznych (np. wiersz 1øø) wymagają, aby argument funkcji był przeliczony na radiany. Od wiersza 2øø argumenty funkcji trygonometrycznych muszą być wyrażane w gradach, aby wynik obliczeń był poprawny.

Następną zasadniczą różnicą w funkcjach standardowych omawianych systemów występuje w generatorze liczb losowych.

Funkcja RND generuje liczby losowe z przedziału [0,1]. W systemie WANG jest funkcją z argumentem RND(X). Argument X ma następujące znaczenie:

X = ø - wydawana jest pierwsza liczba losowa z "pseudolisty",
X ≠ ø - wydawana jest kolejna liczba losowa z "pseudolisty".

Różnica ta powoduje, że w systemie MERA otrzymuje się zawsze taki sam ciąg liczb losowych, natomiast w systemie WANG można otrzymać taki sam lub różny ciąg liczb losowych (w zależności od potrzeb) za każdym przebiegiem programu.

Przykład

```
1Ø   REM GENER LICZB LOSOWYCH
2Ø   FOR I=1 TO 5
3Ø   PRINT RND
4Ø   NEXT I
5Ø   PRINT "KONIEC GENERACJI"
6Ø   STOP
7Ø   END
```

Powyższy program w systemie MERA wydrukuje następujący ciąg liczb losowych:

```
.189292
.454239
.212831
.916276
.813Ø44
```

Powtórne uruchomienie programu spowoduje wydrukowanie takiego samego ciągu liczb losowych. Aby osiągnąć podobne działanie programu w systemie WANG program musiałby mieć postać:

```
1Ø   REM GENER LICZB LOSOWYCH
2Ø   PRINT RND(Ø)
3Ø   FOR I=1 TO 4: PRINT RND(1): NEXT I
4Ø   PRINT "KONIEC GENERACJI"
5Ø   END
```

Wyniki powyższego programu będą różne od programu:

```
1Ø   REM GENER
2Ø   FOR I=1 TO 5: PRINT RND(1): NEXT I
3Ø   PRINT "KONIEC GENERACJI"
4Ø   END
```

Struktura programu

Program w języku BASIC składa się z wierszy zawierających instrukcje przedstawione w punkcie poprzednim. Każdy wiersz opatrzony jest numerem określającym kolejność wykonywania się wierszy.

Numery w zależności od systemu zawierają się w granicach:

WANG - 1 ÷ 9999
MERA - 1 ÷ 32767

Długość wiersza programu nie może przekraczać 192 znaków, tzn. 3 wiersze na monitorze ekranowym w systemie WANG oraz 80 znaków w systemie MERA. Nie jest dopuszczalne kontynuowanie instrukcji w następnych wierszach. W wierszu musi mieścić się cała instrukcja; gdy wyrażenie arytmetyczne jest długie, trzeba podzielić go na części mieszczące się w wierszu.

W zasadzie kolejność wierszy programu wynika z kolejności wykonywania instrukcji zaplanowanej przez programistę, ale są wyjątki, które dotyczą tzw. instrukcji niewykonywalnych, dostarczających systemowi BASIC informacji, i tak:

- instrukcja COM musi występować na początku programu, może ją poprzedzać jedynie instrukcja REM,
- instrukcja DIM musi poprzedzać pierwsze odwołanie do zmiennych zadeklarowanych w instrukcji.

Natomiast deklaracje funkcji i podprogramów mogą występować w dowolnym miejscu w programie, nawet po odwołaniu się do tej funkcji i podprogramu.

Podsumowanie

System MERA-BASIC w porównaniu z systemem BASIC na maszynie cyfrowej WANG wykazuje wiele istotnych ograniczeń. Brak w nim możliwości korzystania z pamięci zewnętrznych, segmentacji progra-

mu, a w programie - możliwości korzystania ze zmiennych alfanumerycznych itd. Ze względu na duże potencjalne możliwości samego systemu MERA-400 wydaje się celowe rozbudowanie systemu BASIC-MERA o następujące elementy:

- segmentacja programu,
- odwołanie się do programu przez nazwę,
- śledzenie wykonywania się programu (TRACE),
- wydruk użytych zmiennych w programie,
- możliwość nadawania wartości zmiennym w trybie natychmiastowym,
- korzystanie z pamięci zewnętrznych (dysk, kaseta).

System BASIC-MERA powinien zmienić charakter: z kalkulatora elektronicznego w zestaw umożliwiający użytkownikowi pracę konwersacyjną, tak jak to gwarantuje system WANG.

Literatura

[1] System MERA-400. Dokumentacja techniczno-ruchowa. Tom III. Część III. Procesory systemowe systemu SOM-3. 11. BASIC. S-OF-4-00004-00A

[2] WANG 2200. Reference Manual, 1973

[3] WANG system 2200 BASIC Programming Manual, 1973

mgr inż. Zbigniew POZNAŃSKI
Instytut Technologii Elektronowej

SIMULA-67 - uniwersalny język programowania. Cz.2

W pierwszej części opisu języka Simula-67 wprowadzono podstawowe pojęcie tego języka, tj. pojęcie klasy. Simula-67 ma jednak silniejszy aparat opisu procesów i systemów wykorzystujący mechanizm prefiksowania klas i bloków. Problemy te, a także koncepcję wirtualności oraz definicję systemowej klasy SIMSET do operacji na strukturach listowych przedstawiono poniżej.

Podklasy

Przystępując do opisu złożonych procesów (systemów), często posiadamy o nich niewiele informacji. Dlatego początkowy opis jest zwykle bardzo ogólny i obejmuje najczęściej tylko ogólną charakterystykę obiektu. W miarę postępującej wiedzy o obiekcie, na etapie projektowania, jesteśmy w stanie wyodrębnić pewne klasy podproblemów, podprocesów, podsystemów. Zazwyczaj mają one różne struktury danych, jednak pierwotny, ogólny opis jest dla nich wszystkich wspólny. Powstaje w ten sposób hierarchia opisu problemów, procesów i systemów.

Jak wspomniano na wstępie, Simula 67 jest językiem opisu systemów, a zatem wyposażona jest w mechanizmy umożliwiające opis systemów w sposób podany wyżej. Do tego celu wprowadzono w Simuli pojęcie podklasy. Klasy można wykorzystać w charakterze prefiksów innych klas, które wtedy stają się podklasami tych pierwszych. Obiekt odpowiadający klasie z prefiksem ma złożoną strukturę danych i nazywa się obiektem złożonym. Zawiera on dane charakterystyczne dla swojej klasy i dane charakterystyczne prefiksu. Działanie obiektu przebiega wg algorytmu prefiksu i jego własnej klasy. Użytkownik może zatem wg własnego uznania tworzyć dowolne struktury klas odpowiadające treści badanych problemów. Klasy używane jako prefiksy innych klas mogą być definiowane systemowo, a ich definicje mogą stanowić części składowe języka.

Deklaracja klasy z prefiksem

Deklaracja klasy może być poprzedzona prefiksem, który jest nazwą innej klasy np.

```
class A(PA); SA;                    A class B(PB); SB
begin   DA;                          begin   DB;
   IA;                                  IB;
   inner;                               inner;
   FA;                                  FB;
end;                                 end;
```

gdzie PA,PB - zbiór parametrów formalnych klas A i B
SA,SB - zbiór specyfikacji parametrów formalnych klas A i B
DA,DB - zbiór deklaracji w klasach A i B
IA,IB - zbiór instrukcji początkowych w klasach A i B
FA,FB - zbiór instrukcji końcowych w klasach A i B

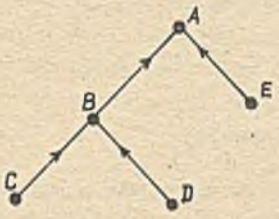
Mówimy, że klasa B jest prefiksowana przez klasę A. Treść deklaracji klasy prefiksowanej nazywa się ciałem właściwym tej klasy. Parametry formalne i atrybuty, których deklaracje są deklaracjami ciała właściwego klasy prefiksowanej nazywają się atrybutami właściwymi tej klasy.

Treść klasy, która powstaje z treści prefiksu przez uzupełnienie jego deklaracji deklaracjami ciała właściwego klasy prefiksowanej i przez wstawienie w miejscu 'inner' (lub przed końcowym end, jeśli symbol 'inner' nie występuje) ciągu instrukcji ciała właściwego klasy prefiksowanej nazywa się ciałem rozszerzonym klasy prefiksowanej. Atrybuty właściwe klasy prefiksowanej oraz atrybuty prefiksu noszą nazwę atrybutów klasy prefiksowanej.

Prefiksy mogą same być klasami prefiksowanymi. Ta własność umożliwia tworzenie dowolnych hierarchii klas w postaci, np.

```
    class A;
A class B...;
B class C...;
B class D...;
A class E...;
```

Graf odpowiadający tej hierarchii jest zorientowanym drzewem (rys. 13).



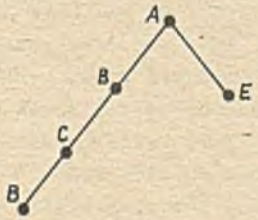
Rys. 13

Korzeń drzewa jest klasą, która nie ma prefiksu. Ciąg prefiksów danej klasy jest ciągiem klas występujących na drodze (jedynej) łączącej daną klasę z korzeniem, np. ciąg prefiksów klasy C jest następujący: C,B,A.

Mówimy, że klasa jest wewnętrzna w stosunku do jej prefiksów, np. klasa C jest wewnętrzna w stosunku do klas B i A. Klasy B i A są zewnętrzne w stosunku do klasy C.

Na prefiksowanie klas nakłada się następujące ograniczenia:

- żadna klasa nie może pojawić się w ciągu jej prefiksów. Prefiksowanie pokazane na rys. 14 jest więc błędne.



Rys. 14

- klasa może być użyta jako prefiks tylko na tym poziomie bloku, na którym jest zadeklarowana (wyjątek stanowią tu klasy systemowe), np.

```
begin
  class A;
  begin
    A class B..; $ <---- nieprawidłowo $
    ............
  end;
  A class C..;   $ <---- prawidłowo   $
  ............
end;
```

Rozważmy teraz prosty przykład, który dotyczy opisu typowych regulatorów przemysłowych. Zdefiniujmy zatem klasę regulator z atrybutami: sygnał_wejściowy, sygnał_wyjściowy. Mamy więc

```
class  regulator ;
begin
  real  sygnał_wejściowy, sygnał_wyjściowy;
end;
```

Zwróćmy uwagę na to, że regulator_P (proporcjonalny) ma wszystkie cechy regulatora zdefiniowanego powyżej, a ponadto atrybut własny, tj. współczynnik wzmocnienia. Można więc klasę regulator_P prefiksować klasą regulator, tzn.

```
regulator class regulator_P;
        begin
          real współczynnik_wzmocnienia;
        end;
```

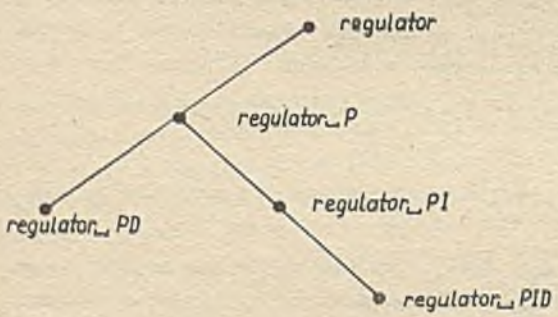
Obiekt klasy regulator_P posiada strukturę danych złożoną z atrybutów klasy regulator_P, tzn. atrybutów właściwych klasy regulator_P oraz atrybutów klasy regulator. W ten sam sposób można opisać także takie regulatory jak PD (proporcjonalno-różniczkujący), PI (proporcjonalno-całkujący) czy PID (proporcjonalno-całkujący-różniczkujący). Mamy więc

```
regulator_P class regulator_PD;
            begin
              real czas_wyprzedzenia;
            end;

regulator_P class regulator_PI;
            begin
              real czas_zdwojenia;
            end;

regulator_PI class regulator_PID;
             begin
               real czas_wyprzedzenia;
             end;
```

Przedstawionej strukturze regulatorów odpowiada drzewo na rys. 15.

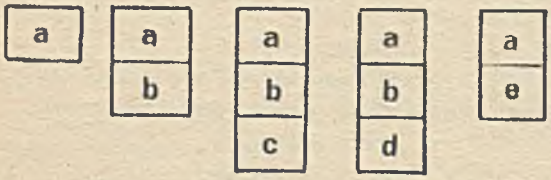


Rys. 15

Złożenie klas (konkatenacja)

Z problemem tym spotkaliśmy się w poprzednim punkcie tworząc strukturę regulatorów. Teraz podamy zasady ogólne.

Podklasa jest równoważna klasie otrzymanej przez złożenie wszystkich klas tworzących przy porządkowany jej ciąg prefiksów, np. klasa C (rys. 13) jest równoważna klasie powstałej przez złożenie klas A,B i C. Podobnie klasa regulator_P (rys. 15) jest równoważna klasie powstałej przez złożenie klasy regulator z klasą regulator_P. Obiekt takiej klasy nazywamy obiektem złożonym. Proces złożenia można wyjaśnić na przykładzie pola zajmowanego przez strukturę danych obiektu złożonego. Powstaje ona przez zsumowanie pól zajmowanych przez struktury danych klas tworzących ciąg prefiksów. Wyjaśnia to rys. 16, na którym pola a,b,c,d,e odnoszą się do klas A,B,C,D,E z rys. 13.



Rys. 16

Niech będą dane*):

```
class A(PA); SA;          A class B(PB); SB;
begin  DA;                begin  DB;
  IA;                       IB;
  inner;                    inner;
  FA;                       FB;
end;                      end;
```

Podklasa B jest równoważna następującej fikcyjnej klasie K:

---

*) Przyjęto oznaczenia jak w p. "Deklaracja klasy z prefiksem".

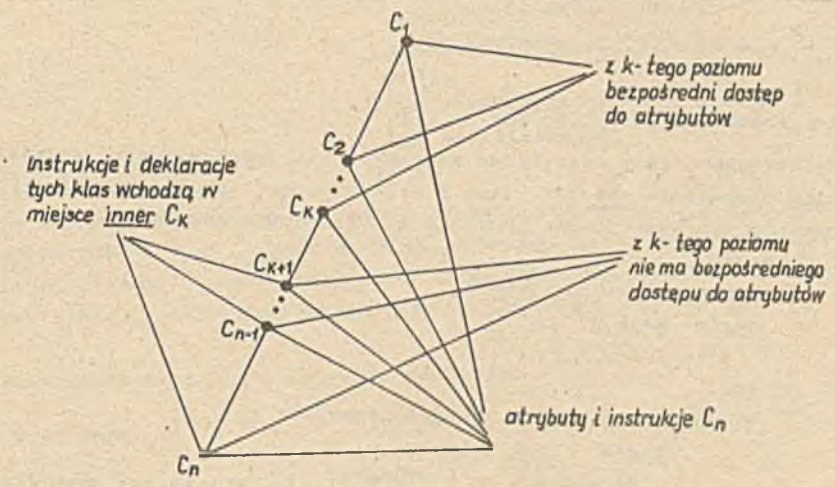
```
class K(PA,PB); SA,SB;
begin
  DA;DB
  IA;
  IB;
  inner;
  FB;
  FA;
end;
```

Treść klasy K powstała w wyniku zastąpienia symbolu 'inner' w treści klasy A ciągiem symboli:

IB; inner; FB;

oraz połączeniu deklaracji DA i DB. Taką właśnie funkcję spełnia symbol 'inner'. Z jednej strony jest on instrukcją pustą, z drugiej zaś określa miejsce wstawienia ciągu symboli stanowiących część "instrukcyjną" treści klasy B. Pominięcie symbolu 'inner' jest równoważne wstawieniu go przed końcowym symbolem end danej klasy.

Niech C_n będzie klasą z ciągiem prefiksów $C_1, C_2, \ldots, C_{n-1}$. Niech ponadto X będzie zmienną, której wartością jest referencja do obiektu należącego do klasy C_n ($ref(C_n)X$). W dalszej części będziemy niekiedy utożsamiali zmienną z jej wartością, mówiąc np. o obiekcie X. Mechanizm złożenia ma następujące własności (rys. 17):



Rys. 17

- obiekt X ma atrybuty będące sumą atrybutów zdefiniowanych w klasach $C_1, \ldots, C_n$; atrybut zadeklarowany w klasie C_k jest zdefiniowany na k-tym poziomie prefiksowania,
- algorytm działania obiektu X składa się ze wszystkich instrukcji zawartych w $C_1, C_2, \ldots, C_n$, wykonywanych w określonej kolejności,
- z k-tego poziomu prefiksowania obiektu X jest bezpośredni dostęp do wszystkich jego atrybutów zdefiniowanych na poziomie k-tym lub w klasach zewnętrznych do k-tego, tzn. $C_1, C_2, \ldots C_k$; wyjątek stanowią tu atrybuty "zakryte" przez definicje konfliktowe, kiedy to bezpośredni dostęp nie jest możliwy; zasady postępowania w takiej sytuacji omówione będą w punkcie "Referencja lokalna this";
- z k-tego poziomu prefiksowania nie ma dostępu do atrybutów obiektu X zdefiniowanych na poziomach prefiksowania wewnętrznych do k-tego tzn. $C_{k+1}, \ldots, C_n$, z wyjątkiem bezpośredniego dostępu do wielkości wirtualnych, p."Wielkości wirtualne".

- na k-tym poziomie prefiksowania symbol 'inner' reprezentuje te instrukcje i deklaracje w algorytmie działania obiektu X, które należą do poziomów prefiksowania wewnętrznych do k-tego, tzn. $C_{k+1},\ldots,C_n$.

Nawiązując do przykładu z punktu "Deklaracja klasy z prefiksem" obiekt należący do klasy regulator_PD ma, oprócz swoich, atrybuty klasy regulator_P i klasy regulator z bezpośrednim dostępem do nich (jeśli nie są konfliktowe). Obiekt klasy regulator_P ma, oprócz swoich, tylko atrybuty klasy regulator. Z obiektu klasy regulator_P nie ma jednak dostępu do atrybutów klasy regulator_PD.

Generacja obiektów złożonych

Zasady generacji obiektów złożonych niewiele różnią się od generacji obiektów prostych. Niech będą dane:

$$\underline{\text{class}}\ A\left(x_1^F, x_2^F, .., x_n^F\right) \qquad \text{oraz}$$

$$A\ \underline{\text{class}}\ B\left(y_1^F, y_2^F, .., y_k^F\right),$$

gdzie x_i^F, y_j^F - parametry formalne $(i=1,..,n;\ j=1,..,k)$.

Obiekt złożony klasy B generuje się za pomocą instrukcji new, tzn.

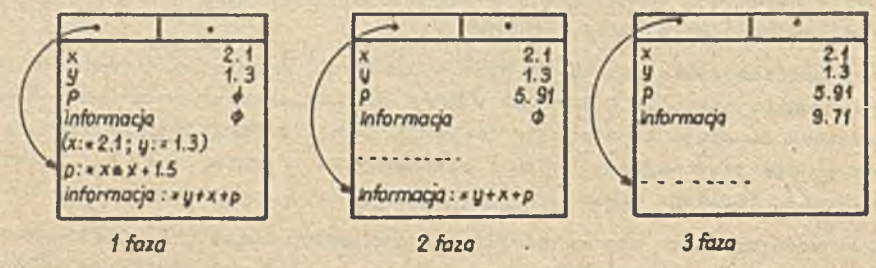
$$\underline{\text{new}}\ B\left(x_1^A, x_2^A, .., x_n^A, y_1^A, y_2^A, .., y_k^A\right),$$

gdzie x_i^A, y_j^A - parametry aktualne $(i=1,..,n;\ j=1,..,k)$.

Przy tworzeniu obiektu złożonego należy więc podać wszystkie parametry aktualne prefiksu i klasy danej, tzn. B. Z chwilą utworzenia obiektu klasy B działa mechanizm złożenia klas (pkt "Złożenie klas") A,B i rozpoczyna się wykonywanie instrukcji obiektu złożonego, np.

```
begin
 class system(x); real x;
 begin  real p;
   p := x*x + 1.5;
 end;
 system class system_informacyjny(y); real y;
        begin   real informacja;
          informacja := y + x + p;   { w klasie tej jest dostęp
        end;                           do tych atrybutów
 new system_informacyjny(2.1,1.3);
end;
```

Rys. 18 ilustruje kolejne fazy generacji obiektu klasy system_informacyjny.



Rys. 18

Inny przykład dotyczy wykorzystania symbolu 'inner'. Niech będą dane 2 obiekty klasy obrabiarka, których algorytm działania nieznacznie różni się, tj. jedna obrabiarka wykonuje kolejno operacje toczenia, wiercenia i frezowania, zaś druga przed frezowaniem wykonuje dodatkowo gwintowanie. Program ma następującą postać:

```
begin
  class obrabiarka_1;
  begin
    procedure toczenie ...;
    procedure wiercenie ...;
    procedure frezowanie ...;
    while true do
    begin
      toczenie;
      wiercenie;
      inner;
      frezowanie;,
    end;
  end;
  obrabiarka_1 class obrabiarka_2;
  begin
    procedure gwintowanie ...;
    gwintowanie;
  end;
  new obrabiarka_1;   $ <-- 01 $
  new obrabiarka_2;   $ <-- 02 $
end;
```

Instrukcja 01 spowoduje wykonanie treści klasy obrabiarka_1, gdzie symbol 'inner' traktowany jest jako instrukcja pusta. Instrukcja 02 spowoduje wykonanie treści klasy obrabiarka_1, przy czym w miejsce 'inner' wywoływana będzie dodatkowo procedura gwintowanie. W ten sposób na dwóch obrabiarkach realizowane są dwa różne algorytmy obróbki.

Podstawienia referencyjne dla obiektów złożonych

Niech będą dane:

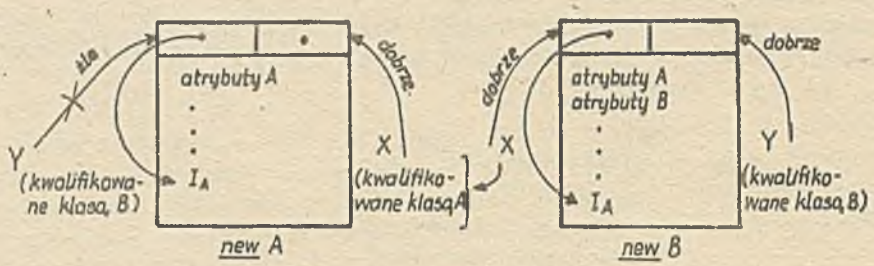
```
      class A;
A class B;
      ref(A)X; ref(B)Y; .
```

Zmiennej referencyjnej X, kwalifikowanej klasą A, można nadać wartość równą referencji do obiektu klasy A lub jej podklasy. Wszystkie instrukcje, w których nadaje się zmiennej X referencję do obiektu klasy np. C nie będącej podklasą klasy A są nieprawidłowe. A zatem prawidłowe są instrukcje:

```
X:- new A      (X kwalifikowane klasą A)
Y:- new B      (Y kwalifikowane klasą B)
X:- new B      (X kwalifikowane klasą A, a B jest podklasą
                klasy A).
```

Nieprawidłowa jest instrukcja Y:- new A, gdyż zmienna Y kwalifikowana jest klasą B, a klasa A nie jest podklasą klasy B.

Powyższe rozważania bliżej ilustruje rys. 19.



Rys. 19

Przykład: Należy wygenerować dwie liczby zespolone 2+3i, 3+4i oraz jedną liczbę rzeczywistą 8.

```
begin
  class liczba_rzeczywista (część_rzeczywista);
  real część_rzeczywista;
  liczba_rzeczywista class liczba_zespolona (część_urojona);
  real część_urojona;
  ref(liczba_rzeczywista) ALFA,BETA;
  ref(liczba_zespolona) GAMMA;
  ALFA:- new liczba_rzeczywista(8.Ø);
  GAMMA:- new liczba_zespolona(2.Ø,3.Ø);
  BETA:- new liczba_zespolona(3.Ø,4.Ø);
end;
```

Operatory "is", "in"

W celu sprawdzania przynależności obiektu do danej klasy wprowadzono operatory "is" i "in". Wyrażenie boolowskie X is C daje wartość true, jeśli wartością wyrażenia X jest referencja do obiektu należącego do klasy C.
Wyrażenie boolowskie X in C daje wartość true, jeśli wartością wyrażenia X jest referencja do obiektu należącego do klasy C lub podklasy klasy C (klas wewnętrznych do C).

Przykład: Niech będzie dany program:

```
begin
  class C;
C class D;
  ref(C)X;
  X:- new D;
end;
```

Zwróćmy tu uwagę na to, że wartością zmiennej X jest referencja do obiektu klasy D będącego podklasą klasy C. Mamy więc:

X is C - false
X in C - true
X is D - true
X in D - true.

Zasady dostępu do atrybutów obiektów złożonych

Zasady dostępu do atrybutów obiektów prostych i złożonych są podobne. W obu wypadkach stosuje się odległy dostęp "kropkowany" oraz tzw. mechanizm połączenia (inspect), omówiony dalej. Jednak w wypadku obiektów złożonych występują pewne subtelności, związane z kwalifikacjami zmiennych referencyjnych. Często zachodzi potrzeba zmiany kwalifikacji tych zmiennych.

Istnieją także określone reguły dostępu do atrybutów konfliktowych.

Odległy dostęp "kropkowany"

Rozważania rozpoocznijmy od przykładu:

```
                begin
                  class A(x); real x;
                A class B(y); boolean y;
                  if x>0 then y:= not y;
                  ref(A)a,c; ref(B)b;
a)                a:- new A(5.0);
b)                b:- new B(3.Ø,false);
c)                c:- new B(-5.Ø,true);
                end;
```

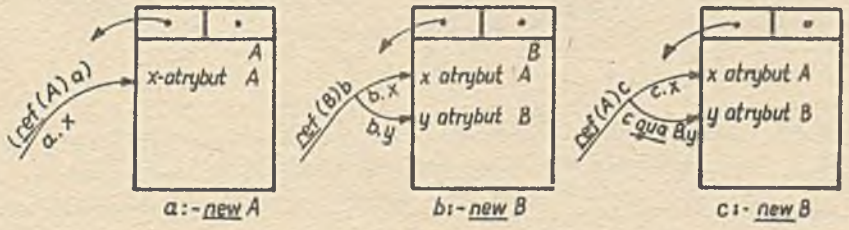
W wypadku a) przez zmienną a, której wartością jest referencja do obiektu klasy A mamy dostęp do wszystkich atrybutów klasy A. Prawidłowy jest więc zapis: a.x.
W wypadku b) przez zmienną b kwalifikowaną klasą B,i której wartością jest referencja do obiektu tej klasy istnieje dostęp do wszystkich atrybutów obiektu złożonego (atrybutów klasy A i B). Prawidłowe zatem są: b.x, b.y.
W wypadku c) zmienna c kwalifikowana jest klasą A, lecz jej wartość jest referencją do obiektu klasy B będącego podklasą klasy A. Istnieje ogólna zasada, która mówi, że za pomocą zmiennej referencyjnej uzyskuje się dostęp tylko do atrybutów obiektu klasy, którą jest ona kwalifikowana. A zatem zmienna c daje dostęp "kropkowany" tylko do atrybutów klasy A, tzn. prawidłowe jest - c.x. Aby za pomocą zmiennej c dostać się do atrybutów klasy B należy zmienić jej kwalifikację na klasę B. W Simuli 67 zmiana ta realizuje się następująco:

```
c qua B.
```

Teraz dostęp do atrybutu y klasy B za pomocą zmiennej c ma postać:

```
c qua B.y.
```

Powyższe rozważania ilustruje rys. 20.



Rys. 20

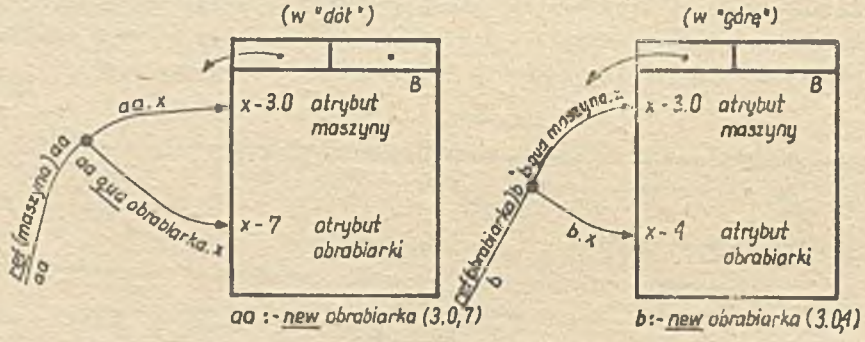
Rozważmy kolejne przykłady.

```
● begin
    class sterownik; real u;
    sterownik class komputer; real y;
    ref(sterownik) X;Z;
    ref(komputer) Y;
    X :- new sterownik;
    Y :- new komputer;
    Z :- new komputer;
  end;
```

Prawidłowe jest: X.u, Y.u. Y.y, Z qua komputer.y .

```
begin
  class maszyna(x); real x;
  maszyna class obrabiarka(x); integer x;
  ref(maszyna)a,aa; ref(obrabiarka)b,bb;
  a :- new maszyna(5.0);
  b :- new obrabiarka(3.0,4);
  aa :- new obrabiarka(3.0,7);
  bb :- new maszyna(0.); $ instrukcja nieprawidłowa $
  $ wartości atrybutów są następujące: a.x=5.0,b.x=4,
  aa.x=3.0, b qua maszyna.x=3.0,aa qua obrabiarka.x=7 $
end;
```

Zwróćmy uwagę na to, że w przykładzie tym wystąpiły zmienne konfliktowe x. Rozważmy obiekt, do którego referencję daje zmienna b. Przez b.x uzyskujemy dostęp do zmiennej x zadeklarowanej w klasie obrabiarka, gdyż tą klasą kwalifikowana jest zmienna b. Chcąc uzyskać dostęp do zmiennej x zadeklarowanej w klasie maszyna musimy zmienić kwalifikację zmiennej b na klasę maszyna. Wtedy b qua maszyna.x jest wartością zmiennej x klasy maszyna. Gdyby konflikt nie wystąpił, tzn. w klasie maszyna zadeklarowanoby np. zmienną y, to dostęp do niej byłby prosty - b.y bez potrzeby zmiany kwalifikacji. Zasady zmiany kwalifikacji "w dół" i "w górę", dla powyższego przykładu, ilustruje rys. 21.



Rys. 21

Wynikają stąd trzy wnioski:

a) zmienna referencyjna do obiektu, daje dostęp "kropkowany" do jego atrybutów na poziomie jej kwalifikacji

b) zmiana kwalifikacji zmiennych referencyjnych "w górę" stosuje się w wypadku wystąpienia nazw konfliktowych

c) zmiana kwalifikacji zmiennych referencyjnych "w dół" stosuje się przy próbie dostępu do atrybutów obiektu zadeklarowanych w klasie będącej podklasą klasy kwalifikującej daną zmienną.

Powyższe wnioski zilustrujmy jeszcze jednym przykładem.

```
begin
  class A; real P,R;
A class B; boolean P,Q;
  ref(A)X; ref(B)Y;
  X:-Y:- new B;
end;
```

Zasady dostępu do atrybutów klasy A i B, w powyższym przykładzie podano w tab. 3.

Tab.3.

Dostęp do atrybutu		Typ atrybutu
X.P	Y qua A.P	real
X.R	Y.R	real
X qua B.P	Y.P	boolean
X qua B.Q	Y.Q	boolean

Mechanizm połączenia (inspect)

Dotychczas przedstawiono jeden typ dostępu odległego, tzw. dostęp "kropkowany". Istnieje jeszcze jeden rodzaj dostępu do atrybutów obiektu danej klasy, wykorzystujący mechanizm połączenia.

Niech będzie dany program:

```
begin
  class system_sterowania;
  begin
    real wejście,wyjście;
    ..........
  end;
  ref(system_sterowania)Z;
  Z :- new system_sterowania;
  Z.wyjście:=Z.wejście;
end;
```

Przez zmienną referencyjną Z mamy dostęp do atrybutów nowoutworzonego obiektu klasy A, tzn. Z.wejście, Z.wyjście. Ten sam efekt można uzyskać wykorzystując instrukcję połączenia, tzn.

```
inspect Z do
begin
  wyjście := wejście;
  .........
end;
```

Powyższą instrukcję traktuje się jako blok przyłączony. W bloku tym możliwy jest bezpośredni dostęp do wszystkich atrybutów klasy, do której należy obiekt wskazany przez zmienną Z. Istnieją dwa sposoby tworzenia bloków przyłączonych, wykorzystujące dwa typy instrukcji połączenia:

a) inspect Z do $<I^0>$ otherwise $<I^*>$;

b) inspect Z when <nazwa klasy 1> do $<I^1>$
when <nazwa klasy 2> do $<I^2>$
.....
when <nazwa klasy n> do $<I^n>$
otherwise $<I^*>$;

gdzie I^0, I^*, I^1,..., I^n - instrukcje.

Część 'otherwise' może być pominięta.

Przykłady:

- inspect Z do x:=y;
- inspect Z when regulator do wejście:=5.3
  when punkt do x:=3.
  when linia do P:- none
  otherwise goto KONIEC;

W wypadku a), jeżeli Z=none, to wykonywana jest instrukcja I^* w przeciwnym razie - I^o. Wykonanie instrukcji połączenia można opisać następująco:

- obliczana jest wartość wyrażenia referencyjnego Z w instrukcji połączenia,
- jeśli występują części "when", to są one analizowane jedna po drugiej; jeśli Z jest referencją do obiektu należącego do klasy równej lub wewnętrznej do tej, której nazwa podana jest po symbolu "when", tzn. jeśli Z in <nazwa klasy i> = true, wtedy wykonywana jest i-ta instrukcja; pozostałe części "when" są pomijane,
- instrukcja występująca po symbolu "otherwise" jest wykonywana wtedy, gdy Z == none lub Z nie jest referencją do obiektu żadnej z klas występujących po symbolach "when".

Podczas wykonywania bloku przyłączonego mówimy, że obiekt Z został przyłączony. Blok przyłączony ma kwalifikację klasą, którą w wypadku a) kwalifikowana jest zmienna referencyjna Z, natomiast w wypadku b) - jej nazwa jest podana po symbolu "when".

Rozważmy przykłady:

- ```
  class automat; real we,wy,parm;
  class operator; real we,wy,nazwa;
  ref(automat)X; ref(operator)Y;
  X :- new automat; Y :- new operator;
  a) inspect X do
     begin
       we:=3.0;
       wy:=4.0;
     end;
  ```
  jest równoważne
  ```
  X.we:=3.0;
  X.wy:=4.0;
  ```
  ```
  b) inspect X do
     inspect Y do
     begin
       X.we:=3.0;
       we:=4.0;
       parm:=4.0;
       nazwa:=6.0;
     end;
  ```
  klasy automat i operator nie mogą być w jednym ciągu przedrostków

Wewnętrzny blok przyłączony ma priorytet wyższy niż blok zewnętrzny z punktu widzenia dostępu do atrybutów. Atrybutami obiektu Y są we,wy,nazwa. Są one dostępne bezpośrednio. Bezpośrednio dostępne są także niekonfliktowe atrybuty obiektu X. Chcąc wykorzystać konfliktowy atrybut we klasy automat musimy odwołać się do niego przez zmienną referencyjną X w celu rozróżnienia go od identycznego atrybutu klasy operator.

```
c) inspect Y do
   begin
     boolean we;          ta zmienna jest dostępna bezpośrednio
     we:=wy > 0;
     if we then Y.we:=2.;
   end;
```

Zmienne zadeklarowane w bloku przyłączonym mają największy priorytet. Jeśli w klasie operator zadeklarowano zmienną we i Y jest referencją do obiektu tej klasy, to w bloku przyłączonym kwalifikowanym klasą operator zmienna we jest niedostępna bezpośrednio, gdyż w bloku tym zadeklarowano zmienną o tej samej nazwie (np. we).

Atrybuty klasy stanowiącej kwalifikację bloku przyłączonego oraz niekonfliktowe atrybuty klas zewnętrznych do niej (jeśli istnieją) dostępne są w tym bloku bezpośrednio (zasady dostępu do atrybutów konfliktowych w blokach przyłączonych podano w punkcie "Referencja lokalna this"). Atrybuty klas wewnętrznych w stosunku do klasy kwalifikującej blok przyłączony dostęp-
```

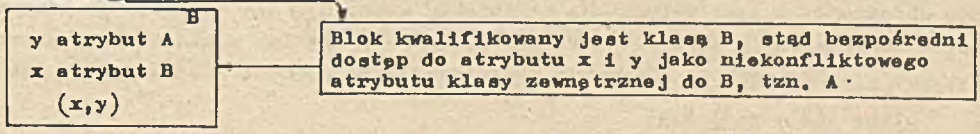
ne są przez zmianę kwalifikacji bądź w sposób podany w wyżej przedstawionym punkcie.

- Niech będą dane deklaracje:

```
  class A(y); integer y;
A class B(x); real x;
  ref(A)X; ref(B)Y;
```

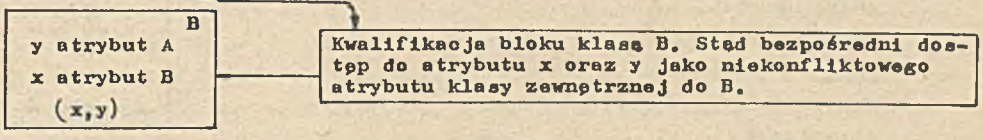
a) Y :- new B(.); istnieją wtedy dwa sposoby dostępu do atrybutów x,y:

- Y.x, Y.y ,
- inspect (Y) do



b) X :- new B(.); istnieją tu cztery sposoby dostępu do atrybutów x,y:

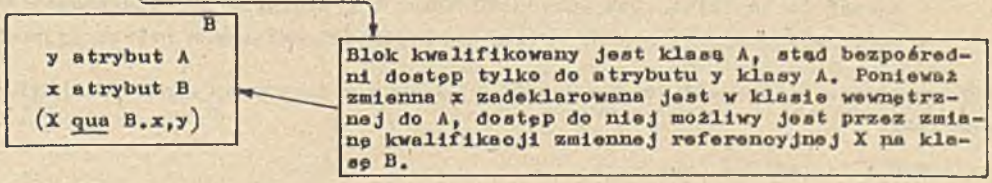
- X qua B.x, X.y,
- inspect X when (B) do



- inspect X qua B do
  (x,y)

Przez zmianę kwalifikacji zmiennej X, blok przyłączony, tak jak poprzednio, kwalifikowany jest klasą B. Umożliwia to bezpośredni dostęp do atrybutów x,y w tym bloku.

- inspect (X) do



Należy podkreślić, że stosowanie mechanizmu połączenia jest wygodne w szczególności wtedy, gdy wykorzystujemy większą liczbę atrybutów danej klasy. Dostęp "kropkowany" może być wtedy uciążliwy.

Referencja lokalna "this"

W programie obiektu klasy A lub klasy prefiksowanej klasą A referencja do tego obiektu może być oznaczona przez this A, gdzie A jest nazwą klasy. Wyrażenie this A jest wyrażeniem referencyjnym. Wykorzystuje się je wtedy, gdy chcemy użyć nazwy obiektu należącego do klasy A w treści tej klasy, np.

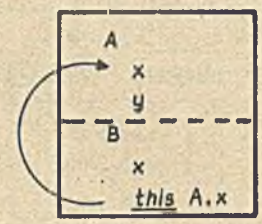
```
begin
  ref(komputer)Y;
  class komputer;
  begin
    ref(łącze)procedure sprzęgnięcie(A);
    ref(komputer)A;
    if A =/= none then
    sprzęgnięcie:-new łącze(this komputer,A);
    Y:- this komputer;
  end;
  class łącze(K1,K2); ref(komputer)K1,K2;
end;
```

Procedura sprzęgnięcie generuje łącze pomiędzy komputerem A i komputerem, w którym jest ona zadeklarowana (this komputer). Podobnie zmienna Y dostarcza referencję do obiektu klasy komputer, w którym wykonana zostanie instrukcja Y:-this komputer.
Wyrażenie referencyjne this A ma sens, jeśli użyte zostanie:

a) w treści klasy A lub treści dowolnej podklasy klasy A,
b) w bloku przyłączonym, kwalifikowanym klasą A lub jej podklasą.

Własność a) można wykorzystać w odległym dostępie do atrybutów konfliktowych. Przy omawianiu mechanizmu złożenia klas (pkt "Złożenie klas") stwierdzone, że z k-tego poziomu prefiksowania obiektu X(X jest referencją do obiektu klasy C_n) nie istnieje bezpośredni dostęp do atrybutów konfliktowych zdefiniowanych na poziomie k-tym lub w klasach zewnętrznych do C_k, tzn. $C_1,\ldots,C_{k-1}$. Trudność tę eliminuje zastosowanie referencji lokalnej this. Rozważmy przykład:

```
•   class A(x); real x,y;
  A class B(x); boolean x;
    begin
     $ w bloku tym bezpośrednio
       dostępne są:
       boolean  x
       real     y .
       Dostęp do real x jest przez
       this A.x $
    end;
```

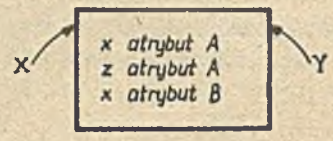


W przykładzie tym wyrażenie this A ma sens ponieważ klasa, w której wystąpiło (B) jest podklasą klasy A.

Uwaga: Wyrażenie referencyjne this A stosuje się w treści klasy prefiksowanej B w celu dostępu do konfliktowych atrybutów klasy A, a zatem wewnątrz obiektu złożonego. Chcąc uzyskać dostęp do konfliktowych atrybutów klasy A z zewnątrz obiektu złożonego, należy zastosować, omówione już, metody zmiany kwalifikacji zmiennych referencyjnych (qua).

Wykorzystanie referencji lokalnej this w bloku przyłączonym (własność b)) ilustruje kolejny przykład.

```
•   ref(A)X; ref(B)Y;
    class A(x,z); real x,z;
  A class B(x); boolean x;
    X:- Y:- new B(.);
    a) inspect Y do
       begin
       $ real z, boolean x -  dostępne bezpośrednio
         this B   -  prawidłowe
         real x - dostępne przez this A.x , Y qua A.x , X.x $
       end;
       Blok przyłączony kwalifikowany jest klasą B, stąd możliwy jest bezpośredni dostęp do
       zmiennej boolowskiej x oraz niekonfliktowego atrybutu klasy A - z. Atrybut konfliktowy
       x klasy A jest tu niedostępny w sposób bezpośredni. Podano trzy sposoby dostępu do
       tego atrybutu.
    b) inspect X when B do
       (inspect X qua B do)
       begin
       $ real z, boolean x - dostępne bezpośrednio
         this B - prawidłowe
         real x - dostępne przez this A.x , X.x $
       end;
```



o) inspect X when A do
begin
$ real z, real x - dostępne bezpośrednio
boolean x - dostępne przez: X qua B.x, this A qua B.x $
end;
Blok przyłączony kwalifikowany jest klasą A, stąd możliwy jest dostęp bezpośredni do zmiennych rzeczywistych y,z. Bezpośredni dostęp do jakichkolwiek atrybutów konfliktowych i niekonfliktowych klasy B jest niemożliwy.

Z powyższych rozważań wynika, że zastosowanie referencji lokalnych (this) jest pomocne przy dostępie do atrybutów konfliktowych w obiektach złożonych, w szczególności, gdy sposób dostępu wykorzystuje mechanizm połączenia.

Wielkości wirtualne

Przystępując do opisu systemu czy procesu często zmuszeni jesteśmy stosować pewne procedury, których efekt działania jest nam znany, lecz nie potrafimy na danym etapie (bądź nie jest to naszym zadaniem) zdefiniować ich treści. Sytuacja taka ma miejsce w szczególności wtedy, gdy jeden program pisany jest przez kilka nie komunikujących się ze sobą osób, z których każda wykorzystuje procedury definiowane przez pozostałe osoby. Wykorzystując koncepcję wirtualności można używać nazw procedur bez podawania ich treści, deklarując je tylko jako tzw. wielkości wirtualne.

Wielkości wirtualne specyfikowane są w deklaracjach klas, po deklaracji parametrów formalnych klasy i poprzedzone symbolem virtual:. Jako wielkości wirtualne specyfikowane mogą być: procedury, procedury funkcyjne, etykiety i przełączniki.
Rozważamy następujący przykład:

```
● class geometria ; virtual: real procedure odległość;
  begin
  .....
   x:=odległość (a,b);    $ a) $
  end;
```

Specyfikując procedurę jako wielkość wirtualną nie podaje się jej parametrów formalnych (jeśli występują). W powyższym przykładzie zadeklarowano klasę geometria, przy czym zachodzi konieczność wykorzystania procedury obliczania odległości między dwoma punktami. Przypuśćmy, że na tym etapie nie potrafimy jej zdefiniować. Specyfikując jednak procedurę odległość jako wielkość wirtualną możemy posługiwać się jej nazwą w treści całej klasy geometria.

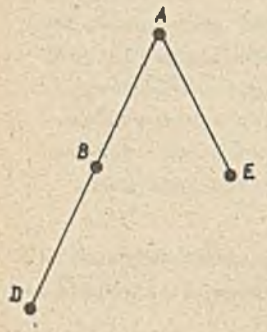
Wprowadźmy teraz pojęcia geometrii euklidesowej i geometrii Łobaczewskiego i zdefiniujmy w nich procedury odległość. Obie te geometrie mają wszystkie własności zadeklarowanej wcześniej klasy geometria, stąd:

```
geometria class geometria euklidesowa;
  begin
     real procedure odległość (x,y);
      < treść procedury > ;
      ........
end;
geometria class geometria Łobaczewskiego;
begin
     real procedure odległość (x,y);
       < treść procedury>;
     ................
end;
```

Działanie mechanizmu wirtualności polega na tym, że w zależności od tego, jaki obiekt zostanie utworzony, w podstawieniu a) tzn. x:=odległość (a,b), wykonana będzie procedura zadeklarowana w odpowiedniej podklasie klasy geometria. I tak po utworzeniu obiektu klasy geometria euklidesowa (new geometria euklidesowa), każde wywołanie procedury odległość w tym obiekcie wykorzystywać będzie treść procedury zadeklarowanej w klasie geometria euklidesowa. Widać stąd, że na pierwszym poziomie prefiksowania (geometria) dostępne są w sposób bezpośredni atrybuty (procedury) zadeklarowane na wewnętrznych poziomach prefiksowania (w podklasach).
Rozważmy kolejny przykład:

```
      class A ; virtual ; procedure X ; begin .. ; X ; .. end ;
A class B ; begin procedure X <treść 1> ; .... X ; .. end ;
B class D ; begin procedure X <treść 2> ; .... X ; .. end ;
A class E ; begin procedure X <treść 3> ; .... X ; .. end ;
```

Drzewo odpowiadające tej hierarchii ma postać



Z chwilą wygenerowania obiektu klasy B (new B) wywołania procedur X w treściach klas A i B odnoszą się do procedury, zadeklarowanej w klasie B. Jeżeli utworzony będzie obiekt klasy D to wywołania procedur X w treściach klas A,B,D odnosić się będą do procedury X, zadeklarowanej w klasie D. Podobnie w chwili wygenerowania obiektu klasy E, wywołania procedur X w klasach A i E odnosić się będą do procedury X, zadeklarowanej w klasie E.
W momencie utworzenia obiektu klasy A wywołanie procedury X odnosi się do procedury X zadeklarowanej w klasie A. Jeżeli w klasie A nie zadeklarowano procedury X (zob. przykład) to w chwili jej wywołania sygnalizowany jest błąd. Dla wszystkich wcześniej omówionych sytuacji deklaracja procedury X na poziomie specyfikacji wielkości wirtualnej tzn. w klasie A nie jest konieczna.

Wielkość wirtualna obiektu jest identyfikowana za pomocą tzw. "atrybutu dopasowującego"(w naszym przykładzie jest to procedura X). "Atrybut dopasowujący" musi być tego samego typu oraz mieć tę samą nazwę co wielkość wirtualna. Jeśli deklaracje atrybutu dopasowującego są podane na więcej niż jednym poziomie prefiksowania w hierarchii klas, wtedy brany jest ten z poziomu najbardziej wewnętrznego. Obowiązuje on wówczas na wszystkich poziomach prefiksowania równych lub wewnętrznych do tego, gdzie występuje specyfikacja wielkości wirtualnej. Wynika stąd możliwość wykorzystania na danym poziomie prefiksowania procedur zadeklarowanych na wewnętrznych poziomach. Jest to jedyny sposób dostępu bezpośredniego w treści danej klasy do atrybutów zadeklarowanych w jej podklasach. Kolejny przykład ilustruje wykorzystanie koncepcji wielkości wirtualnych w procesach obsługi:

```
class kontrola_jakości_wyrobów; virtual; procedure obsługa_wyrobu;

begin
  boolean kontrola_wyrywkowa;
  while true do
  if kontrola_wyrywkowa then obsługa_wyrobu;
end;
kontrola_jakości_wyrobów class wyrób_dobry;
begin
  procedure obsługa_wyrobu;
  begin
    ........
    poślij_wyrób_do_dalszej_obróbki;
    ........
  end;
```

```
end;
kontrola_jakości_wyrobów class  wyrób_zły;
begin
  procedure obsługa_wyrobu;
  begin
    real prob;  prob:=0.9;
    if draw(prob,U) then usuń_wyrób;
  end;
end;
```

W programie powyższym wyroby wadliwe, po wykryciu będą z prawdopodobieństwem 0.9 likwidowane (z chwilą utworzenia obiektu klasy wyrób_zły, wywołanie procedury obsługa_wyrobu w treści klasy kontrola_jakości_wyrobów odnosi się do procedury zadeklarowanej w treści klasy wyrób_zły), wyroby dobre zaś, obsługiwane będą wg procedury zadeklarowanej w treści klasy wyrób_dobry.

Prefiksowanie bloków

Niech będzie dana pewna klasa, w której zdefiniowano złożoną strukturę danych, np.

```
class bank_danych_o_procesie_produkcyjnym;
begin
  class struktura_przestrzenna_procesu;
  class struktura_stanowisk;
  < procedury sterowania linią produkcyjną >;
end;
```

Z dotychczasowych rozważań wynika, że chcąc wykorzystać atrybuty tej klasy w obiekcie np. klasy wyrób, należałoby klasę wyrób prefiksować klasą bank_danych_o_procesie_produkcyjnym. Zwróćmy jednak uwagę na to, że jeżeli w programie wygenerujemy 1000 obiektów klasy wyrób, to za każdym razem zmuszeni jesteśmy generować złożone struktury danych zawarte w klasie prefiksującej. Jest to działanie bardzo nieoszczędne z punktu widzenia wykorzystania pamięci komputera. Wygodny więc byłby mechanizm zapewniający jednokrotne wygenerowanie obiektu klasy bank_danych_o_procesie_produkcyjnym, z możliwością bezpośredniego dostępu do jego atrybutów przez obiekty np. klasy wyrób. W tym celu stworzono w Simuli możliwość prefiksowania bloków, np.

```
bank_danych_o_procesie_produkcyjnym begin
                                      class wyrób;
                          struktura_stanowisk class gniazdo_produkcyjne;
                                      end;
```

Nazwę klasy można prefiksować zwykły blok*). Powoduje to złożenie (konkatenację) danej klasy z prefiksowanym blokiem (podobnie do konkatenacji klas). Oznacza to, że wszystkie atrybuty danej klasy prefiksującej blok są dostępne w tym bloku. Mogliśmy więc w bloku prefiksowanym klasą bank_danych_o_procesie_produkcyjnym klasę gniazdo_produkcyjne prefiksować klasą struktura_stanowisk, będącą atrybutem klasy prefiksującej blok. Nazwa klasy prefiksującej blok odnosi się do klasy zadeklarowanej na poziomie deklaracji bloku prefiksowanego**). Jeżeli jest to nazwa klasy systemowej, odnosi się ona do fikcyjnej deklaracji tej klasy. Ponadto należy zazna-

---

*) Jeżeli klasa prefiksująca blok ma parametry formalne, to prefiksującą nią blok należy podać parametry aktualne

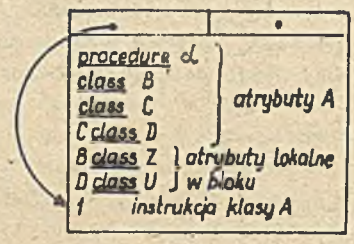
**) Wyjątek stanowią tu klasy systemowe, które nie są deklarowane na poziomie deklaracji bloku prefiksowanego

czyć, że w klasie prefiksującej blok nie może wystąpić wyrażenie this <nazwa tej klasy>. Rozważmy przykład:

```
● begin
    class A;
    begin
      procedure α ;
      class B;
      class C;
    C class D;
    end;
    A begin
        B class Z;
        D class U;
        ..........
      end;
  end;
```

blok prefiksowany

W bloku prefiksowanym klasą A wykorzystano atrybuty tej klasy tzn. klasy B i D, do prefiksowania klas Z i U zadeklarowanych w tym bloku (rys. 22).



Rys. 22

Z chwilą utworzenia jednostki dynamicznej obiektu złożonego (po wejściu do bloku), działa mechanizm złożenia bloku z klasą prefiksującą ten blok. Rozpoczyna się wykonywanie pierwszych instrukcji na zasadach podanych przy omawianiu konkatenacji klas, tzn. wykonuje się instrukcje klasy prefiksującej z instrukcjami bloku wstawionymi w miejsce symbolu inner. W szczególnym wypadku treść klasy prefiksującej blok może zawierać tylko strukturę danych. Wtedy wykonuje się od razu program bloku prefiksowanego tą klasą.

Przejdźmy teraz do następnego przykładu.

```
begin
  class struktura_stanowisk;
  struktura_stanowisk class gniazdo_produkcyjne;
  gniazdo_produkcyjne begin
                        ......
                      end;
end;
```

W Simuli istnieje możliwość prefiksowania bloku dowolną klasą zadeklarowaną na poziomie deklaracji bloku. W szczególności klasa ta może być prefiksowana inną klasą. Ilustruje to powyższy przykład. Można więc tworzyć dowolne hierarchiczne struktury danych. Własność tę wykorzystamy w kolejnym przykładzie.

```
● class Simula;
  <definicje wszystkich atrybutów Algolu i Simuli>
● Simula class Simset;
  <definicje operacji na strukturach listowych>
● Simset class Simulation;
  <definicje procedur umożliwiających symulację> ;
```

W kolejnych punktach przedstawiono deklaracje klas, w których zdefiniowano atrybuty ukierunkowane na określone zastosowania. Przyjmijmy, że powyższe deklaracje stanowią pakiety programów do wykorzystania przez użytkownika.

Jeżeli chcemy wykorzystywać w naszym programie podstawowe pojęcia Simuli i Algolu, np. do obliczeń inżynierskich, wtedy blok programu prefiksujemy klasą Simula, tzn.

```
Simula begin
        <program użytkownika>
      end;
```

Jeżeli zachodzi potrzeba tworzenia struktur listowych, np. banki danych to wygodnie jest blok programu prefiksować klasą Simset, tzn.

```
Simset begin
        <program użytkownika>
      end;
```

Jeżeli chcemy przeprowadzać eksperymenty symulacyjne to napiszemy:

```
Simulation begin
             <program użytkownika>
           end;
```

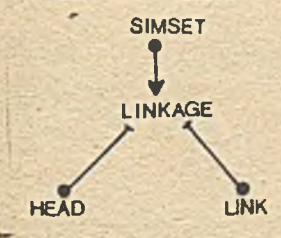
Klasy prefiksujące blok mogą być definiowane systemowo. Może je także definiować sam użytkownik. Chcąc na przykład symulować system kontroli procesu produkcyjnego, można klasę "symulacja␣procesu␣produkcyjnego", której treść tworzy użytkownik prefiksować klasą Simulation, a następnie tę pierwszą prefiksować blok programu, tzn.

```
begin
  Simulation class symulacja␣procesu␣produkcyjnego;
              begin
                <deklaracje atrybutów opisujących symulację procesu produkcyjnego>;
              end;
  symulacja␣procesu␣produkcyjnego begin
                                    <program użytkownika>
                                  end;
end; $ programu $.
```

Klasa Simset

W Simuli 67 istnieje możliwość operowania na zbiorach implementowanych za pomocą struktur listowych dzięki systemowej klasie Simset. Wprowadzono tu listy dwukierunkowe oraz mechanizmy działania na nich. Każdy element listy ma swojego poprzednika i następnika. A zatem obiekty będące listami bądź ich elementami mają zawsze referencję do swojego poprzednika (PRED) i następnika (SUC). W klasie Simset, wykorzystując możliwość prefiksowania, zdefiniowano 3 klasy, tzn.

```
class Simset;
begin
  class linkage...;
  linkage class head...;
  linkage class link...;
end;
```



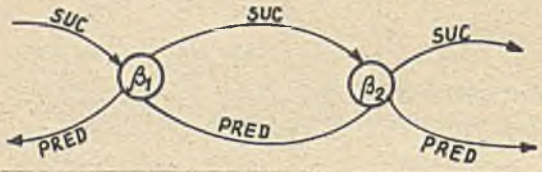
Rys. 23

Ilustruje to rys.23.
Zbiory są tu reprezentowane przez obiekty klasy head, natomiast ich elementy przez obiekty klasy link. Prefiksowanie bloku programu klasą Simset umożliwia korzystanie ze wszystkich pojęć w niej zdefiniowanych.

Klasa linkage

```
Definicja: class linkage;
           begin
             ref(linkage) SUC,PRED;*)
             ref(link)procedure suc;
             suc :- if SUC in link then SUC else none;
             ref(link)procedure pred;
             pred :- if PRED in link then PRED else none;
           end;
```

W klasie linkage zadeklarowano atrybuty wspólne dla klas link i head. Zmienna referencyjna SUC daje dostęp do następnika obiektu klasy linkage, PRED - do jego poprzednika. Ilustruje to rys. 24 przy czym β_1, β_2 są obiektami klasy linkage.



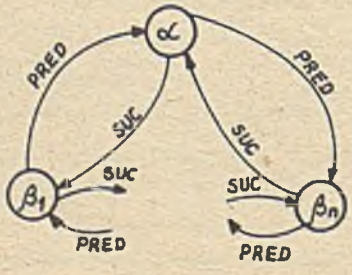
Rys. 24

Procedury suc i pred umożliwiają dostęp do atrybutów SUC i PRED. Wartości atrybutów SUC i PRED mogą być zmieniane tylko przez użycie procedur zadeklarowanych w klasach link i head.

Klasa link

```
Definicja: linkage class link;
           begin
             procedure out...;
             procedure precede...;
             procedure follow...;
             procedure into...;
           end;
```

Obiekty należące do klasy link lub jej podklasy mogą być elementami zbioru. W danej chwili obiekt może być elementem tylko jednego zbioru. A zatem obiekty klasy link, np. β_1, β_2 *) można ustawiać w kolejki wiążąc je w ten sposób, aby atrybut SUC obiektu β_1 dostarczał referencję do obiektu β_2, a atrybut PRED obiektu β_2 dostarczał referencję do obiektu β_1 (rys. 24). Kolejka α reprezentowana jest przez obiekt klasy head (pkt "Klasa head"). Rys. 25 ilustruje obiekty β_1 klasy link w kolejce α. Kolejno omówimy teraz procedury zadeklarowane w klasie link.



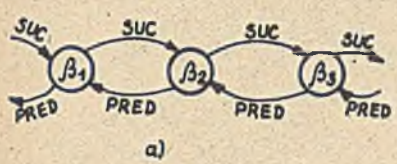
Rys. 25

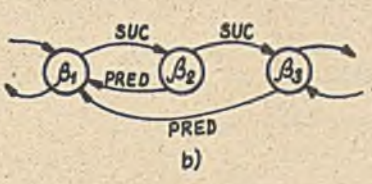
- procedure out;
  if SUC =/= none then
  begin
    SUC.PRED :- PRED;
    PRED.SUC :- SUC;
    SUC :- PRED :- none;
  end;

*) nazwy pisane dużymi literami są niedostępne dla programisty

**) wyznaczone przez zmienne β_1, β_2 typu ref (link)

Procedura out usuwa obiekt ze zbioru. Kolejne fazy usuwania obiektu β_2 wg procedury out ilustruje rys. 26 (a,b,c,d).



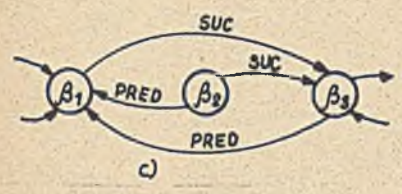


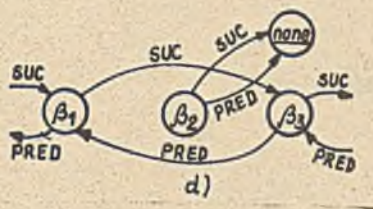
a) Konfiguracja początkowa ref(link)$\beta_1, \beta_2, \beta_3; \beta_2$.out*)

b) SUC.PRED:-PRED

c) PRED.SUC:-SUC

d) SUC:-PRED:-none



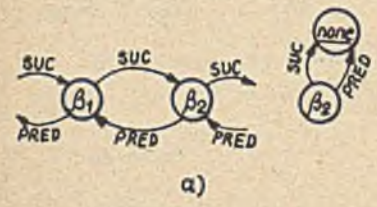


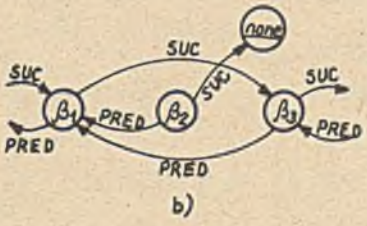
Rys. 26

Jeżeli obiekt nie należy do żadnego zbioru, to działanie procedury out nie daje żadnego efektu.

```
procedure follow(x);
ref(linkage) x;
begin   out;
  if x=/= none then
  begin
    if x.SUC=/= none then
    begin
      PRED :- x;
      SUC :- x.SUC;
      SUC.PRED :- x.SUC :- this linkage;
    end;
  end;
end;
```

Procedura follow (x) powoduje wstawienie danego obiektu klasy lub podklasy klasy link za obiekt x.Rys.27 ilustruje kolejne fazy wstawiania obiektu β_2 za obiekt β_1. Zwróćmy uwagę,



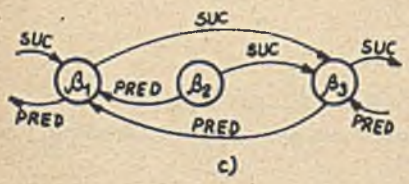


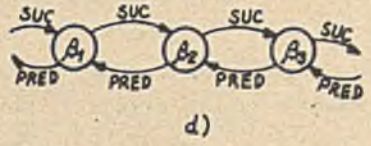
a) Konfiguracja początkowa ref(link)$\beta_1, \beta_2, \beta_3; \beta_2$.follow($\beta_1$);

b) PRED:-β_1

c) SUC:-β_1.SUC

d) SUC.PRED:-β_1.SUC:-this linkage;(atrybuty β_1.SUC i β_3.PRED odnoszą się do obiektu β_2)





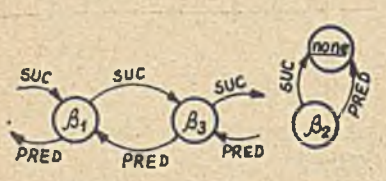
Rys. 27

*) Po wywołaniu β_2.out atrybuty PRED i SUC w procedurze out odnoszą się do obiektu β_2. Uwaga ta dotyczy także pozostałych procedur.

że procedura follow(x) przed wstawieniem obiektu danego za obiekt x, usuwa ten pierwszy ze zbioru, w którym się znajduje.

```
• procedure precede(x);
  ref(linkage)x;
  begin  out;
    if x=/= none then
    begin
      if x.SUC =/= none then
      begin
        SUC :- x;
        PRED :- x.PRED;
        PRED.SUC :- x.PRED :- this linkage;
      end;
    end;
  end;
```

Procedura precede(x) wstawia dany obiekt klasy link lub jej podklasy przed obiekt x, po uprzednim usunięciu tego pierwszego ze zbioru, w którym może się on znajdować. Np. instrukcja β_2. precede(β_3) powoduje wstawienie obiektu β_2 przed obiekt β_3. Graficzną ilustrację działania procedury precede przedstawia rys. 28 (a,b,c,d).



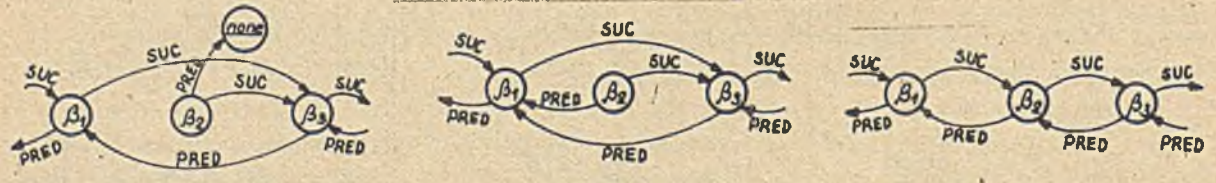
a)

a) Konfiguracja początkowa
ref(link) $\beta_1, \beta_2, \beta_3; \beta_2$.precede($\beta_3$)

b) SUC:-β_3

c) PRED:-β_3.PRED

d) PRED.SUC:-β_3.PRED:-
this linkage

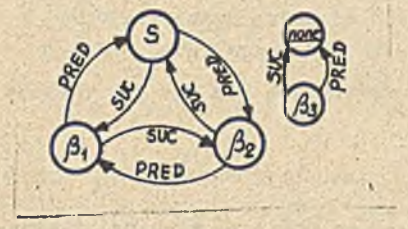


b) c) d)

Rys. 28

```
• procedure into(S);
  ref(head)S;
  begin
    precede (S);
  end;
```

Procedura into(S) wstawia dany obiekt do zbioru S (na ostatnim miejscu). Np. instrukcja β_n.into(S) spowoduje wstawienie obiektu β_n do zbioru S. Działanie procedury into ilustruje rys. 29.



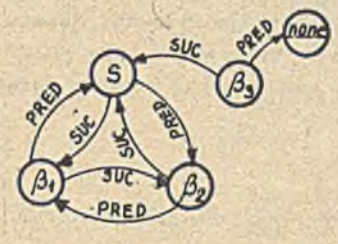
a)

a) Konfiguracja początkowa
ref(link) $\beta_1, \beta_2, \beta_3$;
ref(head) S; β_3.into(S)

b) SUC:-S

c) PRED:-S.PRED

d) PRED.SUC:-S.PRED:-this linkage



b)

c)

d)

Rys. 29

Uwaga: Wywołanie procedur out , precede , follow i into w treści obiektu klasy link lub jej podklasy przy założeniu, że procedury te dotyczą tego obiektu jest bezpośrednie, np.

```
link class wiadomość;
begin
  ref(head)pamięć;
  pamięć :- new head;
  into(pamięć);
end;
```

Instrukcja new wiadomość spowoduje wstawienie obiektu klasy wiadomość do zbioru pamięć (zbiór pamięć należy uprzednio wygenerować np. pamięć :- new head).

Klasa head

```
Definicja: linkage class head;
           begin
             ref(link)procedure first: first :- suc;
             ref(link)procedure last; last :- pred;
             boolean procedure empty;
             empty := SUC == this linkage;
             integer procedure cardinal;
             begin
               integer i; ref(linkage) x;
               x :- this linkage;
               for x:-x.suc while x =/= none do
               i := i + 1;
               cardinal := i;
             end;
```

```
    procedure clear;
    begin
      ref(linkage)x;
      for x:=first while x =/= none do
      x qua link.out;
    end;
    SUC :- PRED :- this linkage;
  end;
```

Obiekty klasy head lub jej podklasy służą do opisu zbiorów, przy czym same nie mogą być elementami innych zbiorów. Każdy obiekt klasy head ma tzw. "głowę" (head). Podczas generacji obiektu α klasy head jego atrybutom SUC i PRED przypisane zostają wartości stanowiące referencję do niego samego (rys. 30).

Rys. 30

Obiekt klasy head α reprezentujący kolejkę wraz za znajdującymi się w niej obiektami klasy link ilustruje rys. 25. Obiekt klasy link można umieścić w zbiorze α tylko wtedy, gdy zbiór ten został wygenerowany (instrukcja new head) . Omówimy teraz procedury zdefiniowane w klasie head:

- procedura first daje referencję do pierwszego elementu zbioru, np. x.first, gdzie x jest nazwą zbioru (ref(head)x),
- procedura last daje referencję do ostatniego elementu zbioru x, np. x.last,
- procedura empty daje wartość true, jeśli zbiór jest pusty, np. while not x.empty do ....
- procedura clear usuwa wszystkie elementy zbioru, np. x.clear,
- procedura cardinal wyznacza liczbę elementów zbioru (długość kolejki), np. if x.cardinal < n then .....

Przykłady

Rozważmy prosty przykład ilustrujący rysunkiem efekt działania każdej instrukcji:

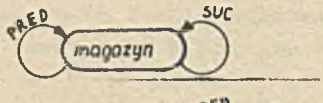
```
begin
  ref(head)magazyn;
  ref(wyrób)W1,W2;
  link class wyrób;
  ..............
  magazyn:-new head;

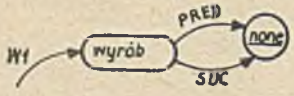
  W1 :- new wyrób;

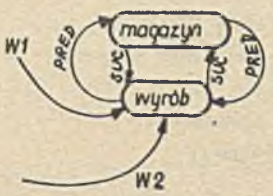

  W1.into(magazyn);

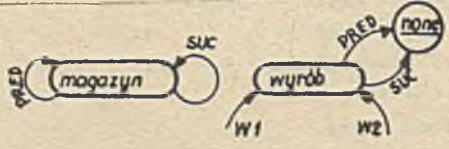

  W2 :- magazyn.first;
  $ magazyn.cardinal=1 $


  magazyn.clear;
end;
```









Rozwiążmy teraz następujący problem.

- Napisać program wstawiający wyroby do magazynu a następnie przenoszący je do czterech pojemników. Każdy pojemnik składa się z dwóch części przeznaczonych na składowanie wyrobów z kodem parzystym i nieparzystym.

Przedstawmy magazyn jako zbiór (obiekt klasy head), tzn.

head class magazyn;

Wyroby są rozróżniane poprzez kod. Można je przedstawić jako elementy magazynu, tzn. obiekty klasy link mające atrybut 'parametr'. Należy ponadto zdefiniować klasę pojemnik. Pojemnik składa się z dwóch zbiorów (obiektów klasy head), z których każdy zawiera wyroby z odpowiednim kodem. Napiszmy pierwszą wersję programu. Chcąc wykorzystać w nim pojęcia klasy SIMSET musimy blok programu prefiksować tą klasą.

```
Simset begin
        head class rodzaj_wyrobu(kod); integer kod;
        head class magazyn;
        link class wyrób(parametr); integer parametr;
        class pojemnik(i); integer i;
        begin
          ref(rodzaj_wyrobu)array H(1:2);
          integer j;
          for j:=1 step 1 until 2 do
          H(j) :- new rodzaj_wyrobu(j);
        end;
        <procedury generowania wyrobów, magazynu i przenoszenia wyrobów
         do magazynu> ;
      end;
```

Pozostało nam jeszcze zdefiniowanie procedur wybierania wyrobów z magazynu i wstawiania ich do odpowiednich pojemników. Procedura wybierania wyrobów wyb_wyrób(i), gdzie i jest bieżącą liczbą wyrobów w magazynie, losując liczbę p z przedziału [1,i] zgodnie z rozkładem równomiernym, z wyrobów ułożonych w kolejności od 1 do i, wybiera ten, który znajduje się na pozycji p. Procedura wstawiania wyrobu do pojemnika wstaw_wyrób musi mieć jako parametr formalny referencję do odpowiedniego pojemnika. W ten sposób uzyskuje się dostęp do obu części H(i) pojemnika, przy czym wyrób zgodnie z kodem wstawiany jest do jednej z nich. Pełną treść programu z deklaracjami powyższych procedur oraz licznymi komentarzami podano poniżej.

```
Simset begin
        head class rodzaj_wyrobu(kod); integer kod;
        head class magazyn;
        begin integer r;
          ref(wyrób)procedure wyb_wyrób(i); integer i;
          begin
            ref(wyrób) x; integer j,p;
            p:=randint(1,i,U);  *)
            $ losowanie pozycji wyrobu w magazynie $
            x :- first;
            $ x wskazuje na pierwszy wyrób w magazynie $
            for j:=2 step 1 until p do
            x :- x.suc;
            $ x wskazuje na p-ty wyrób w magazynie $
            wyb_wyrób :- x;
          end;
          for r:=1 step 1 until 100 do
          new wyrób(randint(1,2,U)) .into(this head);
          $ generacja wyrobów z kodami i wstawienie ich do magazynu $
        end;
```

*) procedura randint(A,B,U) losuje liczby całkowitą z przedziału A,B zgodnie z rozkładem równomiernym

```
link class wyrób(parametr); integer parametr;
begin
  procedure wstaw_wyrób(P); ref(pojemnik)P;
  begin
    ref(rodzaj_wyrobu)S;
    S :- P.H(parametr);
    $ S wskazuje na jedną z dwóch części pojemnika $
    into(S);
    $ wstawienie wyrobu do odpowiedniej części pojemnika $
  end;
end;
class pojemnik(i); integer i;
begin  integer j;
  ref(rodzaj_wyrobu)array H(1:2);
  for j:=1 step 1 until 2 do
  H(j) :- new rodzaj_wyrobu(j);
  $ utworzenie dwóch części pojemnika H(1), H(2) $
end;
ref(pojemnik)array poj(1:4);
ref(wyrób)wyr;
ref(magazyn)mag;
integer i,j;
for i :=1 step 1 until 4 do
poj(i) :- new pojemnik(i);
$ utworzenie pojemników $
mag :- new magazyn;
$ utworzenie magazynu $
for i:=100 step -1 until 1 do
begin
  wyr :- mag. wyb_wyrób(i);
  $ wybranie wyrobu $
  if j=4 then j:=1 else j:=j+1;
  $ zmiana numeru pojemnika $
  wyr. wstaw_wyrób(poj(j)) ;
  $ wstawienie wyrobu do kolejnego pojemnika $
end;
end; $ programu $
```

Należy zaznaczyć, że przedstawiony powyżej program stanowi jedną z wielu możliwości opisu i definicji poszczególnych obiektów. Można go także rozszerzyć o bardziej skomplikowane regulaminy obsługi, zbliżone do sytuacji rzeczywistych.

W następnej części będą podane możliwości Simuli w zakresie operacji na tekstach, generowania liczb pseudolosowych, a także przedstawiona będzie kolejna klasa systemowa BASICIO do wprowadzania i wyprowadzania danych.

Literatura

[1] Birthwistle G. i in.: Notes on the simula Language. Publication No.S-7 NCC Forskningsveien 1 B, Oslo 1969

[2] Birthwistle G. i in.: Common Base Language. Simula Information. Publication No. S-22 NCC Forskningsveien 1 B. Oslo 1970

[3] Birthwistle G. i in.: Simula Begin. Auerbach Publishers Inc. Phipadelphia, Pa, 1973

[4] Simula sous SIRIS7/SIRIS8 manuel d'utillisation. T 1. CII, 1973

[5] Iohbiah J.D., Morse S.: General Concepts of the Simula 67. Programming Language. Compagnie Internationale pour l'Informatique, Les Clayes Sous Bois, France.

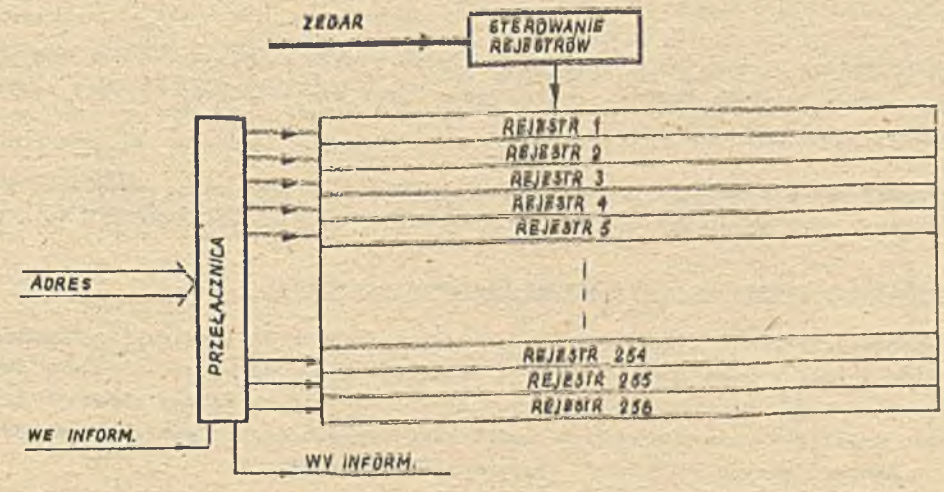
mgr inż. Jan WRONA
Biuro Generalnych Dostaw CNPTKiP

Zastosowanie rejestrów dynamicznych w pamięciach wewnętrznych z bezpośrednim dostępem w minikomputerach

Spośród produkowanych pamięci półprzewodnikowych rejestry dynamiczne należą obecnie do układów o największej pojemności w jednym układzie scalonym i najniższej cenie za jeden bit. Znalazły one zastosowanie jako wewnętrzne odpowiedniki pamięci wirujących z blokowym przesyłaniem informacji między pamięcią wirującą a pamięcią operacyjną. W artykule przedstawiono możliwości stosowania w minikomputerach rejestrów dynamicznych jako pamięci wewnętrznych z bezpośrednim dostępem.

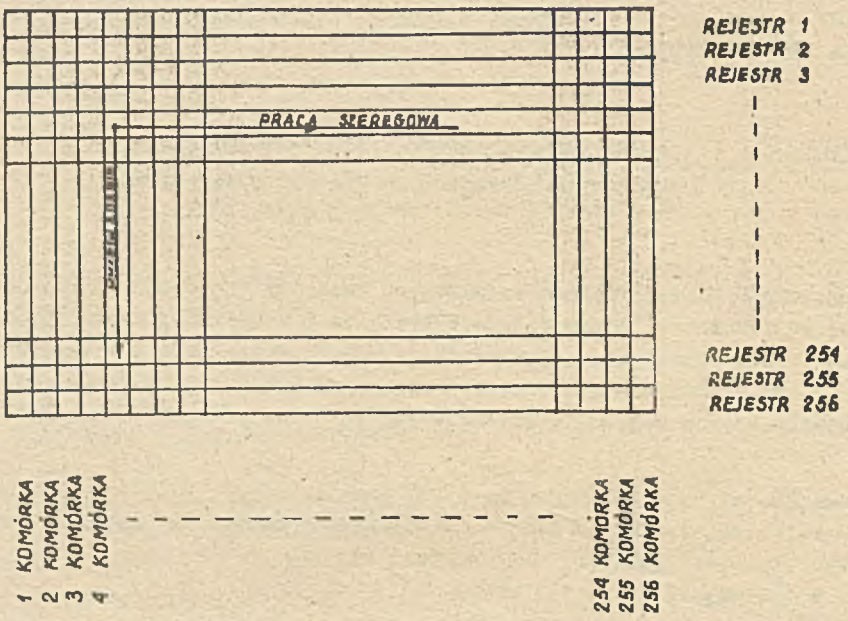
Rejestry dynamiczne są tak zbudowane, że przez podanie adresu z zewnątrz można tylko wybierać poszczególne rejestry, dostęp do komórek w rejestrach odbywa się kolejno, przez podawanie impulsów zegarowych. Na rys. 1 pokazany jest schemat blokowy 64 k-bitowego elementu pamięciowego składającego się z 256 rejestrów po 256 bitów.

Tego typu układy pamięciowe wymagają aby co pewien czas (którego dolna granica jest ściśle określona), nastąpiło odnowienie zawartości komórek pamiętających. Układy nie posiadające wewnętrznego odnawiania wymagają, aby zawartość każdego z rejestrów została odnowiona osobno, przez dostęp do dowolnej komórki w nim.



Rys. 1. Schemat blokowy 64 k-bitowego elementu pamięciowego składającego się z 256 rejestrów po 256 bitów

Typowe układy dopuszczają dwa sposoby pracy: szeregową i przeglądanie. Ilustruje je schematycznie rysunek 2.



Rys.2. Schemat dwóch rodzajów pracy rejestrowej pamięci dynamicznej

Przy przeglądaniu dostęp do komórek o tym samym numerze w poszczególnych rejestrach odbywa się przez zmianę adresu rejestrów; przy pracy szeregowej przy niezmienionym adresie rejestru ma się dostęp do kolejnych w nim komórek.

Warunkiem poprawnej pracy przedstawionych poniżej układów jest napędzanie procesora i pamięci ze wspólnego zegara, przy czym cykl podstawowy procesora i pamięci powinien być taki sam.

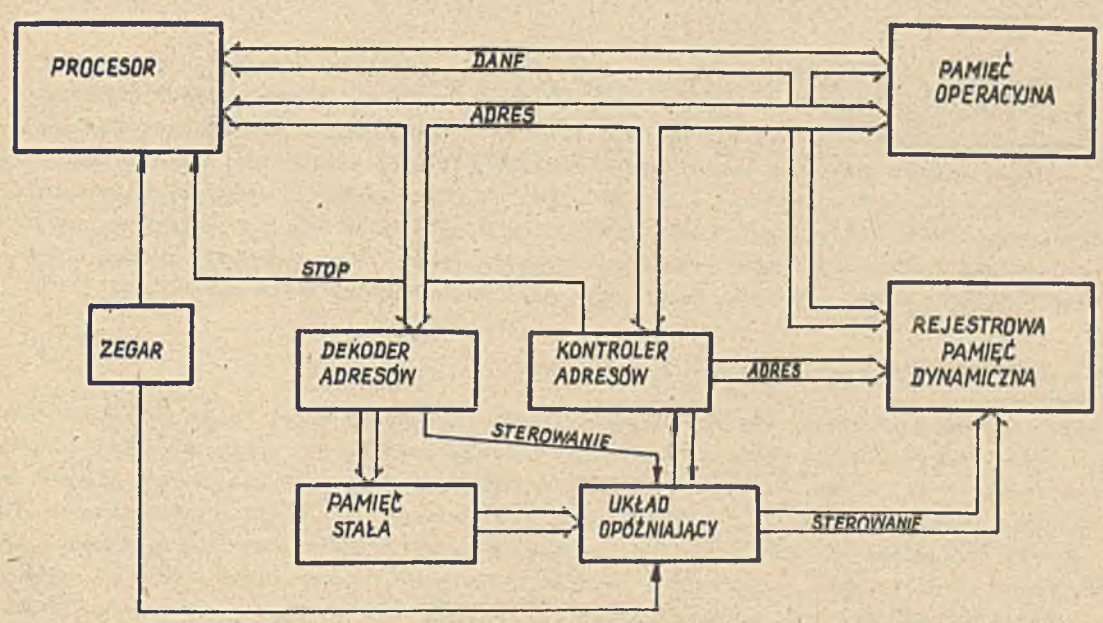
Synchroniczna komunikacja procesora z rejestrową pamięcią dynamiczną

Rejestry dynamiczne przy pracy szeregowej dopuszczają zmianę okresu zegara traktującego w pewnych granicach (przeciętnie w granicach do 1:100). Niektóre układy dopuszczają zmianę okresu w trzech taktach, podczas gdy czwarty powinien mieć stały i ściśle określony okres; w tym wypadku należy sztucznie umożliwić zmianę okresu we wszystkich taktach zegarowych, wprowadzając odpowiedni rejestr, do którego wcześniej wpisuje się zawartość pamięci.

Właściwość zmiany okresu zegara traktującego została wykorzystana w przedstawionym układzie. Założeniem jest, aby realizacja mikroprogramu przesłania do lub pobrania z pamięci trwała w czasie jednego taktu zegarowego pamięci. Pozwala to na synchroniczną komunikację z kolejnymi komórkami pamięci. Oczywiście różne mikroprogramy wymagają różnych czasów ich realizacji.

Układając mikroprogramy należy pamiętać, żeby stosować rozkazy mające stały czas ich wykonania oraz, aby czas realizacji mikroprogramu nie przekraczał dopuszczalnego czasu trwania jednego taktu zegara pamięci.

Schemat układu realizującego pracę synchroniczną pokazany jest na rys. 3.

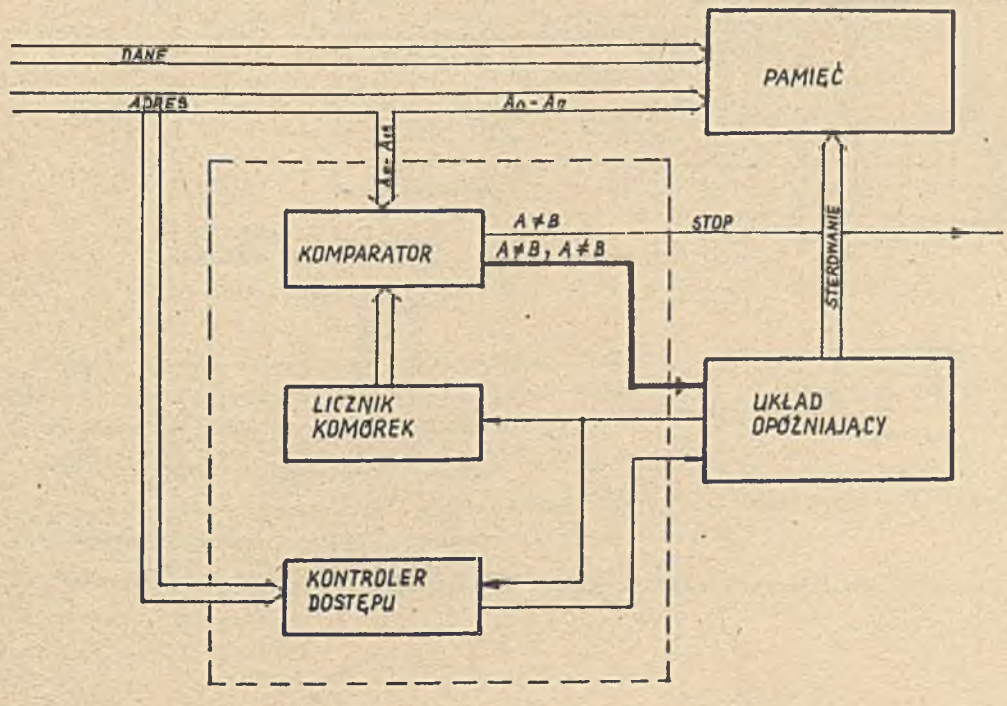


Rys. 3. Schemat układu realizującego pracę synchroniczną

Praca układu jest następująca: Adres pobieranego rozkazu przez PROCESOR z PAMIĘCI OPERACYJNEJ podawany jest na DEKODER ADRESÓW. Każdemu adresowi pierwszego rozkazu mikroprogramu przypisana jest komórka w PAMIĘCI STAŁEJ, w której zapisana jest liczba taktów zegara potrzebnych do wykonania mikroprogramu. Zawartość tej komórki podawana jest do UKŁADU OPÓŹNIENIA, który powoduje, że czas taktu zegarowego pamięci trwa przez podaną z PAMIĘCI STAŁEJ liczbę taktów zegara.

Specjalną rolę odgrywa w układzie KONTROLER ADRESÓW. Sprawdza on czy adres wysłany przez PROCESOR jest adresem kolejnej komórki w pamięci. Jeżeli nie, wysyła sygnał STOP podający wstrzymanie pracy PROCESORA do czasu znalezienia zaadresowanej komórki i powoduje wyszukiwanie jej z maksymalną szybkością.

Pracę KONTROLERA ADRESÓW ilustruje schemat blokowy na rys. 4.



Rys. 4. Schemat blokowy ilustrujący pracę KONTROLERA ADRESÓW

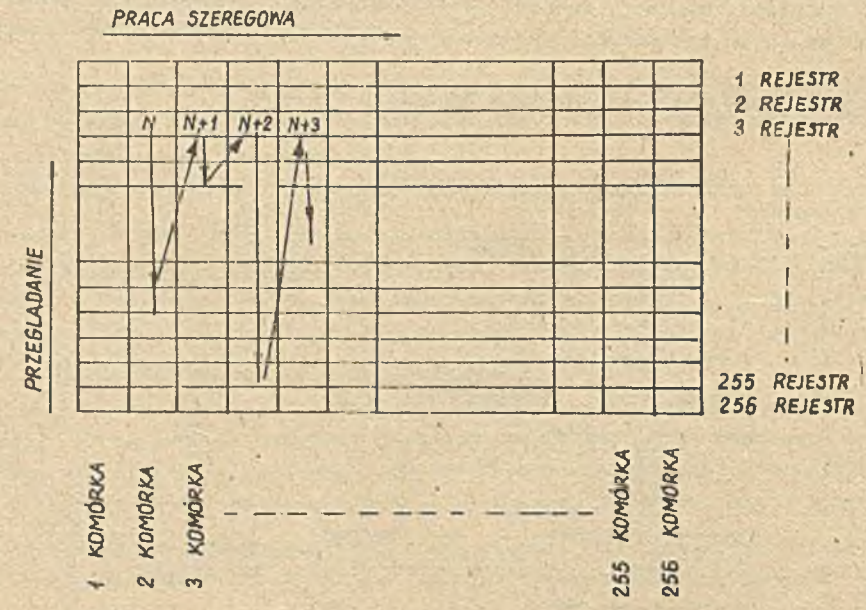
Jak wiadomo, przez podanie adresu wybierane są tylko poszczególne rejestry, zaś komórki w rejestrze są wybierane kolejno, przez podawanie do pamięci impulsów zegarowych. LICZNIK KOMÓREK zliczający takty zegara określa wybraną komórkę w rejestrze. KOMPARATOR stwierdza, czy młodsza część adresu jest taka sama, jak w LICZNIKU KOMÓREK. Jeżeli tak, zezwala na dostęp do PAMIĘCI, jeżeli nie, wysyła sygnał STOP do PROCESORA i sygnał włączenia maksymalnej szybkości wybierania komórek do UKŁADU OPÓŹNIAJĄCEGO. KONTROLER DOSTĘPU sprawdza, czy do wybranej komórki PAMIĘCI było odwołanie się PROCESORA, jeżeli nie, powoduje wysłanie sygnału do UKŁADU OPÓŹNIAJĄCEGO , powodującego wybieranie komórek z normalną szybkością.

Budowa pamięci operacyjnej na rejestrach dynamicznych

Od pamięci operacyjnej wymaga się szybkiego dostępu do jej zawartości; szybki dostęp bowiem decyduje w dużej mierze o wykorzystaniu komputera. Stosując rejestry dynamiczne należy przewidzieć takie rozwiązanie, które pozwoli skrócić do minimum czas wyszukiwania poszczególnych komórek. Pierwszym warunkiem jest, żeby pamięć operacyjna składała się z kilku niezależnych bloków, np. jeżeli przyjmiemy, że wykonujemy obliczenia to wskazane jest żeby były w trzech blokach. Taka organizacja pozwoli na uniknięcie skoków związanych z pobieraniem danych i zapamiętaniem wyników obliczeń przy założeniu, że praca bloków jest synchroniczna, to znaczy, że kolejne wybieranie komórek nie wymaga wyszukiwania ich w rejestrach.

Jedyną możliwością takiej realizacji pamięci jest wykorzystanie obu rodzajów pracy: szeregowej i przeglądania. Dostęp do pamięci odbywa się w reżimie pracy szeregowej, zaś - jeżeli tak to można określić - wytracanie czasu między kolejnymi odwołaniami dla uzyskania pracy synchronicznej, odbywa się w reżimie przeglądania.

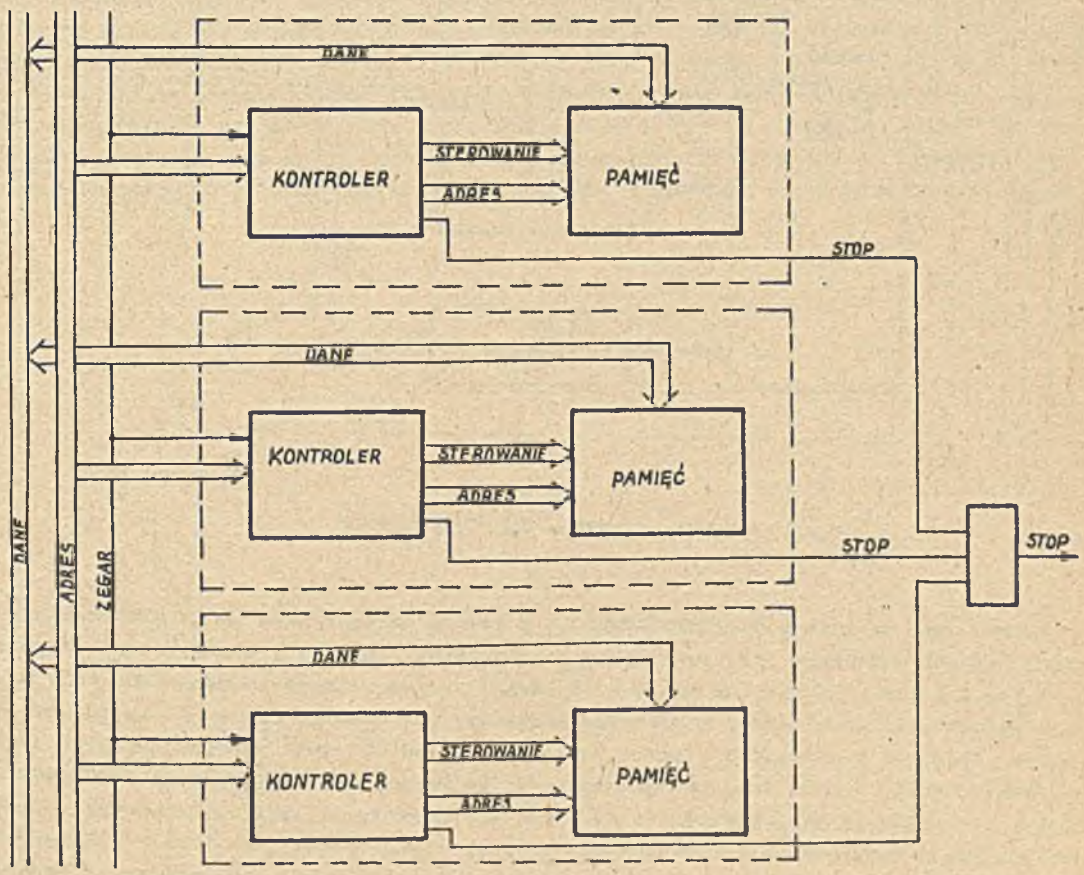
Schematycznie połączenia obu rodzajów pracy pokazane jest na rys. 5.



Rys. 5. Schemat połączenia w reżimie pracy szeregowej i w reżimie przeglądania

Procesor wystawia adres komórki o numerze N, po realizacji pobrania pamięć przechodzi do przeglądania komórek o tym samym numerze w rejestrach, gdy przychodzi adres komórki N +1 przeglądanie jest przerwane, po realizacji pobrania następuje przejście do przeglądania komórek o numerze N+1 i tak dalej.

Schemat blokowy pamięci zbudowanej z trzech bloków pokazany jest na rys. 6.

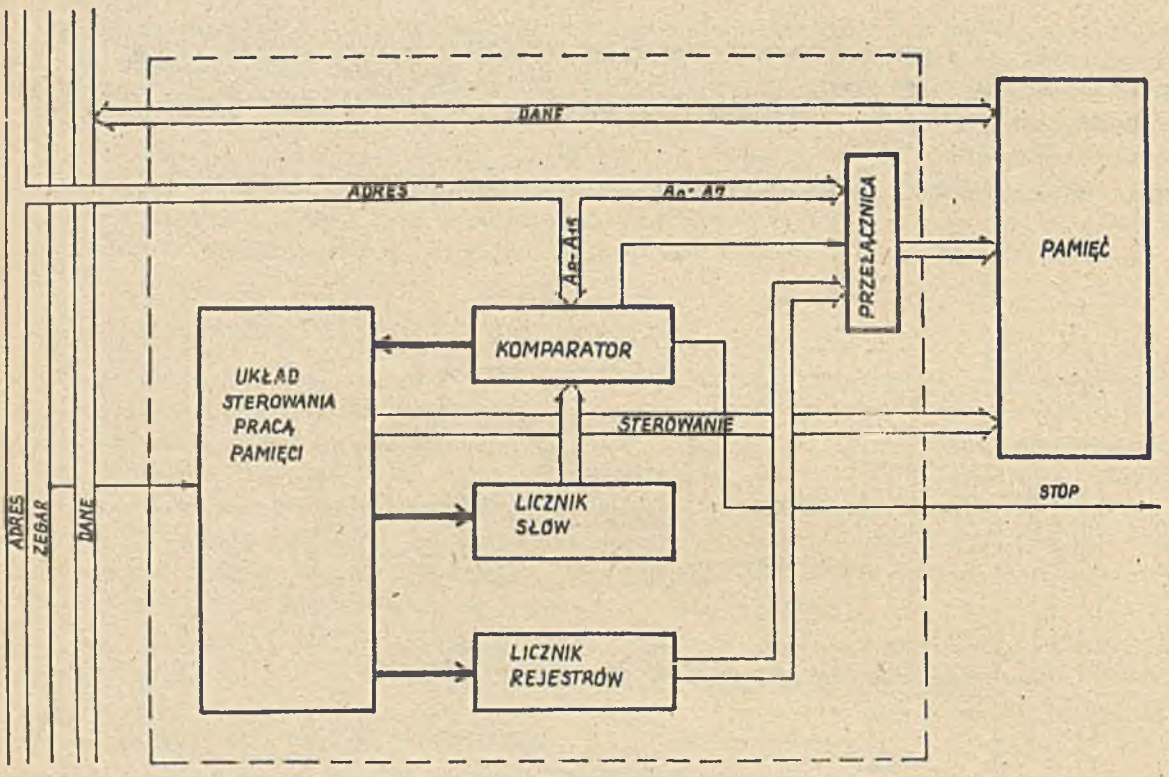


Rys. 6. Schemat blokowy pamięci operacyjnej zbudowanej z trzech bloków

Każdy z modułów zaopatrzony jest w UKŁAD KONTROLI, którego zadaniem jest wybieranie odpowiednich reżimów pracy, a także wytwarzanie sygnału STOP do PROCESORA wstrzymującego jego pracę, do czasu wyszukania zaadresowanej komórki przy skokach.

Schemat skokowy UKŁADU KONTROLI pokazany jest na rys. 7.

Można oczywiście przewidzieć jeden wspólny kontroler, oszczędności jednak będą niewielkie, zaś komplikuje to możliwość dołączania kolejnych modułów.



Rys. 7. Schemat skokowy UKŁADU KONTROLI

Podstawowym układem w UKŁADZIE KONTROLI w module pamięci jest UKŁAD STEROWANIA PRACĄ PAMIĘCI, zadaniem którego jest wybieranie rodzaju pracy pamięci przez wysyłanie do niej odpowiednich impulsów. Napędzany on jest impulsami z zegara napędzającego PROCESOR. Gdy z procesora podany jest nowy adres, KOMPARATOR sprawdza, czy młodsza część adresu (odpowiadająca numerowi komórki w rejestrze) jest o jeden większa od ostatnio wybranej komórki zapisanej w LICZNIKU SŁÓW. Jeżeli tak, specjalnym sygnałem zezwala na przejście UKŁADOWI KONTROLI na jeden takt zegarowy od przeglądania do pracy szeregowej, co jest równoznaczne z umożliwieniem dostępu przez PROCESOR do zaadresowanej komórki. Jeżeli adres jest inny - KOMPARATOR wstrzymuje przeszukiwanie, powodując wybranie zaadresowanej komórki w pamięci przez podawanie impulsów zegarowych. Na czas jej wyszukiwania KOMPARATOR podaje sygnał STOP do PROCESORA. W czasie przeszukiwania UKŁAD STEROWANIA PRACĄ PAMIĘCI podaje impulsy do LICZNIKA REJESTRÓW, którego stan jest podawana do PAMIĘCI jako adres kolejnych rejestrów przy przeglądaniu.

Uwagi końcowe

Przedstawione rozwiązania organizacji pamięci opracowano z uwzględnieniem jak najszybszej współpracy procesora z pamięciami zbudowanymi na rejestrowych pamięciach dynamicznych. Organizacja modułów przedstawionych w drugim przykładzie pozwala na stosowanie ich także jako pamięci wewnętrznych. Jednak dla tego zastosowania, lepsze wydaje się pierwsze rozwiązanie. Pomijając większą prostotę konstrukcyjną, umożliwia ono dostęp do pamięci wyłącznie z mikroprogramów; ma to tę zaletę, że eliminuje niepożądany dostęp do pamięci.

Zastosowanie rejestrowych pamięci dynamicznych w pamięci operacyjnej musi prowadzić do spowolnienia pracy minikomputera - spowodowanej skokami. Przyjęcie jednak pracy kilkoblokowej

zmniejsza ich liczbę. Oczywiście zupełnie nie da się skoków uniknąć. Jeżeli przyjmiemy np., że jeden skok programowy przypada na dwadzieścia rozkazów, średni czas wykonania rozkazu wynosi 5 μs, zaś średni czas wyszukania komórki w rejestrze 100 μs (dla typowych układów), to minikomputer z taką pamięcią będzie pracował dwa razy wolniej.

Pamięć składająca się z czterech bloków po 64 k słów 16 bitowych (zbudowana na układach o pojemności 64 k bitów), powinna się zmieścić na jednym pakiecie. Jej koszt (przy aktualnych cenach) powinien być nie większy niż koszt zbudowanej na układach RAM pamięci o pojemności maksimum kilkudziesięciu k słów.

Stosowanie rejestrów dynamicznych w pamięciach wewnętrznych minikomputerów pozwala na tanią budowę niekonwencjonalnych systemów z dużymi pamięciami. Niezbędne to jest np. w systemach sterowania procesami wytwarzania, przy konieczności instalowania komputera w halach produkcyjnych w warunkach uniemożliwiających instalację pamięci zewnętrznych na nośnikach magnetycznych. Wymaga to zwiększenia pamięci dla zainstalowania programów sterujących procesami i programów wykonawczych dla automatów, przy czym czasy dostępu do pamięci w takich systemach nie są krytyczne. Drugim przykładem zastosowania mogą być systemy pomiarowe, wymagające szybkiego zbierania dużej ilości danych w warunkach uniemożliwiających stosowanie pamięci zewnętrznych.

dr inż. Ryszard PAWLIK
Instytut Maszyn Matematycznych

z cyklu: Komputeryzacja programowania

Systemy metody elementu skończonego - projekt systemu wzorcowego

Wstęp

W nr 5/6 z 1977 r. Biuletynu Informacyjnego OSK opublikowano dwa artykuły pod wspólnym tytułem "Systemy metody elementu skończonego", w których w wielkim skrócie zaznajomiono czytelników z problematyką obliczeń wytrzymałościowych metodą elementu skończonego. W jednym z nich [3] omówiono eksploatowane w kraju systemy metody elementu skończonego oraz naszkicowano koncepcję systemu wzorcowego, który pozbawiony byłby wad i braków systemów już eksploatowanych.

Prezentowany artykuł stanowi kontynuację cyklu "Systemy metody elementu skończonego" i przedstawia projekt realizacji systemu wzorcowego. Podstawowym celem podjęcia pracy nad takim systemem jest stworzenie warunków dla usprawnienia procesu projektowania wytrzymałościowego konstrukcji. Ten zasadniczy cel można rozbić na dwa główne zadania, których rozwiązanie pozwoli osiągnąć założony cel:

- minimalizacja czasu, który upływa od momentu sformułowania zadania do momentu uzyskania efektywnych wyników,
- minimalizacja nakładu pracy niezbędnej dla osiągnięcia poprawnych wyników.

Zakłada się przy tym, że usprawniany proces dotyczy projektowania wytrzymałościowego minimum tak szerokiej klasy zadań, jaką obejmują istniejące w kraju programy, czy systemy przeznaczone do obliczeń wytrzymałościowych.

Z przeprowadzonej analizy rozwiązywanych zadań wynika, że wiele z nich można z zadowalającą dokładnością rozwiązywać metodami klasycznymi, przy czym do metod klasycznych zaliczamy metody, w których wykorzystuje się dokładne lub przybliżone, lecz analityczne wzory na obliczanie naprężeń, odkształceń i innych interesujących projektanta wielkości charakteryzujących projektowaną konstrukcję. Z tego względu do projektu systemu włączono część opartą na metodach klasycznych. Jednakże zdecydowany nacisk położono na zagadnienia związane z realizacją metody elementu skończonego.

Należy jeszcze dodać, że przedstawiany projekt nie dotyczy części realizującej zasadnicze obliczenia metodą elementu skończonego, a jedynie usprawnienia procesu wykorzystania istniejących i sprawdzonych już programów realizujących te obliczenia.

Założenia projektowanego systemu

Przeznaczenie systemu

Projektowany system adresowany jest do konstruktorów pracujących w biurach konstrukcyjnych, ośrodkach badawczo-rozwojowych zajmujących się projektowaniem maszyn i urządzeń, w biurach projektowych oraz do wszystkich tych, którzy na codzień mają do czynienia z obliczeniami wytrzymałościowymi.

Tak bogaty krąg użytkowników narzuca na projektowany system warunek, że musi to być system uniwersalny, za pomocą którego można rozwiązywać dowolnego typu zadania z zakresu obliczeń wytrzymałościowych maszyn i urządzeń.

Z założenia tak szerokiego kręgu użytkowników wynika wniosek, że użytkownicy ci nie będą informatykami i nie będą chcieli z własnej woli stać się nimi. Stawia to przed projektantem systemu warunek maksymalnej prostoty korzystania z niego.

Zakres obliczeń wytrzymałościowych realizowanych przez system

Analiza wytrzymałościowa statyczna oraz dynamiczna ciał fizycznych korzysta z różnych teorii opisujących stan ciała pod działaniem sił.

Jedna z możliwych klasyfikacji tych teorii jest następująca:

- teorie liniowej sprężystości przy małych odkształceniach badanego ciała;
- teorie nieliniowe:
  - nieliniowość wynikająca z uwzględnienia zjawisk plastyczności,
  - nieliniowość wynikająca z dużych odkształceń badanego ciała,

Natomiast metody praktycznego rozwiązywania zadań, niezależnie od teorii, za pomocą której opisujemy zadanie można podzielić na dwie klasy:

- metody klasyczne,
- metoda elementu skończonego.

Do metod klasycznych zaliczamy metody wykorzystujące dokładne lub przybliżone lecz analityczne wzory na obliczanie żądanych wielkości.

Metoda elementu skończonego jest stosunkowo młodą metodą sformułowaną na przełomie lat 50-60; istotą tej metody jest zastąpienie obliczanej konstrukcji jej skończenie elementowym modelem i prowadzenie na tym modelu znacznie uproszczonych obliczeń.

Metody klasyczne możemy stosować tam, gdzie zachowane są założenia o ciągłości, jednorodności i izotropii materiału, płaskości przekrojów i usztywnień, prostym kształcie ciała oraz odkształceniach w granicach sprężystości materiału.

Metodami klasycznymi można więc np. obliczać:

- belki zginane statycznie wyznaczalne,
- belki zginane statycznie niewyznaczalne,
- belki na sprężystym podłożu,
- pręty skręcane,
- kratownice i ramy,
- płyty prostokątne o prostym obciążeniu.

Inne zadania rozwiązywane mogą być metodą elementu skończonego, która nadaje się teoretycznie do szerokiej klasy zadań z zakresu obliczeń wytrzymałościowych opisywanych za pomocą liniowej i nieliniowej teorii. W obecnym stadium rozwoju metody elementu skończonego może ona nie dawać dobrych wyników w zadaniach, w których należy uwzględniać problem plastyczności materiału, utraty stateczności konstrukcji i jej zachowania pokrytycznego.

Jak więc widać zakres obliczeń wytrzymałościowych przeznaczonych do realizacji przez projektowany system jest bardzo szeroki, ograniczony jedynie stanem rozwoju teorii w tej dziedzinie.

Sposób eksploatacji systemu

Projektowany system z założenia ma być systemem interaktywnym. Oznacza to, że użytkownik musi mieć możliwość ingerencji w proces obliczeniowy w czasie jego trwania.

Najlepszym sposobem eksploatacji systemu zapewniającym tę możliwość jest praca w trybie konwersacyjnym. Jednakże praca w takim trybie jest celowa po spełnieniu kilku podstawowych warunków, z których najważniejszy jest optymalny średni czas reakcji maszyny. Czas reakcji jest to czas, jaki upływa od momentu wprowadzenia ostatniego znaku przez użytkownika do momentu otrzymania pierwszego znaku odpowiedzi maszyny. Według badań przeprowadzonych przez psychologów czas ten powinien wahać się w granicach od 1 do 4 s. Czasy dłuższe niż 15 s praktycznie wykluczają możliwość pracy konwersacyjnej. Z tego względu przy realizacji programów konwersacyjnych należy zwrócić szczególną uwagę na prawidłowość podziału programu na części tak, aby czas reakcji mieścił się w optymalnych granicach.

Z analizy zadań i metod stosowanych do ich rozwiązywania wynika, że projektowany system nie bę-

dzie mógł być w całości systemem konwersacyjnym. Wynika to z faktu, że część systemu, która będzie wykorzystywać metodę elementu skończonego nie nadaje się do pracy w trybie konwersacyjnym. Czas rozwiązywania układu równań w metodzie elementu skończonego nawet dla małych zadań wielokrotnie przekracza maksymalne czasy możliwe do przyjęcia dla czasu reakcji maszyny przy pracy konwersacyjnej.

Zadecydowało to o tym, że projektowany system obliczeń wytrzymałościowych eksploatowany będzie w sposób mieszany. Część dotycząca obliczeń metodami klasycznymi oraz pewne fazy obliczeń w metodzie elementu skończonego realizowane będą konwersacyjnie a pozostałe fragmenty obliczeń metodą elementu skończonego, wsadowo. Proces obliczeń metodą elementu skończonego składa się z trzech głównych faz:

- przygotowanie danych (budowa modelu),
- właściwe obliczenia,
- interpretacja wyników.

Zakłada się, że faza druga realizowana będzie metodą wsadową. Faza pierwsza przygotowywania danych realizowana będzie konwersacyjnie. Faza trzecia częściowo wsadowo a częściowo konwersacyjnie.

Dodatkowe założenia dotyczące realizacji systemu

Oczywiste jest, że należy dążyć do minimalizacji czasu oraz kosztów realizacji projektowanego systemu.

Problem obliczeń wytrzymałości konstrukcji jest prawie tak samo stary jak projektowanie konstrukcji. Dlatego też w miarę rozwoju maszyn cyfrowych powstało bardzo dużo programów i systemów przeznaczonych do obliczeń wytrzymałościowych.

W projektowanym systemie przewiduje się w maksymalnym stopniu wykorzystanie istniejących systemów czy programów. Podejście takie umożliwia znaczną obniżkę kosztów i czasu realizacji systemu. Problem ten dotyczy zarówno części systemu opartego na metodach klasycznych jak i na metodzie elementu skończonego. Należy podkreślić, że czas realizacji systemu opartego na metodzie elementu skończonego waha się w granicach około 100 - 200 osobolat [5] a więc nie jest to problem błahy.

Przyjęto jeszcze jedno założenie wynikające z przesłanek ekonomicznych a mianowicie, że podstawę realizacji systemu stanowi sprzęt Jednolitego Systemu (JS EMC) produkcji krajowej lub krajów RWPG, a jedynie w wyjątkowych sytuacjach z pozostałych krajów.

Projekt systemu - wersja docelowa

Założenia sposobu użytkowania systemu

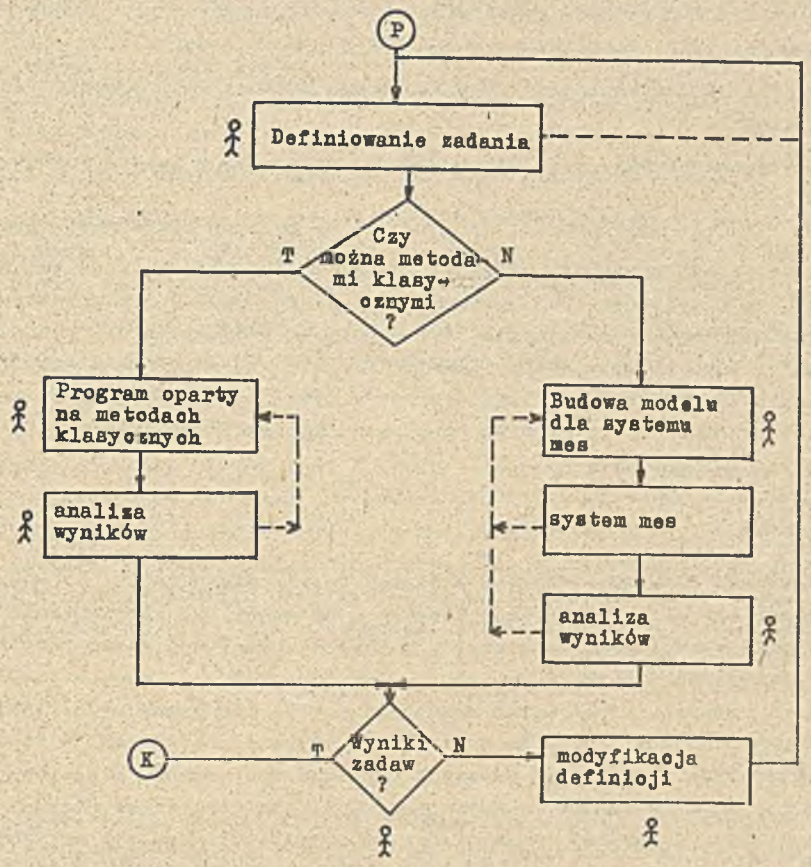
Użytkownik zgłaszający chęć skorzystania z usług systemu będzie musiał wykonać następujące czynności:

- zdefiniować swój problem,
- pomagać systemowi w doborze odpowiedniej metody do rozwiązania problemu,
- poprawiać błędy ujawniane w trakcie obliczeń a niemożliwe do wykrycia na etapie definiowania problemu,
- przeanalizować i zaakceptować wyniki.

Poglądowo czynności te przedstawiono na rys. 1. W następnych punktach omówione zostaną bardziej szczegółowo poszczególne czynności pokazane na tym rysunku.

Definiowanie zadania

W zagadnieniach wytrzymałościowych przez definicję zadania rozumie się podanie następujących danych o obliczanej konstrukcji:



Rys. 1. Organizacja systemu

Symbol "🧍" występujący na rys. 1 oznacza, że w czynnościach oznaczonych tym symbolem możliwy, a niekiedy nawet konieczny jest udział człowieka

- geometrię konstrukcji, tzn. jej kształt, wymiary, sposób połączeń poszczególnych elementów itp.,
- własności materiałów, z których konstrukcja jest wykonana,
- własności otoczenia, w którym konstrukcja będzie pracować,
- rodzaj, wielkość i sposób przyłożenia maksymalnych obciążeń, które ma przenieść konstrukcja.

Projektując oprogramowanie do tego typu zadań należy rozważyć sposób przedstawienia zadania w języku dostępnym do przetworzenia przez maszynę cyfrową.

Językiem, którym posługują się w tym wypadku konstruktorzy jest rysunek techniczny. Język ten doskonały w dotychczasowej praktyce inżynierskiej jest trudną do pokonania barierą w systemach komputerowych. Powstaje zatem do rozwiązania problem jednoznacznego zapisu geometrii części w sposób zrozumiały dla systemu a jednocześnie prosty dla użytkownika. W projektowanym systemie musi powstać taki język, który dalej zwać będziemy językiem opisu zadań (JOZ). Język ten będzie musiał minimalizować liczbę danych opisujących projektowaną konstrukcję.

Każda konstrukcja zapisana w języku maszyny cyfrowej przedstawiona jest jako zbiór liczb, np. w systemach metody elementu skończonego konstrukcję reprezentują setki tysięcy liczb, z których dużą część - w zależności od systemu - musi podać użytkownik. Konieczność podania bez błędu dziesiątków czy nawet setek tysięcy liczb jest poważną barierą utrudniającą stosowanie tych systemów w praktyce. Z tego względu przyjęto założenie o minimalizacji liczby danych opisujących konstrukcję. Dla prostych kształtów problem jest banalny i od dawna rozwiązany. Wiadomo przecież, że dla jednoznacznego opisu okręgu na płaszczyźnie potrzeba trzech liczb, prostopadłościanu w postaci

3-wymiarowej sześciu itd. Dla złożonych konstrukcji o skomplikowanych kształtach problem nie jest już tak prosty, szczególnie, że istnieje wiele możliwych sposobów opisu konstrukcji.

Zadaniem JOZ będzie pełne zdefiniowanie rozwiązywanego zadania tzn. podanie danych opisujących konstrukcję i stan obciążeń.

Zakłada się, że użytkownik będzie miał do dyspozycji następujący zestaw graficznych urządzeń peryferyjnych:

- plotter,
- digitezer,
- monitor graficzny z piórem świetlnym.

Urządzenia te będą stanowić pomoc na etapie definiowania zadania. Monitor ekranowy będzie umożliwiał komunikowanie się z systemem oraz wprowadzanie pewnych danych z klawiatury lub za pomocą pióra świetlnego. Digitezer będzie pełnił podstawową rolę przy odczytywaniu rysunków technicznych. Zadaniem plottera będzie wizualizacja zdefiniowanego zadania mająca na celu sprawdzenie poprawności wprowadzonych danych. Oczywisty jest fakt udziału w tych czynnościach człowieka. System - przez odpowiedni dobór urządzeń peryferyjnych oraz przez specjalistyczne oprogramowanie - może mu w tym pomóc.

Wybór metody rozwiązania

Jak już było powiedziane w punkcie "Zakres obliczeń wytrzymałościowych..." w systemie przewidziano dwie zasadnicze grupy metod, za pomocą których będą mogły być rozwiązywane poszczególne zadania - metoda elementu skończonego oraz metody klasyczne. Powstaje zatem problem wyboru dla konkretnego zadania odpowiedniej metody. Jak już powiedziano, metoda elementu skończonego jest metodą uniwersalną. Każde zadanie, które można rozwiązać metodą klasyczną można także rozwiązać metodą elementu skończonego. Metody klasyczne są tańsze i szybsze, co stanowi czynnik decydujący o ich stosowaniu do określonych zadań, gdyż programy rozwiązujące zadania z zakresu wytrzymałości metodami klasycznymi nie są uniwersalne. Jednym programem można rozwiązać niewielką klasę zadań bardzo zbliżonych do siebie. Jednak do zadań, które wykonywane są często a analiza teoretyczna dała dobrze uzasadnione wzory do ich rozwiązywania nie należy stosować metod klasycznych ze względów ekonomicznych. Z tego powodu w systemie przewidziano rozwiązywanie zadań metodami klasycznymi oraz metodą elementu skończonego.

Po sformułowaniu zadania system przystąpi do analizy warunków zadania a następnie przeglądu katalogu, w którym zebrane będą wszystkie informacje o programach rozwiązujących zadania metodami klasycznymi. W katalogu tym każdy program opisany będzie zbiorem informacji opisujących zadanie, które można za pomocą tego programu rozwiązać. Przedstawiona organizacja systemu w części dotyczącej metod klasycznych pozwala włączyć do systemu wiele istniejących już i sprawdzonych w praktyce programów opartych na metodach klasycznych. Programy te wymagać będą w najgorszym wypadku niewielkich zmian związanych z możliwością wykorzystania ich w systemie pod warunkiem, że będą mogły być wykonywane na maszynach cyfrowych, na bazie których zrealizowany będzie system. Ponadto proponowana organizacja pozwoli na stałą rozbudowę systemu w miarę rozwoju badań teoretycznych nad metodami klasycznymi, co umożliwiać będzie powstawanie coraz lepszych programów opartych na metodach klasycznych. Umożliwi to także lepsze dostosowanie możliwości systemu do potrzeb konkretnego użytkownika. Wybór odpowiedniego programu odbywać się będzie automatycznie, jednakże przewiduje się możliwość wyboru metody rozwiązania przez użytkownika, jeżeli jego doświadczenie oraz dobra znajomość zawartości biblioteki programów opartych na metodach klasycznych pozwoli mu na podjęcie takich decyzji.

Poprawianie błędów

Na każdym etapie rozwiązywania zadania mogą być wykryte błędy. Dotyczy to szczególnie zadań skomplikowanych, o dużej liczbie danych. Oczywiście w systemie muszą istnieć różnego rodzaju zabezpieczenia przed wykonywaniem obliczeń dla błędnych danych. Jest oczywiste, że im wcześniej błąd będzie wykryty, tym mniejsze są szkody, jakie może wyrządzić. Najwięcej błędów popełnianych będzie w pierwszej fazie, tj. definiowania zadania. Jednakże należy zdawać sobie sprawę z tego, że błędy mogą ujawniać się we wszystkich fazach rozwiązywania zadania, nawet dopiero w fazie analizy wyników.

Błędy mogące pojawiać się w trakcie rozwiązywania zadania można podzielić na dwie podstawowe grupy:

- błędy możliwe do wykrycia przez system,
- błędy niemożliwe do wykrycia przez system (możliwe do wykrycia przez człowieka śledzącego obliczenia lub analizującego wyniki).

W systemie przewiduje się trzy podstawowe mechanizmy wykrywania błędów.

- Pierwszy mechanizm jest to automatyczne wykrywanie błędów. Metodą tą można wykryć wiele błędów we wprowadzonych danych. Przyjęto jednak zasadę wykrywania tą metodą jedynie tych błędów, których koszt wykrycia jest znacznie mniejszy od kosztu rozwiązania zadania.
- Drugim mechanizmem wykrywania błędów jest wizualizacja danych przez graficzne przedstawianie obliczanej konstrukcji. Dotyczy to szczególnie etapu budowy modelu w metodzie elementu skończonego. Perspektywiczne przedstawienie pod różnymi kątami obliczanej części pozwala łatwo sprawdzić zgodność modelu z oryginałem.
- Trzecim mechanizmem usuwania błędów jest możliwie częste przedstawianie wyników częściowych użytkownikowi, np. na ekranie monitora. Metoda ta może być stosowana szczególnie w części systemu dotyczącej metod klasycznych. Należy jeszcze podkreślić, że w wypadku wykrycia błędu przez system, użytkownik otrzyma pełną diagnostykę błędu obejmującą jego opis, możliwe źródło wystąpienia oraz sposób usunięcia. Jest to bardzo ważne zagadnienie wymagające wielu badań nad diagnostyką błędów. Prawidłowe jego rozwiązanie będzie miało duże znaczenie dla prawidłowej pracy systemu.

Analiza wyników

Użytkownik otrzymuje wyniki będące efektem działania systemu. W zadaniach, do rozwiązywania których przeznaczony jest projektowany system, zazwyczaj będą to wartości sił wewnętrznych, naprężeń oraz przemieszczeń wybranych elementów konstrukcji.

Użytkownik otrzyma wydruk wyników na drukarce oraz ich graficzną interpretację na ploterze. Szczególnie należy tu podkreślić rolę plotera w interpretacji wyników. Dane przedstawione na drukarce w pełni opisują zachowanie się konstrukcji pod wpływem przyłożonych obciążeń. Jednakże opis ten zawiera bardzo dużo liczb, znacznie więcej niż było danych. Ich interpretacja na podstawie odczytu wydruków z drukarki, choć jest możliwa, jest jednak bardzo czasochłonna i łatwo o pomyłkę. Z tego względu w systemie przewiduje się maksymalne ułatwienie interpretacji wyników.

Przewiduje się dwie podstawowe metody wspomagające interpretację wyników:

- wizualizacja wyników na ploterze,
- przetwarzanie wyników z przedstawieniem rezultatów na drukarce.

Wizualizacja wyników pozwala w sposób szybki, lecz mało dokładny, na zorientowanie się o odkształceniach konstrukcji, miejscach koncentracji naprężeń, rozkładzie obciążeń. Następnie wykorzystując rysunek można dokładne wyniki odczytać z wydruków uzyskanych na drukarce w tych miejscach konstrukcji, w których występują największe siły czy naprężenia.

Drugą metodą wspomagającą interpretację wyników jest ich przetwarzanie, mające na celu znalezienie obszarów, w których występują maksymalne przemieszczenia konstrukcji, maksymalne naprężenia, maksymalne różnice w naprężeniach itp. Wyniki tej analizy w powiązaniu z wizualizacją wyników pozwolą znacznie skrócić etap interpretacji wyników i podjęcie przez użytkownika decyzji o zakończeniu obliczeń lub rozpoczęciu ich od nowa z nieco zmodyfikowanym modelem zadania.

Modyfikacje modelu dotyczyć mogą zmian w konfiguracji obciążeń, w wymiarach, bądź w zmienionym układzie poszczególnych elementów projektowanej konstrukcji. Ponadto przy wykorzystywaniu metody elementu skończonego może zachodzić potrzeba zmian w skończenie elementowym modelu konstrukcji. Wynikać to może z konieczności zagęszczenia podziału w tych fragmentach konstrukcji, w których występują duże naprężenia lub inne zjawiska wymagające dokładniejszego zbadania. Czasem także może zachodzić potrzeba użycia innego rodzaju elementów pozwalających dokładniej opisać występujące w konstrukcji zjawiska.

Oczywisty jest fakt, że wszystkie decyzje tego typu może podjąć tylko człowiek na podstawie analizy otrzymanych wyników oraz własnego doświadczenia i wiedzy fachowej.

Oprogramowanie użytkowe systemu

Jak już wielokrotnie wspomniano, projektowany system będzie składał się z dwóch podstawowych części. Część pierwsza będzie obejmować rozwiązywanie wybranych zadań metodami klasycznymi, a część druga metodą elementu skończonego. Wspólny dla obu części będzie JOZ (zob. pkt. "Założenia sposobu użytkowania systemem").

Język ten będzie klamrą spinającą cały system w zwarty organizm pozwalający użytkownikowi rozwiązywać różnorodne zadania.

Prace nad tym językiem są bardzo żmudne i wymagają jeszcze dużego nakładu sił i środków na jego zaprojektowanie i oprogramowanie.

Część klasyczna systemu

Część systemu realizująca obliczenia metodami klasycznymi będzie składać się ze zbioru specjalizowanych programów przeznaczonych do rozwiązywania konkretnych zadań. Zbiór ten będzie skatalogowany. W katalogu tym każdy program będzie opisany w taki sposób, żeby na podstawie opisu zadania było można jednoznacznie stwierdzić, czy istnieje program w katalogu rozwiązujący dane zadanie.

Dokładną organizację katalogu należy projektować łącznie z językiem opisu zadań, co pozwoli na optymalną jego organizację.

Z analizy istniejących w kraju systemów przeznaczonych do obliczeń wytrzymałościowych wynika, że istnieje dużo programów napisanych w różnych językach i uruchomionych na różnych maszynach cyfrowych. Ze zbioru tego każdy właściciel systemu powinien wybrać odpowiedni dla siebie zestaw programów i włączyć go do systemu. Część tych programów będzie można włączyć bez żadnych przeróbek, część po dokonaniu pewnych modyfikacji, a inne po gruntownym przerobieniu. Jednakże rozwiązanie takie pozwoli na szybkie zapełnienie biblioteki dobrymi i tanimi programami. Zasadnicze zmiany w istniejących programach dotyczyć będą możliwości wykorzystania najnowocześniejszych urządzeń peryferyjnych przewidzianych w systemie.

Jak już było powiedziane, obecny rozwój metod klasycznych daje rozwiązania dla belek, prętów i płyt przy spełnieniu założeń o ciągłości, jednorodności i izotropii materiału, płaskości przekrojów i usztywnień oraz niezbyt skomplikowanych kształtach i obciążeniach. Mimo tak mocnych ograniczeń, metodami tymi, przy pewnych uproszczeniach rzeczywistego modelu obliczanej konstrukcji, można obecnie osiągnąć dobre rezultaty dla wielu konstrukcji inżynierskich. Dla ułatwienia budowy programów opartych na metodach klasycznych, a przeznaczonych do bardziej skomplikowanych konstrukcji projektuje się wyposażenie systemu w programy do rozwiązywania problemów najczęściej spotykanych w praktyce inżynierskiej. Programy te będą mogły być wykorzystane w dwojaki sposób:

- jako samodzielne programy przeznaczone do rozwiązywania prostych zadań inżynierskich,
- jako podprogramy przeznaczone do wykorzystania w bardziej skomplikowanych programach.

Do problemów, najczęściej spotykanych w praktyce inżynierskiej, dających się rozwiązywać metodami klasycznymi zaliczono:

- belki zginane statycznie wyznaczalne, tj. wspornikowe, swobodnie podparte, swobodnie podparte z jednym wspornikiem, swobodnie podparte z dwoma wspornikami,
- belki zginane statycznie niewyznaczalne jedno- i wieloprzęsłowe,
- belki ciągłe długie na sprężystym podłożu.

W programach tych przewiduje się obliczanie reakcji podporowych i maksymalnych momentów zginających, sił poprzecznych i momentów zginających w zadanych punktach, maksymalne ugięcia belki dla różnych konfiguracji obciążeń.

Następna grupa programów dotyczyć będzie obliczania różnego rodzaju prętów pracujących w różnych warunkach; będą to w szczególności:

- pręty swobodnie skręcane o różnych przekrojach,
- pręty zakrzywione o małej krzywiźnie, statycznie wyznaczalne, zginane w płaszczyźnie przekroju i prostopadle do niego,
- pręty zakrzywione statycznie niewyznaczalne, obustronnie utwierdzone o różnym kącie rozwarcia,
- pierścienie koliste zginane w swej płaszczyźnie,
- pręty zakrzywione o dużej krzywiźnie,
- pręty smukłe ściskane i zginane, zginane i rozciągane.

W programach tych obliczane będą momenty zginające w dowolnym przekroju pręta, ugięcie pręta w dowolnym miejscu i inne wielkości charakterystyczne dla poszczególnych rodzajów prętów, dla różnych przypadków obciążeń pręta.

Dalsza grupa programów dotyczyć będzie obliczeń wytrzymałościowych, najczęściej występujących w praktyce płyt, rur i zbiorników.

Podany zestaw programów pozwoli na bezpośrednie ich zastosowanie do rozwiązywania elementarnych zadań, a ponadto stanowić będzie zestaw podprogramów do wykorzystania w większych specjalizowanych programach przeznaczonych do bardziej złożonych konstrukcji. Konstrukcje te bardzo często składają się z różnego rodzaju połączeń belek, prętów czy płyt.

Programy realizujące tę część systemu w zasadzie powinny być przystosowane do pracy w trybie konwersacyjnym. Praca w tym trybie może być bardzo wydajna i efektywna pod warunkiem, że spełnione będą podstawowe wymagania stawiane takim programom, przy czym za jedno z najważniejszych wymagań stawianych programom konwersacyjnym należy uznać zapewnienie optymalnego średniego czasu reakcji systemu (zob. pkt "Sposób eksploatacji systemu").

Innym, niemniej ważnym problemem jest uodpornienie programu na błędy popełniane przez użytkownika. Problem ten jest niezależny od sposobu eksploatacji programu. O ile jednak przy przetwarzaniu wsadowym po wystąpieniu błędu jest czas na analizę przyczyny jego wystąpienia i sposobu usunięcia tej przyczyny, to przy pracy w trybie konwersacyjnym nie ma czasu na dłuższą analizę przyczyny błędu. Należy więc dążyć do tego, żeby błędy wykrywać możliwie jak najwcześniej. Jednym ze sposobów jest sprawdzenie przez użytkownika, czy wprowadzona odpowiedź do programu jest taka, jaka powinna być, czy wprowadzone liczby są dobre. Pozwala to uniknąć prostych pomyłek popełnianych przy wprowadzaniu danych z klawiatury. Najtrudniej jest wykryć przyczyny błędów ujawnianych dopiero w trakcie wykonywania obliczeń. Należy wtedy przekazać użytkownikowi możliwie bogatą informację o rodzaju błędu, miejscu jego wystąpienia i możliwych sposobach usunięcia (zob. pkt "Założenia sposobu użytkowania systemu").

Następnym zagadnieniem, które należy rozwiązać przy projektowaniu programów konwersacyjnych jest organizacja dialogu. Biorąc za kryterium podziału aktywność użytkownika podczas prowadzenia dialogu, można wyróżnić dwa podstawowe typy dialogów:

- bierny użytkownik - użytkownik odpowiada na pytania stawiane przez maszynę,
- czynny użytkownik - maszyna wykonuje polecenia użytkownika.

W praktyce, szczególnie przy bardziej skomplikowanych zadaniach, w jednym dialogu mogą występować oba typy razem; będzie to wtedy dialog mieszany. W dialogu takim, w jednej fazie stroną aktywną może być maszyna, a w innej użytkownik.

Należy jednak zdawać sobie sprawę, że ze względu na dużą różnorodność rozwiązywanych zadań, należy przed przystąpieniem do budowy programu analizować różne warianty budowy dialogu i wybierać optymalny dla danego zadania oraz sprzętu, którym się dysponuje.

Część systemu oparta na metodzie elementu skończonego

Zgodnie z podanymi fazami procesu obliczeń konstrukcji metodą elementu skończonego przyjęto następujące nazewnictwo dla programów obsługujących te fazy. Główną część systemu realizującą obliczenia zasadnicze nazywa się procesorem systemu, programy pomagające przygotowywać dane zwane są

preprocesorami, a programy pomagające analizować wyniki postprocesorami.

Doświadczenia zebrane z eksploatacji różnych systemów metody elementu skończonego [1] pozwalają na stwierdzenie, że około 60 do 80% czasu przeznaczonego na rozwiązanie zadania pochłania faza pierwsza, 20 do 30% faza trzecia, a jedynie 5 do 10% właściwe obliczenia. Chcąc zatem opracować dobry system oparty na metodzie elementu skończonego, należy główny nacisk położyć na etap pierwszy - przygotowanie danych.

Danymi w metodzie elementu skończonego są [6]:

- współrzędne węzłów siatki w podziale konstrukcji na elementy,
- informacje o typie, liczbie i sposobie łączenia elementów,
- informacje o warunkach brzegowych, tzn. o sposobie zamocowania konstrukcji, jej temperaturze, początkowych przemieszczeniach lub naprężeniach,
- informacje o rodzaju, wielkości i rozmieszczeniu obciążeń,
- charakterystyki materiałowe,
- stałe fizyczne.

Ogólna liczba danych rośnie bardzo szybko wraz ze wzrostem złożoności kształtu i wielkości obliczanej konstrukcji.

Dużo zadania opisywane są nawet przez kilkadziesiąt tysięcy liczb. W związku z tym prawidłowe ich przygotowanie stanowi dużą trudność i praktycznie wymaga wielokrotnych poprawek. Ponadto podawane współrzędne węzłów siatki w większości wypadków użytkownik musi obliczać, co stanowi dodatkowe źródło błędów. Wydaje się, że najlepszym sposobem minimalizacji liczby danych podawanych przez użytkownika jest uniwersalny generator siatki. Niestety, mimo intensywnych badań i sygnalizowanych w literaturze efektywnych rozwiązań żaden z systemów dostępnych w kraju nie ma dobrego generatora. Na przykład w systemie SESAM-69 istnieje generator siatki ukierunkowany na opis konstrukcji statków, szczególnie ich kadłubów, ale do konstrukcji nieregularnych, charakterystycznych dla przemysłu maszynowego jego przydatność jest wątpliwa.

W projektowanym systemie przewiduje się opracowanie uniwersalnego generatora siatki, co powinno przyczynić się do znacznego uproszczenia korzystania z systemu w części dotyczącej metody elementu skończonego.

Przyjęto następujące założenia do budowy uniwersalnego generatora siatki:

- minimalizacja liczby danych niezbędnych do prawidłowego działania generatora,
- podział wybranych fragmentów konstrukcji na żądany rodzaj elementów (z dostępnych w systemie),
- podział wybranych fragmentów konstrukcji z żądaną przez użytkownika dokładnością.

Podstawową grupę danych dla generatora siatki stanowić będą dane uzyskane przy opisie zadania.

Przewiduje się, że generator siatki będzie programem konwersacyjnym. W trakcie jego działania użytkownik będzie informowany o rodzaju użytych elementów i dokładności podziału konstrukcji na elementy. Użytkownik będzie mógł wskazywać fragmenty konstrukcji, dla których żąda gęstszego podziału lub zmiany rodzaju użytych elementów. Podzielona na elementy konstrukcja przedstawiana będzie graficznie na ploterze, co jest najlepszym sposobem sprawdzenia poprawności dokonanego podziału.

Przewiduje się, że realizacja generatora siatki wraz z JOZ, znacznie usprawni pierwszą fazę (budowy modelu) obliczeń metodą elementu skończonego. Ponadto przewiduje się realizację następujących preprocesorów usprawniających etap przygotowania danych i zapewniających wzrost prawdopodobieństwa osiągnięcia sukcesu w drugiej zasadniczej fazie obliczeń metodą elementu skończonego:

- szacowanie czasu obliczeń zasadniczych,
- podawanie wykazu użytych elementów i ich rodzaj,
- podawanie maksymalnych i minimalnych odległości między węzłami w całej konstrukcji.

Drugim etapem obliczeń w metodzie elementu skończonego są właściwe obliczenia, realizowane za pomocą odpowiedniego procesora, który we wszystkich istniejących systemach jest elementem najbardziej skomplikowanym i najkosztowniejszym. Jednocześnie są one już obecnie dość dobrze opracowane pod względem analizy użytych algorytmów i optymalizacji ich zaprogramowania. Z tego względu

zrezygnowano w projektowanym systemie z opracowywania tej fazy obliczeń od nowa.

Wykorzystanie istniejącego, sprawdzonego w praktyce systemu, pozwoli na znaczną obniżkę kosztów realizacji projektowanego systemu i znaczne skrócenie czasu jego realizacji. Z kilku dostępnych w kraju procesorów wybrano procesor systemu STRUDL II. System ten jest jednym z nowocześniejszych systemów i charakteryzuje się następującymi zaletami:

- dobra, jasna i przejrzysta diagnostyka błędów,
- łatwość dobudowywania preprocesorów,
- uruchomiony jest na maszynie IBM 360 i R20,
- dysponuje bogatą biblioteką elementów.

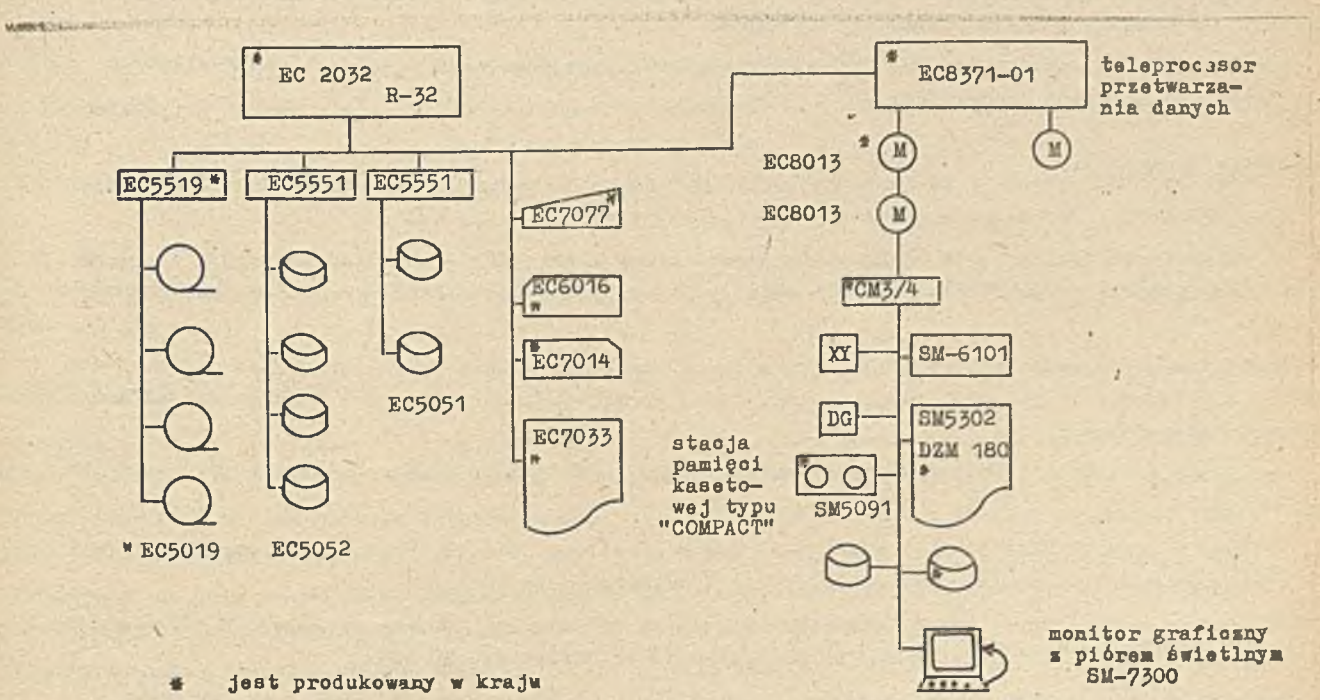
Wszystkie te zalety przemawiają za tym, żeby w projektowanym systemie wykorzystać właśnie procesor systemu STRUDL II.

Dla usprawnienia trzeciej fazy obliczeń, tj. analizy wyników - przewiduje się opracowanie następujących postprocesorów:

- podawanie ekstremalnych wartości obliczonych wielkości (naprężeń i przemieszczeń),
- wizualizacja otrzymanych wyników na ploterze,
- obliczanie pewnych funkcji na wielkościach obliczonych, pozwalających oceniać jakość konstrukcji, np. ciężar, objętość itp.,
- inne postprocesory - wykonujące przetwarzanie otrzymanych wyników w sposób żądany przez użytkowników.

Konfiguracja sprzętu niezbędnego do realizacji systemu

Przedstawiona w poprzednich punktach organizacja logiczna systemu wymusza określoną jego strukturę sprzętową umożliwiającą pełną realizację zadań stawianych przed systemem. Można przypuszczać, że wymagania te spełniać będzie wielodostępny abonencki system cyfrowy, którego konfigurację przedstawiono na rys. 2.



Rys. 2. Konfiguracja sprzętu

Jak widać na rys. 2 podstawową funkcję w systemie spełnia maszyna cyfrowa o dużej pojemności pamięci operacyjnej - minimum 512 K. Warunek ten spełniają maszyny z szeregu RIAD, typu R-30, R-32, R-40 i R-50.

Maszyna ta wyposażona jest w stacje dysków oraz taśm, a także musi mieć inne podstawowe urządzenia peryferyjne, jak drukarkę, konsolę operatorską, czytnik oraz perforator kart. Do maszyny tej podłączony jest procesor teleprzetwarzania danych, który umożliwia współpracę dużej maszyny z programowanymi punktami abonenckimi (PPA) za pośrednictwem linii telekomunikacyjnej. Każdy z PPA wyposażony jest w minikomputer jednolitego systemu SM.

Do minikomputera podłączone są następujące urządzenia wejścia/wyjścia:

- monitor ekranowy,
- czytnik kart,
- drukarka mozaikowa,
- pamięć kasetowa typu "COMPACT",
- pamięć taśmowa,
- pamięć kasetowa.

Specyfikacja sprzętu

Jak już było powiedziane we wstępie, system oparto na sprzęcie produkcji krajowej oraz krajów RWPG, a więc na sprzęcie Jednolitego Systemu. Wszystkie elementy systemu produkowane są w kraju bądź w krajach wspólnoty RWPG, z wyjątkiem urządzenia oznaczonego na rys. 2 symbolem DG. Jest to "digitizer" nie produkowany w krajach RWPG. Trudno jest proponować jakiś konkretny model z krajów kapitalistycznych. Można tu jedynie powiedzieć, że możliwe będzie podłączenie do systemu jakiegokolwiek urządzenia tego typu, współpracującego z minikomputerem amerykańskiej firmy PDP.

Należy zaznaczyć, że proponowane modele poszczególnych urządzeń są reprezentantami tych urządzeń produkowanymi w kraju lub najbardziej u nas rozpowszechnionymi. Oczywisty jest fakt, że można je zastąpić innymi urządzeniami tego typu ze sprzętu Jednolitego Systemu, a nawet spoza tego systemu, nawet zachodnimi. Dotyczy to szczególnie plotera, który jest trudno dostępny w kraju, ponieważ producent dopiero niedawno rozpoczął jego produkcję, podczas gdy w kraju pracują plotery firm zachodnich, jak np. Benson czy Calcomp. Wykorzystanie ich może wymagać dodatkowych nakładów na oprogramowanie czy budowę jednostek sterujących akceptowanych przez system, ale przy braku plotorów JS nakłady te mogą okazać się niezbędne.

Poniżej będzie przedstawiona krótka charakterystyka poszczególnych urządzeń oraz możliwości zastąpienia ich innymi urządzeniami.

Jednostka centralna

Jako podstawową jednostkę centralną proponuje się EC-1032, czyli R-32.

Jednostka centralna EC 1032 produkowana jest w kraju i składa się z trzech podstawowych bloków funkcjonalnych, tj. z procesora, pamięci operacyjnej oraz kanałów multiplekserowych i selektorowych.

Dla potrzeb projektowanego systemu minimalną pojemnością pamięci operacyjnej jest 512 K. Zaleca się rozszerzenie pojemności pamięci operacyjnej do maksymalnej pojemności 1024 K, co znacznie usprawni pracę systemu.

Jednostkę centralną EC 1032 można zastąpić następującymi jednostkami centralnymi z rodziny RIAD:

- EC 1030 produkcji ZSRR i WRL, składającej się z procesora EC 2030, pamięci operacyjnej EC 3203 (maks. do 512 K), kanałów multiplekserowego i selektorowego EC 4030,
- EC 1040 produkcji NRD, składającej się z procesora EC 2040, pamięci operacyjnej EC 3204 (maks. 1024 K), kanału multiplekserowego EC 4011, kanału selektorowego EC 4032,
- EC 1050 produkcji ZSRR, składającego się z procesora EC 2050, pamięci operacyjnej EC 3205 (maks. 1024 K), kanału multiplekserowego EC 4012 oraz kanału selektorowego EC 4035 .

Warto zauważyć, że jednostka EC 1040 jest bardziej wydajna od EC 1032 i EC 1030, ma większą szybkość, a więc nadawałaby się lepiej do pracy w projektowanym systemie niż poprzednie.

* Wg najnowszych danych w Czechosłowacji rozpoczęto produkcję digitizerów EC 7907

Model ten jest najwydajniejszym obecnie modelem w Jednolitym Systemie i zapewniłby on najszybszą pracę projektowanego systemu.

Ponadto przewiduje się, że w najbliższym czasie będą produkowane modele EC 1060 i EC 1065 jako najwydajniejsze z maszyn JS. Będą więc mogły z powodzeniem być użyte w projektowanym systemie.

Jednostki pamięci taśmowej

W systemie proponuje się jednostki pamięci taśmowej produkcji krajowej EC 5019, które są sterowane jednostką sterującą EC 5517, do której można podłączyć do 8 jednostek pamięci.

Pamięci EC 5019 można zastąpić (bez zmiany jednostki sterującej) następującymi typami pamięci taśmowej:

- EC 5010 produkcji ZSRR,
- EC 5012 produkcji ZSRR i LRB,
- EC 5017 produkcji ZSRR i NRD,
- EC 5022 produkcji CSRS,
- EC 5029 produkcji CSRS.

Wszystkie wymienione jednostki pamięci taśmowej mają metodę zapisu NR21, 9 ścieżek, gęstość zapisu od 9 do 32 bit/mm, a różnią się jedynie prędkościami przesuwu taśmy oraz maksymalną prędkością przesyłania danych.

Jednostki pamięci dyskowej

W systemie przewiduje się zainstalowanie pamięci dyskowej o wymiennych i stałych dyskach.

Podstawową jednostką o wymiennych dyskach jest pamięć EC 5022 produkcji krajowej lub LRB. Jest to pamięć o pojemności jednego pakietu 7,25 Mb. Jednostkę tę można zastąpić jednostkami: EC 5050 i EC 5056 - produkcji ZSRR.

Jako jednostkę o stałym dysku proponuje się EC 5051 produkcji ZSRR o pojemności 100 Mb.

Pamięci dyskowe dołączone są do jednostki centralnej przez jednostkę sterującą EC 5551 produkcji ZSRR. Do jednej jednostki sterującej można podłączyć do 8 jednostek pamięci dyskowej.

Drukarki

Jako podstawową drukarkę w systemie proponuje się drukarkę wierszową EC 7033 produkcji krajowej.

W programowanym punkcie abonenckim pracować będzie drukarka mozaikowa EC 7126 produkcji krajowej.

Obie drukarki mogą być zastąpione dowolnymi innymi drukarkami podobnej klasy. Jednakże ze względu na dobrze rozwiniętą w kraju produkcję drukarek i ich wysoką jakość nie podaje się oznaczeń innych modeli.

Czytniki i perforatory kart

Do dużej maszyny podłączony jest czytnik kart EC 6016 oraz perforator kart EC 7014 produkcji krajowej. Czytnik EC 6016 może być zastąpiony czytnikiem EC 6014 produkcji ZSRR.

Perforator EC 7014 może być zastąpiony jednym z perforatorów: EC 7010 i EC 7012 produkcji ZSRR oraz EC 7013 produkcji CSRS.

Urządzenia te podłączone są do jednostki centralnej przez kanał selektorowy lub multiplekserowy. Podłącza się je za pomocą złącza standardowego - standard interface.

W programowanym punkcie abonenckim zastosowano czytnik kart EC 6112.

Konsola operatorska

Duża maszyna wyposażona jest w konsolę operatorską EC 7077 produkcji krajowej. Można ją zastąpić konsolami:

- EC 7070 produkcji ZSRR,
- EC 7071 produkcji CSRS,
- EC 7073 produkcji NRD,
- EC 7074 produkcji LRB.

Procesor teleprzetwarzania

Jednym z podstawowych elementów projektowanego systemu jest krajowej produkcji procesor teleprzetwarzania danych (PTD) EC 8371-01.

PTD pośredniczy w przesyłaniu informacji między dużą maszyną cyfrową i zestawom programowanych punktów abonenckich. Procesor ten jest urządzeniem programowanym, wyposażonym w pamięć o pojemności do 256 Kb w zależności od potrzeb. Umożliwia on podłączenie do 352 półdupleksowych linii teletransmisyjnych. Programowalność TPD pozwala na poważne odciążenie jednostki centralnej i tworzenie różnorodnych konfiguracji sieci.

Modemy

Modemy służą do zamiany sygnałów z dopasowaniem do przesyłania typowymi liniami telekomunikacyjnymi. Przetwarzają one ciągi bitów (sygnały cyfrowe) na analogowe sygnały zmodulowane i odwrotnie.

W Systemie zastosowano modem EC 8013 o szybkości transmisji 1200/2400 b/s produkcji krajowej. Modem ten można zastąpić dowolnym z niżej przedstawionych:

- EC 8002 20 b/s produkcji PRL, NRD, WRL, CSRS,
- EC 8006 600/1200 b/s produkcji PRL, WRL, CSRS,
- EC 8019 24000/48000 b/s produkcji ZSRR.

Minikomputery

W projektowanym systemie przewidziano minikomputer SM.

Należy jednak podkreślić, że produkcja tych minikomputerów dopiero rozpoczyna się i obecnie brak jest szczegółowych informacji o tych maszynach. Wiadomo jedynie, że będą one przystosowane do współpracy z urządzeniami Jednolitego Systemu.

Monitory ekranowe

W systemie na poziomie programowanego punktu abonenckiego, zastosowano monitor ekranowy z piórem świetlnym SM 7300 produkcji ZSRR. Można go zastąpić podobnym monitorem produkcji węgierskiej.

Plotery

Proponuje się bębnowy ploter produkcji ZSRR, który to ploter ma być przedstawiony do badań w 1982 r.

Urządzenia dodatkowe

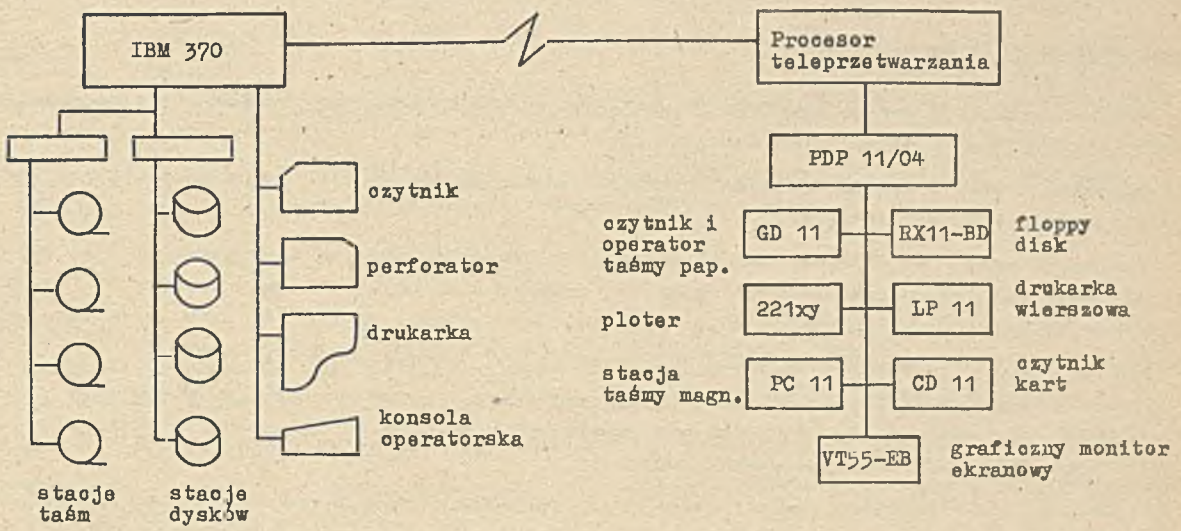
Ponadto system musi być wyposażony w dodatkowy sprzęt niezbędny w każdym ośrodku obliczeniowym, taki jak np. urządzenia do sprawdzania kart lub stacje przygotowania danych.

Uwagi dodatkowe o sprzęcie

Proponowany zestaw sprzętu można oczywiście zastąpić innym, w zależności od wymagań i możliwości jednostki eksploatującej system.

Wiele firm zachodnich oferuje kompletne zestawy urządzeń odpowiadających wymaganiom systemu. Można np. wymienić ofertę firmy Hewlett - Packard. Firma ta proponuje najnowszy swój system 9835, w którym minikomputer ma podstawową pojemność pamięci operacyjnej 64 Kb z możliwością rozbudowy do 256 Kb. Minikomputer ten można wyposażyć maksymalnie w 128 urządzeń peryferyjnych. Tak duży zestaw urządzeń peryferyjnych nie jest oczywiście potrzebny. Na potrzeby projektowanego systemu

wystarozy konfiguracja pokazana na rys. 3. Minikomputer ten może pracować z dużymi maszynami np. serii IBM 370, jako inteligentna końcówka w systemie wielodostępu i teleprzetwarzania.



Rys.3. Konfiguracja systemu z minikomputerami i urządzeniami peryferyjnymi firmy PDP

Zakończenie

Przedstawiony projekt systemu wzorcowego wymaga około 20 osobolat pracy oraz bogatego wyposażenia w sprzęt. Wiadomo jednak, że są trudności ze skompletowaniem odpowiedniego sprzętu, szczególnie aktywnego sprzętu graficznego. Z tego względu zdecydowano się na przyjęcie pewnych ograniczeń umożliwiających szybkie przystąpienie do realizacji i osiągnięcie efektów w krótkim czasie.

Powstała w ten sposób wersja modelowa. Praca nad tą wersją pozwoli sprawdzić w praktyce przyjęte założenia i udoskonalić koncepcję wersji docelowej.

Do realizacji wersji modelowej systemu przyjęto następujące założenia:

- pracochłonność realizacji maksymalnie 5 osobolat,
- wykorzystuje się dostępny sprzęt,
- wersja modelowa musi mieć walory użytkowe, tzn. powinna być systemem sprawnie działającym, choć o mniejszych możliwościach niż wersja docelowa.

Ograniczenia wersji modelowej w stosunku do wersji docelowej

W wersji modelowej rezygnuje się z JOZ. Część klasyczna i część oparta na metodzie elementu skończonego działają niezależnie, a wyboru metody dokonuje użytkownik systemu. Na etapie tym zbiera się wybrane programy metod klasycznych i opisuje je w katalogu. Programy te powinny obejmować podstawowe problemy, najczęściej spotykane w praktyce inżynierskiej.

W zakresie metody elementu skończonego główny wysiłek będzie skierowany na opracowanie generatora dokonującego podziału tylko na dwa elementy:

- izoparametryczny element 20 węzłowy - sześcian,
- izoparametryczny element 15 węzłowy - pryzmat.

Projektant będzie opisywał konstrukcję za pomocą minimalnej liczby tych elementów, a generator podzieli ją z żądaną dokładnością. Pozwoli to na zebranie doświadczeń do dalszej rozbudowy generatora przez wyposażenie go w inne rodzaje elementów. Da to także podstawy do sformułowania zasad języka opisu zadań tak, żeby był on efektywny dla potrzeb generatora siatki.

Opracowany generator powinien być napisany w języku ICES-TRAN (wersja PORTRAN-u), co pozwoli na łatwe włączenie go do systemu STRUDL-II.

Sposób eksploatacji wersji modelowej

Sposób eksploatacji wersji modelowej zależny będzie od sprzętu, jakim będzie dysponował użytkownik tego systemu. Możliwe tu są dwa warianty:

- eksploatacja przy istnieniu pełnego sprzężenia minikomputera z dużą maszyną,
- eksploatacja bez tego sprzężenia.

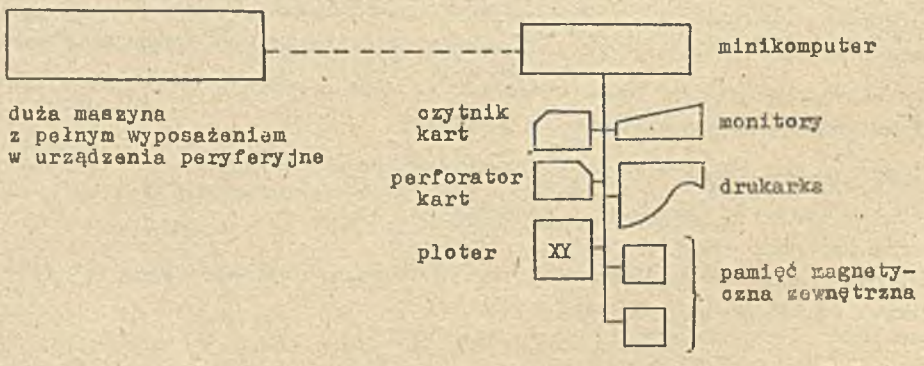
W pierwszym wariancie dane dla generatora siatki przygotowywane są na minikomputerze, a następnie przekazywane do programu generatora siatki przez minikomputer do dużej maszyny. W drugim wariancie użytkownik po przygotowaniu danych na minikomputer musi zapisać te dane na taśmę magnetyczną, taśmę papierową lub karty perforowane i wprowadzić do pamięci dużej maszyny.

Gdyby minikomputer, którym dysponować będzie użytkownik, wyposażony był w odpowiednio pojemną pamięć operacyjną możliwe jest uruchomienie generatora siatki na minikomputerze, co znacznie uprościłoby i przyspieszyło przygotowanie danych.

Modelowa wersja generatora siatki powinna być uruchomiona na minikomputerze oraz na dużej maszynie. Wersja eksploatowana na minikomputerze mogłaby być stosowana do mniejszych zadań, a do dużych zadań wymagających dużej pojemności pamięci operacyjnej należałoby stosować dużą maszynę.

Wymagania sprzętowe dla wersji modelowej

Istnieje minimalna konfiguracja sprzętowa, bez której eksploatacja nawet wersji modelowej byłaby nie tylko niemożliwe, ale i bezcelowa, ponieważ nie dawałaby nawet minimalnych efektów. Konfigurację taką przedstawia rys. 4.



Rys. 4. Konfiguracja sprzętu wersji modelowej

Dopuszcza się brak w minimalnej konfiguracji digitizera, monitora z piórem świetlnym oraz procesora teleprzetwarzania. W poważnym stopniu ogranicza to możliwości systemu, lecz dopuszczenie ich jest konieczne ze względu na trudności z dostępem do ww sprzętu w większości ośrodków zainteresowanych eksploatacją systemów metody elementu skończonego.

W konfiguracji przedstawionej na rys. 4 nie specyfikuje się dokładnie typów poszczególnych urządzeń. Dopuszcza się urządzenia dowolnego producenta, takie jakimi dysponować będzie użytkownik.

Wnioski do realizacji wersji modelowej

Wersja modelowa powinna być realizowana w ścisłej współpracy z przyszłym jej użytkownikiem i w miarę możliwości w jego ośrodku obliczeniowym lub w innym na takim samym lub podobnym sprzęcie. Zagwarantuje to w maksymalnym stopniu użyteczność wersji modelowej. Ponadto pozwoli na weryfika-

cję projektu wersji docelowej, po zebraniu doświadczeń z eksploatacji wersji modelowej, gwarantującą maksymalną użyteczność i sprawność systemu.

Uwagi końcowe

Przedstawiony w niniejszym opracowaniu projekt systemu należy traktować jako propozycję, którą należy przedyskutować w celu usunięcia jej wad.

Z tego względu autor prosi o kierowanie uwag związanych ze zgłoszoną propozycją. Szczególnie cenne uwagi mogą wnieść przyszli użytkownicy systemu, a więc projektanci.

Literatura

[1] RUMIŃSKI K., POŁCH E.Z., PAWLAK D.: Problemy efektywnego wdrażania systemów opartych na MES do praktyki inżynierskiej. Materiały z Konferencji COMPCONTROL, Warszawa 1977

[2] PAWLIK R.: Wybrane problemy współpracy człowieka z maszyną cyfrową. BIOSK 1977 nr 5/6,

[3] LUTOBORSKI A., PAWLIK R.: Analiza istniejących w kraju systemów MES i koncepcja systemu wzorcowego. BIOSK 1977 nr 5/6

[4] Katalog sprzętu Jednolitego Systemu. ODRI: Warszawa 1976

[5] FREDRIKSSON B., MACKERTE J.: Structural Mechanics Finite Element Computer Programs. Surveys and Availability. Linkoping Institute of Technology 1975, Linköping, Sweden

[6] BATHE K.J., WILSON E.L., PETERSEN F.E.: SAP IV. A Structural Analysis Program For Statio and Dynamic Response of Linear Systems. Berkley 1973 odbitka kserograficzna - materiały w posiadaniu autora

[7] FUCHS G., Schrem F.: ASKA - a Computer System For Structural Engineers odbitka kserograficzna - materiały w posiadaniu autora

[8] Biblioteka Programów. Sesam-69. Instrukcja użytkowania. Centrum Techniki Okrętowej, Gdańsk

[9] SZMELTER J., DACKO M.: Instrukcja programu "Analiza statyczna konstrukcji powłokowo-prętowych". KM WAT: Warszawa 1976

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