POLSKIE TOWARZYSTWO INFORMATYCZNE

Od GAM'a do ZAM'a
Niewykorzystana szansa polskiej informatyki?
Część III - Oprogramowanie

Opracowanie: Krzysztof Bytnerowicz

Sydney, maj 2020

Zeszyty Historyczne PTI

Zeszyt nr 6 - Od GAM-a do ZAM-a. Zaprzepaszczona szansa polskiej Informatyki?

CZĘŚĆ III - Oprogramowanie

ISBN 978-83-65750-05-1 (kolekcja, części I-III; wydania elektroniczne)

ISBN 978-83- 65750-15-0 (część III - Oprogramowanie; wydanie elektroniczne)

Opracowanie:
Krzysztof Bytnerowicz

Redakcja techniczna:
Krzysztof Bytnerowicz, Jerzy S. Nowak

Projekty graficzne:
Krzysztof Bytnerowicz

Wydawca:
Polskie Towarzystwo Informatyczne
Zarząd Główny
ul. Solec 38/103, 00-394 Warszawa

Wydanie I: maj 2020 r.

Zeszyty Historyczne PTI_
(Nº 6)

Od GAM-a do ZAM-a

Zaprzepaszczona szansa polskiej informatyki?

CZĘŚĆ III – OPROGRAMOWANIE

red. Krzysztof Bytnerowicz

Sydney, Australia
2020

Od GAM’a do ZAM’a, zaprzepaszczona szansa polskiej informatyki?

CZĘŚĆ III - OPROGRAMOWANIE

Informacje zebrał i komentarze dodał
Krzysztof Bytnerowicz
Sydney, Australia
2019-2020

Spis Treści

Oprogramowanie Maszyn ZAM ... 5
Zasady realizacji oprogramowania maszyny ZAM-41 ... 6
Wstęp ... 6
Założenia pracy ... 6
Etap 1 - System Oprogramowania 41 (SO 41) ... 6
Etap 2 - System Oprogramowania 141 (SO 141) ... 7
Etap 3 - System Oprogramowania 241 (SO 241) ... 7
Dane techniczne ... 8
Dane dotyczące ilości rozkazów zrealizowanego już systemu oprogramowania ... 9
Dyrygent ... 10
Bloki współpracy z urządzeniami zewnętrznymi ... 10
Blok współpracy z czytnikami ... 10
Blok współpracy z perforatorami ... 11
Blok współpracy z drukarką wierszową ... 12
Blok współpracy z monitorem ... 12
Blok współpracy z czytnikiem kart ... 13
Blok współpracy z pamięcią bębnową ... 13
Blok współpracy z pamięcią taśmową ... 13
Konfiguracje M.C. ZAM 41 ... 15
System Operacyjny SO-41 dla ZAM-41 ... 16
System operacyjny SO-141 ... 18
Biblioteka systemu oprogramowania ... 18
Edycje SO-141 ... 19
Wieloprogramowość ... 19
Język operacyjny maszyny (JOM) ... 20
Język PJP ... 21
Języki PJES i PJEG ... 21
Język SAS ... 21
Język SMAD ... 22
Języki BIB1 i BIB2 ... 22
Translatory WEBIN, WYBIN i WYSE ... 22
Język SMAO ... 23
Pogramy użytkowe ... 24
Programy manipulacyjne ... 24
Programy do przetwarzania danych ... 27
Testy ... 29
Język LOGOL dla maszyny ZAM-2 ... 30
(Jerzy Mysior) ... 30
Język i translator SAKO dla maszyny XYZ oraz maszyn ZAM 2, ZAM 21 i ZAM 41 ... 30
Translator ALGOL 60 na ZAM 21 ... 31
Translator ALGOL 60 dla ZAM 41 ... 31
Translator języka COBOL na ZAM 41 ... 31
CEMMA — Język modelowania symulacyjnego procesów ciągłych ... 32
EOL — Język do przekształcania symboli ... 33
Język GPSS ... 34
Język ASTEK ... 34

Języki BIGRAF i PLEXOL....34
System Diagnostyczny MC ZAM-41....35
WSTĘP....35
System Testów Kontrolnych K-400....36
Ogólne założenia Systemu Testów Kontrolnych K-400....36
Metoda Programowej Kontroli MC....36
Struktura Testów Kontrolnych K-400....37
Charakterystyka konstrukcji Systemu Testów kontrolnych K-400....40
Charakterystyka własności kontrolno-lokalizujących Systemu Testów Kontrolnych K-400....44
System Testów Lokalizujących L-141....45
Ogólne założenia Systemu Testów Lokalizujących L-141....45
Metoda lokalizacji uszkodzeń w MC....45
Struktura Systemu Testów Lokalizujących L-141....46
Charakterystyka konstrukcji Systemu Testów Lokalizujących L-141....49
Charakterystyka własności lokalizujących Systemu L-141....49
Uwagi o wynikach badań skuteczności systemu diagnostycznego MC ZAM-41....51
System pracy w czasie rzeczywistym - TRAN....52
Eksperymentalny system transmisji danych z maszyną cyfrową ZAM-41....53
Budowa systemu....53
Oprogramowanie systemu....55
Badania systemu....57
Wnioski....57
Generowanie konkretnego systemu oprogramowania - Język PJEG....58
Symboliczne Operacje Wejścia i Wyjścia....61
Operacje Wejścia-Wyjścia....61
Symboliczne operacje WE-WY w systemie oprogramowania ZAM-41....62
JOM - Realizacja....65
Symbole urządzeń We-Wy....65
Podprogramy Wejścia-Wyjścia....66
Niektóre dekodery wejściowe na KW6....66
Niektóre dekodery wyjściowe z KW6....66
Niektóre dekodery wejściowe na KW8....67
Niektóre dekodery wyjściowe z KW8....67
Zagadnienie generowania systemu oprogramowania....68
System Oprogramowania i jego Dokumentacja....71
System oprogramowania....71
Dokumentacja systemu oprogramowania....71
ZAM-41 - LISTA INSTALACJI MASZYN....74
TROCHĘ OSOBISTYCH UWAG....75
Terminologia....75
Włoszczyzna....75
Tracący czas....75
Post Mortem....75
Ściągaczka systemu....75
Podprogram za Prince Polo....75
Sobie ukochanemu ten fragment kodu poświęcam....75
IRIS 50 a sprawa polska....76
KISR i W.M. Turski – świat jest mały....76
DODATKI....77

System Automatycznego Kodowania – SAKO..... 77
WSTĘP..... 79
Ogólna struktura programów w języku SAKO.....81
Metodyka przygotowania programu.....83
FORMA I OPIS POJĘĆ PODSTAWOWYCH W JĘZYKU SAKO.....84
Liczby..... 84
Liczba ułamkowa..... 84
Liczba całkowita..... 85
Słowo bulowskie.....86
Blok liczbowy..... 86
Zmienne..... 88
Zmienna prosta.....88
Zmienna indeksowana.....89
Numery..... 89
Funkcje.....90
Wyrażenia arytmetyczne.....91
Wyrażenia całkowite i ułamkowe..... 93
Skala obliczania wartości wyrażenia..... 93
Wyrażenia bulowskie.....94
Lista.....95
FORMA I OPIS POSZCZEGÓLNYCH ROZKAZÓW SAKO.....96
Deklaracje..... 96
Rozkazy sterujące..... 104
Rozkazy arytmetyczne i bulowskie.....111
Rozkazy wejścia i wyjścia..... 116
Rozkazy współpracy z bębnem.....122
Komentarz.....123
PODPROGRAMY W JĘZYKU SAKO.....124
Terminologia i oznaczenia..... 124
Ogólne własności systemu korzystania z podprogramów w języku SAKO.....128
Budowa podprogramów SAKO.....128
Korzystanie z podprogramów, napisanych w języku SAKO.....131
Parę przykładów napisanych w SAKO dla ZAM-41..... 135
Jan Ściegienny - Maszyny ZAM-2 w NRD.....136
Władysław Klepacz, Jan Wierzbowski – Zastosowanie ZAM-2 w przedsiębiorstwie ubezpieczeniowym..... 138
Wstęp..... 138
Potrzeby obliczeniowe „WARTY”..... 138
Decyzja realizacji problemu..... 139
Przygotowanie dokumentów źródłowych.....140
Koncepcja rozwiązania systemu analizy..... 140
Realizacja systemu.....141
Modyfikacja systemu.....142
Druga koncepcja oraz eksploatacja systemu.....143
Podsumowanie..... 144

Oprogramowanie Maszyn ZAM

(J. Dańda)

Samodzielne opracowanie systemu oprogramowania SO-141, oprócz efektu bezpośredniego, jakim było wyposażenie maszyn cyfrowych ZAM-41 w nowoczesny (w owym czasie) system operacyjny, przyniosło dużo ważniejsze skutki pośrednie – powstał mianowicie zwarty zespół konstruktorów oprogramowania, przygotowanych do podjęcia jeszcze ambitniejszych zadań.

Było to w owym czasie największe przedsięwzięcie z dziedziny oprogramowania maszyn o całkowicie oryginalnej polskiej konstrukcji. Prace rozpoczęto w 1967 r. i prowadzono w zespole ok. 60 programistów przez okres ok. 4 lat.

Zasady realizacji oprogramowania maszyny ZAM-41

Andrzej M. Wiśniewski
Instytut Maszyn Matematycznych

Wstęp

Referowane poniżej wyniki dotyczą prac prowadzonych na maszynie ZAM-41 w Instytucie Maszyn Matematycznych.

Założenia pracy

W pracach bierze udział około 60 osób. Jest to jak na warunki polskie duży zespół osób, lecz w porównaniu z innymi firmami, takimi jak ICT lub IBM, jest to zespół od 10 do 15-krotnie mniejszy niż w tych firmach. Aby zabezpieczyć mimo to niezbędny zakres oprogramowania, włączono w skład oprogramowania środki pozwalające na sprawne pisanie elementów oprogramowania, a mianowicie języki:

PJES - maszynowo zorientowany język symboliczny bez możliwości używania nazw symbolicznych, dołączenia podprogramów bibliotecznych i korzystania z liczb zmiennoprzecinkowych, a więc podstawowych elementów koniecznych do sprawnego pisania programów użytkowych w języku symbolicznym. Natomiast język ten ma wygodny aparat działania na adresach, możliwości działania na licznikach translatora służących do rozmieszczania programu wynikowego w PAO i na bębnie oraz możliwość łączenia części programów w jedną całość, a więc wszystkie cechy potrzebne do pisania elementów oprogramowania.

E**OL** - służący do pisania translatorów (Z.Wrotek, J.,Walasek: „Język EOL i jego realizacja na maszynie ZAM-41”).

Prace nad oprogramowaniem maszyny ZAM-41 zaplanowano na okres około 3 lat. Całość prac nad oprogramowaniem podzielono na trzy etapy:

Etap 1 - System Oprogramowania 41 (SO 41)

Elementy oprogramowania z wyjątkiem biblioteki podprogramów podczas działania maszyny znajdują się na bębnie. Podprogramy wchodzące w skład biblioteki znajdują się na taśmie papierowej zaś na bęben wprowadza się tylko aktualnie potrzebny zestaw podprogramów. System ten jest przygotowany do pracy w zestawie standardowym. Dopuszcza się

dostosowywanie go do różnej liczby stacji taśm magnetycznych w zakresie od 2 do 6 oraz dwóch różnych typów drukarek i czytników kart. Dostosowanie to nie jest osiągane przy pomocy środków programowych, ale na drodze ręcznej zmiany parametrów zgodnie z określoną metodyką.

System ten pozwala na równoległe wykonywanie dwu programów w ustalonych zestawach.

Etap 2 - System Oprogramowania 141 (SO 141)

Wszystkie elementy oprogramowania znajdują się w Bibliotece Systemu. Biblioteka Systemu może być rozmieszczona na bębnie lub taśmach magnetycznych (S. Sawicki, E. Zaborowska: „Generowanie Systemu Oprogramowania"). System ten poza wyżej wymienionymi właściwościami "organizuje" prace na taśmach magnetycznych.

Etap 3 - System Oprogramowania 241 (SO 241)

Jest to system, który poza wszystkimi cechami wymienionymi wyżej pozwala na automatyczne przystosowywanie systemu do zadanego zestawu,tzn. generowanie systemu. System ten w swej końcowej postaci będzie pozwalał na 'zamawianie' potrzebnych do realizacji danego zadania środków (w sensie PAO, we-wy itp.) oraz wykonywanie do czterech programów równolegle.

Taki podział prac pozwolił stosunkowo szybko wykonać etap 1 (S0 41) (w rok po rozpoczęciu prac i w okresie, kiedy wyprodukowano pierwszą maszynę), a co za tym idzie, jak najszybciej zabezpieczyć pracę użytkownika na maszynie. Jednocześnie podział ten pozwoli na prowadzenie prac nad oprogramowaniem w sposób ciągły.

Duża ilość kodów zewnętrznych stosowanych na urządzeniach do przygotowywania danych oraz dopuszczenie już w założeniach ich stosowania na maszynie ZAM-41 spowodowało, że maszynę wyposażono w środki programowe pozwalające na pisanie programów niezależnych od kodów konkretnych urządzeń zewnętrznych (J. Witaszek: "Symboliczne operacje wejścia-wyjścia programów").

Obecnie są już zakończone prace nad SO 41, a w końcu 1968r. zostanie oddany do użytku SO 141.

Przyjęto zasadę, że użytkownik w odcinkach półrocznych będzie otrzymywał odpowiednio ulepszony i skorygowany System Oprogramowania.

Dane techniczne

W części tej zostaną podane przykładowe dane liczbowe pozwalające na ocenę możliwości obliczeniowych maszyny ZAM-41 oraz ocenę wielkości zainwestowanych prac w oprogramowanie.

Dane dotyczące możliwości obliczeniowych maszyny ZAM-41

Nazwa zadania	Czas zadania
Opracowanie (obejmujące sortowanie danych na 4 stacjach magnetycznych) 50 tys. dokumentów 30-znakowych zapisanych na taśmie magnetycznej i wypisanie sprawozdań w pięciu przekrojach.	2 godz.
Opracowanie 1000 dokumentów obejmujące: wczytanie z taśmy papierowej, posortowanie w PAO i wypisanie sprawozdania.	5 min.
Przejrzenie taśmy magnetycznej zawierającej 2700 dokumentów 600-znakowych z wybraniem pozycji według zadanego klucza bez czasu drukowania wybranej pozycji.	1 min.
Czas sortowania 100 tys. pozycji 8-znakowych na 4 taśmach magnetycznych (Pozycje były przed sortowaniem ułożone w odwrotnej kolejności).	56 min.
Rozwiązywanie prostego zagadnienia simpleksowego zmodyfikowaną metodą Danziga dla macierzy o wymiarach 37 x34.	20 min.
Analiza drogi krytycznej (czas, koszty) dla sieci o 583 działaniach.	3 min.
Rozwiązywanie prostego zagadnienia transportowego (bez ograniczeń i węzłów swobodnych) zmodyfikowaną metodą Fordu-Fulkersana dla macierzy o wymiarach 169x33.	7 sek nie licząc czasu wczytywania i drukowania danych

Podane powyżej dane stanowią konkretne zadania lub fragment zadań, które były rozwiązywane na maszynie ZAM-41.

Wykonywano również badania nad efektywnością dwu-programowości. Do badań używano zadań numerycznych (nie korzystając z taśm magnetycznych). Otrzymane wskazują przystosowanie dwu-programowości zyskując od 20 do 30% czasu maszyny.

Dane dotyczące ilości rozkazów zrealizowanego już systemu oprogramowania

Nazwa pozycji	Ilość rozkazów
System oprogramowania SO 41 wraz z już funkcjonującymi elementami SO 141	120000
System operacyjny obejmujący: Supervisor oraz zespół programów nadzorujących wykonanie problemu	6000
Supervisor zabezpieczający m in. obsługę przerwań i współpracę z urządzeniami zewnętrznymi.	4000
Translator JOM (Język służący do opisywania czołówki (JOB-u)programu	1500
Translator PJES	2500
SAS	10000
EOL	4000
Interpreter EOL	2500
Translator ALGOL	6000
Interpreter ALGOL	5000
Translator PJEG	3000
Biblioteka podprogramów i programów do przetwarzania danych obejmującą głównie programy kopiujące współpracę z TM i innymi urządzeniami we-wy.	7500
Programy z zakresu zastosowań do przetwarzania danych (Systemy)	4000
Biblioteka podprogramów i programów z zakresu: metod numerycznych, metod programowania liniowego, metod transportowych oraz programowania sieciowego.	2400

Poprzednik SO-41

(KB) Bardzo ciekawy opis systemu sterującego pracą maszyny ZAM-41 Alfa, być może również ZAM-21, w skrypcie Politechniki Gdańskiej „PJP ZAM 41 Podstawowy język programowania 1974”. Jest to unikalny dotychczas dokument opisujący logikę działania części Dyrygenta, podprogramów obsługi urządzeń zewnętrznych, zwanych w wewnętrznej terminologii ‘włoszczyzną’.

Dyrygent

O funkcjach programu dyrygent wspomnieliśmy już nieco w pkt. 3.1. Obecnie omówione funkcje dyrygenta podamy **s**ystematycznie.

Dyrygent :

a) nadzoruje pracą programów sterujących urządzeniami zewnętrznymi, tworząc kolejki operacji do wykonania
b) kontroluje czas wykonywania się niektórych operacji w urządzeniach zewnętrznych
c) kontroluje czas wykonywania programu użytkowego
d) steruje rozpoczęciem i zakończeniem wykonywania programu użytkowego
e) przyjmuje przerwania zegara wewnętrznego.

Bloki współpracy z urządzeniami zewnętrznymi

Blok współpracy z czytnikami

Ponieważ bloki obu czytników działają identycznie, rozpatrzymy tylko blok współpracy z czytnikiem na stoliku I.

Program sterujący pracą czytnika zawiera dwa bufory o pojemności 16 słów oznaczone przez A i B, umieszczone w określonym miejscu pamięci operacyjnej. Po uruchomieniu programu „System operacyjny” rozpoczyna się operacja czytania 32 rządków taśmy, w wyniku której oba bufory zostają zapełnione. Tak więc, jeżeli nawet nie zostało wydane żadne polecenie czytania znaku, to program sterujący, na polecenie dyrygenta, spowoduje przeczytanie 32 rządków taśmy.

Jeżeli później, w trakcie wykonywania programu pojawi się polecenie czytania znaku (w formie rozkazu SLR 1), to polecenie to nie spowoduje fizycznego ruchu taśmy związanego z odczytem, zaś jedynie przesianie z bufora A do rejestru B (zgodnie z określeniem rozkazu SLR 1) pierwszego z wczytanych znaków. Dalsze rozkazy SLR 1 powodować będą również tylko przesyłanie kolejnych znaków z bufora A do rejestru B aż do wyczerpania zawartości bufora tj. pobrania z niego 16 znaków. Wówczas

nazwy obu buforów zostają zamienione (przez zamianę nazw należy tu rozumieć zamianę adresów obu buforów) ewentualne dalsze rozkazy SLR 1 powodować będą pobieranie znaków z bufora A (po zamianie nazwy). Z chwilą opróżnienia jednego z buforów uruchamiany jest, niezależnie od ewentualnych rozkazów czytania znaku SLR l, program sterujący czytnikiem, należący do systemu operacyjnego. Rezultatem działania tego programu, o którym zakłada się, że nie może trwać dłużej niż 1 sek., jest wczytanie do pustego bufora dalszych 16 rządków taśmy. Jeżeli program nie zakończy się w przewidzianym czasie l sek., co wskazuje na zacięcie czytnika lub koniec taśmy. czytnik jest wyłączany, za w nieuzupełnione miejsca bufora wpisuje się zera.

Zdarza się również, że program główny szybciej pobiera znaki z buforów, niż program sterujący czytnikiem jest w stanie je zapełnić. Wówczas wykonywanie programu głównego jest zawieszone aż do do czasu wypełnienia jednego z buforów.

Program sterujący czytnikiem każdorazowo przed uruchomieniem czytnika bada stan jego wskaźników programowych, których stan wskazuje na zezwolenie, lub nie, czytania taśmy. Zmiana tych wskaźników następuje w wyniku naciśnięcia przycisku „Zgłoszenie operatora” ZO na stoliku operatora.

Wyczerpanie zawartości jednego z buforów czytnika II przy braku zezwolenia na czytanie powoduje zapalenie lampki alarmu LA.

Blok współpracy z perforatorami

Podobnie jak przy czytnikach, bloki sterujące perforatorami działają identycznie, wobec tego podamy opis bloku sterowania perforatorem 1.

Działanie bloku perforowania stanowi jakby lustrzane odbicie programu sterowania czytnikiem, opisanym W punkcie 3.3.1. Tak więc program ten zawiera również w pamięci operacyjnej dwa bufory l6-słowowe oznaczone przez A i B. Na początku pracy systemu operacyjnego oba bufory zostają wyzerowane.

W tej sytuacji pojawienie się rozkazu drukowania rzadka taśmy (SLR2) spowoduje jedynie zapamiętanie rzadka (czyli zawartości mnożnika M) w buforze A. Nastopne rozkazy SLR 2 powodować będą również zapamiętywanie kolejnych rządków w buforze A, aż do wypełnienia bufora. Wówczas następuje zamiana nazw buforów **i** dalsze znaki wpisywane są również do bufora A (po zmianie nazwy). Jednocześnie, w chwili zapewnienia jednego z buforów uruchamiany jest program drukowania rządków taśmy zarejestrowanych w tym buforze. Program drukowania bada każdorazowo stan klucza KP na stoliku operatora — jego wciśnięcie stanowi zezwolenie na perforowanie. Brak takiego zezwolenia przy wypełnionym jednym z buforów powoduje zapalenie lampki alarmu LA.

Jeśli program główny powoduje szybsze wypełnienie buforów niż trwa ich opróżnienie związane zadrukowaniem to, podobnie jak przy czytaniu, wykonywanie programu użytkowego zostaje zawieszone aż do wydrukowania jednego z buforów.

Blok współpracy z drukarką wierszową

Struktura tego bloku jest stosunkowo prosta. Wprowadzane znaki w kodzie DW (znaki w kodzie M2 są zamieniane na kod DW) ładowane do 30-słowowego (120 znaków 6-bitowych) bufora w pamięci operacyjnej. Właściwym poleceniem drukowania jest rozkaz posuwu papieru (również o 0 linii) lub próba umieszczenia w buforze więcej niż 120 znaków.

Przed wykonaniem tego rozkazu zawartość bufora w pamięci operacyjnej przesyłana jest do pamięci buforowej drukarki.

Przed każdym wydrukiem lub posuwem papieru badany jest wskaźnik programowy zezwolenia na drukowanie, ustawiany poprzez naciśnięcie przycisku „Zgłoszenie operatora" ZO na drukarce. Żądanie drukowania wiersza lub wysuwu papieru przy braku zezwolenia na drukowanie powoduje zapalenie się lampki alarmu LA na stoliku operatora 1.

Blok współpracy z monitorem

W odróżnieniu od czytnika. perforatora czy drukarki, sterowanie monitorem, ze względu na jego zadania związane z drukowaniem informacji o przebiegu wykonywania programu oraz ewentualnym przyjmowaniu instrukcji od operatora, musi być tak zorganizowane, by polecenie drukowania choćby jednego znaku powodowano niezwłoczne jego wydrukowanie. Wynika stąd konieczność rezygnacji z tworzenia buforów, używanych w blokach sterowania czytnikami i perforatorami, co oczywiście nie wyklucza w ogóle stosowania buforów przy współpracy z monitorem, niemniej jednak sterowanie wypełnianiem buforów należeć musi do programu użytkowego.

Blok współpracy z monitorem realizuje dwie podstawowe transmisje: przesyłanie znaków z monitora do określonego obszaru pamięci operacyjnej oraz przesyłanie innego obszaru pamięci do monitora w celu wydrukowania. Odpowiednie polecenia (rozkazy SLR 8, SLR 18, SLR 9) inicjują jedynie rozpoczęcie przesyłania, które sterowane jest przez blok współpracy z monitorem. Program czytania, tj. przesyłanie z monitora do pamięci, posiada również ciekawa własność, która polega na możliwości zainicjowania procesu czytania dopiero po napisaniu przez operatora znaku — jeśli to nie nastąpiło, to operacja nie została zainicjowana, a tym samym monitor może przystąpić np. do drukowania.

Blok współpracy z monitorem interpretuje również rozkaz oczekiwania na zakończenie transmisji (SLR 20) oraz rozkaz badający, czy zainicjowana transmisja została już zakończona (SLR 21).

Wykonanie każdej operacji poprzedzone jest badaniem

programowego wskaźnika gotowości monitora. Jeżeli monitor (dalekopis) nie jest przygotowany do pracy, to operacja zostaje zawieszona aż do przygotowania monitora (co jest sygnalizowane przez operatora naciśnięciem przycisku ZO na pulpicie sterującym monitora – naciśnięcie powoduje zanegowanie stanu programowego wskaźnika gotowości). Jednocześnie zapala się lampka alarmu LA na stoliku I.

Blok współpracy z czytnikiem kart

Blok ten inicjuje proces czytania karty, przyjmuje przerwania nadchodzące po wczytaniu kolejnych kolumn, przesyła odczytane kolumny do pamięci, a także przyjmuje przerwania spowodowane błędnym odczytem. Blok posiada szczególnie prostą budowę, co wynika m.in. z tego, że wykonywanie rozkazu czytania karty (SLR 15) kończy się dopiero po wczytaniu karty, **a nie** po zainicjowaniu operacji, co ma miejsce np. przy monitorze.

Przed rozpoczęciem czytania każdej karty badany jest programowy wskaźnik gotowości czytnika kart (każde naciśnięcie przycisku „Zgłoszenie operatora" ZO neguje zawartość **tego** wskaźnika). **Jeżeli** czytnik kart nie jest przygotowany do pracy to operacja zostaje zawieszona aż do przygotowania czytnika kart. Jednocześnie zapala się lampka alarmu LA na stoliku I.

Blok współpracy z pamięcią bębnową

Program współpracy z pamięcią bębnową steruje transmisjami z pamięci bębnowej do operacyjnej i na odwrót. Transmisje te, w odróżnieniu od transmisji do/z pamięci taśmowej, realizowane są przez część centralną maszyny, co uniemożliwia jednoczesne wykonywanie innego programu.

Polecenie wykonania transmisji (SLR 3) powoduje przygotowanie odpowiedniej sekwencji rozkazów realizujących przesyłanie (pkt. 2.3.1), zapalenie wskaźnika pisania lub czytania bębna, a następnie powrót do programu użytkowego. Na 600μs przed rozpoczęciem transmisji pamięć bębnowa wysyła przerwanie do części centralnej, co powoduje ponowne wejście do bloku współpracy z pamięcią bębnową i rozpoczęcie przesyłania.

Blok **nie** interpretuje przerwania spowodowanego błędem bębna, w takim wypadku maszyna **jest** zatrzymywana.

Blok współpracy z pamięcią taśmową

Zadaniem bloku jest inicjowanie transmisji do/z pamięci taśmowej oraz inicjowanie pozostałych operacji taśmowych. Ze względu na połączenie pamięci taśmowej z operacyjną poprzez kanał pamięci taśmowej struktura omawianego bloku różni się w istotny sposób od pozostałych bloków. Istota zagadnienia polega na tym, że kanał wysyła sygnał przerwania po wykonaniu całej

transmisji, nie zaś jej fragmentu, jak to ma miejsce w pozostałych urządzeniach. Jednakie, z uwagi na dość duży stopień skomplikowania pamięci taśmowej oraz ilość czynności, które może ona wykonywać, blok współpracy z pamięcią taśmową jest dość obszerny.

Jak już wspomniano w pkt. 2.3.2 kanał taśmy magnetycznej jest przystosowany do dołączenia trzech jednostek taśmy magnetycznej (a przy pewnych modyfikacjach nawet 7). Stąd również blok współpracy z pamięcią taśmową jest przystosowany do sterowania praca 7 taśm, przy czym w danej chwili może pracować tylko jedna jednostka. Sprawa ta nie jest istotna, ponieważ m.c. ZAM 41 posiada tylko l jednostkę taśmy magnetycznej.

Zainicjowanie operacji taśmy (rozkazem SLR 7) powoduje wpisanie do odpowiednich rejestrów kanaru parametrów operacji, **po** czym sterowanie zostaje oddane programowi użytkowemu.

W przypadku poprawnego zakończenia transmisji blok współpracy z pamięcią taśmową może zainicjować następną transmisję. Przyjmuje **się**, **że czas** operacji odwijania taśmy do początku nie powinien przekroczyć 300 sek – przekroczenie tego czasu wskazuje na uszkodzenie jednostki taśmy magnetycznej.

Niepoprawne zakończenie operacji zapisu lub odczytu powoduje trzykrotne jej powtórzenie i jeśli nie da to wyniku pozytywnego, to w przypadku operacji czytania rezygnuje się z dalszych prób odczytu, zapalając tylko odpowiedni wskaźnik programowy błędu, zaś w przypadku operacji pisania kasuje się pewien odcinek taśmy, a następnie za odcinkiem skasowanym, dokonuje się ponownie trzykrotnej próby zapisu **itd.**

Jeżeli w odpowiedzi na wydane polecenie kanał poinformuje blok współpracy z pamięcią taśmową o braku gotowości taśmy, to dla wszystkich operacji, z wyjątkiem odwijania do początku, odpowiedź jest interpretowana jako uszkodzenie jednostki taśmy magnetycznej.

Po sygnalizacji korca taśmy, blok współpracy dopuszcza jeszcze możliwość zapisu lub odczytu l bloku taśmowego. Niedozwolone są natomiast operacje: kasuj i odwiń do przodu.

Identyfikacja uszkodzenia taśmy magnetycznej nie powoduje przerwania aktualnie wykonywanego programu użytkowego. Przeciwnie, jest on wykonywany dopóty, dopóki program użytkowy nie zapyta o zakończenie poprzednio zainicjowanej operacji lub spróbuje zainicjować nowa operację. Dopiero wówczas wykonywanie programu użytkowego zostaje zakończone, przy czym drukowany jest tekst AWARIA TAŚMY.

Konfiguracje M.C. ZAM 41

Maszyna cyfrowa ZAM 41 wyposażona jest w dwa translatory języków: PJP, który stanowi przedmiot niniejszego opracowania, oraz ZAM-Algolu. Translatory te zgrupowane są w dwóch zestawach różniących się wielkością zajmowanej pamięci bębnowej, systemem operacyjnym i niektórymi parametrami.

Zestaw I zawiera skrócony system operacyjny (bez bloków współpracy z pamięcią taśmową, monitorem, czytnikiem kart i stolikiem operatora II), translator języka PJP oraz translator języka ZAM-Algol. Zestaw ten zajmuje na bębnie 16 384 skowa.

Zestaw II zawiera pełny system operacyjny oraz translator języka PJP. Zestaw ten zajmuje na **bębnie** 8192 słowa.

Prócz tego każdy z zestawów zawiera także podprogramy standardowe definiujące operacje drukowania liczb, tekstów itp., czytania oraz operacje arytmetyczne zmiennoprzecinkowe.

System Operacyjny SO-41 dla ZAM-41

Informatyka 3/1971

System ten został zakończony w Instytucie Maszyn Matematycznych w roku l968. Jest to najpełniejszy system operacyjny ze wszystkich opracowanych dotychczas w Polsce. Spełnia on różnorodne zadania.

Najważniejszym z nich jest zwiększenie przepustowości maszyny poprzez wykorzystanie jej części centralnej w systemie wieloprogramowym. Dotychczas zrealizowano pracę dwu-programową przy sztywnym, chociaż dowolnym podziale maszyny na dwa zestawy.

System SO/41 zapewnia też prostą obsługę operatorską przez traktowanie maszyny jako urządzenia do przetwarzania kolejnych zleceń, przy czym sposób wykonania każdego zlecenia opisany jest w czołówce za pomocą Języka Operacyjnego Maszyny (JOM).

Należy tu dodać, że zasady obsługi wszystkich urządzeń zewnętrznych są podobne, a wszystkie dane wyjściowe opatrzone łatwo rozróżnialnymi etykietami ustalającym i ich przynależność do przetwarzanych zleceń.

System zapewnia również łatwy dostęp do elementów systemu oprogramowania, zawartych w Bibliotece Systemu, umieszczonej na taśmie magnetycznej oraz na bębnie. Elementami biblioteki są zbiory identyfikowane przez nazwę. System SO/41 zapewnia łatwą adaptację programów do różnych urządzeń zewnętrznych. Jest to zrealizowane przez standaryzację operacji wejścia i wyjścia oraz odwoływanie się do tych urządzeń za pomocą symboli, których znaczenie określa się poza programem przy użyciu języka systemu operacyjnego.

System SO-41 zawiera też w sobie System Generowania Konkretnych SO, dający możliwość automatycznego generowania wersji Systemów dostosowanych do różnych konfiguracji maszyny. SO/41 może być łatwo rozbudowany przez dołączanie nowych programów do Biblioteki Systemu lub też nowych podprogramów wejścia i wyjścia do supervisora. W szczególności do SO/41 można dołączyć stosunkowo łatwo translatory nowych języków.

SO-41 umożliwia łatwą adaptowalność maszyny do zmiennych wymagań eksploatacyjnych danego ośrodka. Osiąga się to przez umożliwianie podjęcia szeregu decyzji o trybie pracy systemu w

momencie jego startu. W szczególności można wybrać odpowiednią wersję supervisora, określić rozmieszczenie elementów systemu w pamięciach pomocniczych (taśmie magnetycznej lub bębnie) oraz określić podział maszyny na zestawy.

W czasie pracy SO/41 prowadzi rejestrację pracy oraz rejestrację wykorzystywanych szpul magnetycznych. Supervisor zajmuje na stałe 3—5 K słów pamięci operacyjnej (24-bitowych) oraz 6-12 K takich słów na bębnie.

(KB)

Oficjalna dokumentacja techniczna dostarczana użytkownikom składała się z kilku tomów formatu A4. Dla systemu SO-41 była to 'srebrna seria', dla systemu SO-141 była to 'niebieska seria'. Nazwa pochodzi od koloru plastikowych okładek, w które dokumentacja była oprawiona. Niestety, nie udało się na razie zlokalizować żadnej kopii tej dokumentacji.

System operacyjny SO-141

(Jan Wierzbowski - ZAM-41 Kompendium Oprogramowania)

Programy systemu operacyjnego SO-141 (zwanego w skrócie SO) zostały napisane w Języku PJEG i w tej postaci zapisane za pomocą języka SMAD na taśmach magnetycznych, w tzw. postaci źródłowej. Za pomocą specjalnych środków przekształcono postać źródłową na tzw. postać binarną. W czasie tego procesu przekształcania, zwanego generowaniem, ustawiono niektóre parametry tworzonego systemu.

Jednym z głównych elementów systemu operacyjnego jest tzw. "Superwizor", tj. program zarządzający obsługą urządzeń wejścia i wyjścia i realizujący podział czasu między poszczególne programy, jednocześnie załadowane do pamięci operacyjnej.

System operacyjny w postaci binarnej jest zapisany na taśmie magnetycznej, zwanej w dalszym ciągu taśmą systemu. Przed rozpoczęciem pracy system operacyjny zostaje załadowany na bęben, a jego część tzw. rezydent - do pamięci operacyjnej. Elementy przechowywane na bębnie są sprowadzane do pamięci operacyjnej wtedy, gdy mają być wykonywane.

Powyższy opis dotyczy systemu operacyjnego SO-141. Wcześniej istniał system SO-41,który jest jeszcze czasem eksploatowany, ale już od dość
dawna nie jest rozwijany.

Biblioteka systemu oprogramowania

Biblioteka systemu oprogramowania, zwana w skrócie biblioteką systemu, składa się ze zbiorów, które mogą być translatorami, programami użytkowymi lub tekstami. Każdy z wymienionych typów zbiorów ma określone własności.
Elementy biblioteki systemu są zapisane w postaci wynikowej (tj. binarnej) lub do niej zbliżonej na taśmie systemu, tzn. na tej samej
taśmie magnetycznej co system operacyjny. Natomiast wersje źródłowe, zwane też plikami źródłowymi tych elementów, są zapisane w różnych
językach na różnych taśmach magnetycznych.
W skład biblioteki systemu wchodzą:

- translatory języków programowania,
- funkcje języków,

- programy manipulacyjne,
- niektóre programy użytkowe,
- testy.

Każdy element biblioteki systemu ma swoją nazwę, przez którą jest wywoływany.

Edycje SO-141

System operacyjny wraz z biblioteką są stale rozwijane. Od czasu do czasu ukazują się kolejne edycje taśmy systemu. Są one oznaczone trzema literami, stanowiącymi tzw. numer edycji systemu. W chwili pisania tych słów przygotowywana jest edycja BCW, co oznacza, że od wprowadzenia SO-141 wydano przeszło 50 edycji. Generalną zasadą przestrzeganą przy wprowadzaniu nowej edycji jest zachowanie wszystkich możliwości starej edycji. W stosunku do poprzedniej edycji systemu nowa edycja może zawierać nowe elementy oprogramowania, nie zawiera natomiast błędów dostrzeżonych w starej edycji.

Zapewnia to działanie wszystkich dotychczas działających programów. Może się jednak zdarzyć, że wskutek rozbudowy systemu, zmniejsza się obszar pamięci operacyjnej lub bębnowej przeznaczony dla programu.

Wieloprogramowość

W trakcie generowania systemu operacyjnego określa się liczbę programów, jakie mogą być wykonywane jednocześnie, a ściślej mówiąc,które jednocześnie znajdują się w pamięci operacyjnej i są wykonywane na przemian, pod nadzorem systemu operacyjnego. Ze względu na stosunkowo małą pamięć operacyjną oraz niewielką liczbę urządzeń zewnętrznych, na ogół generuje się systemy dopuszczające maksymalnie pracę dwuprogramową. Podział maszyny na zestawy obsługujące poszczególne programy jest przeważnie następujący:

a) **Reżim A**: zestaw O zawiera połowę pamięci operacyjnej i bębnowej oraz połowę każdego rodzaju urządzeń, zaokrągloną w razie potrzeby w górę (np. gdy są trzy jednostki taśmy magnetycznej, to zestaw O zawiera dwie z nich); pozostała część pamięci operacyjnej i bębnowej oraz pozostałe urządzenia są przydzielane do zestawu 1;

b) **Reżim B**: zestaw 0 zawiera 4 Jednostki taśmy magnetycznej, zestaw 1 - pozostałej pamięć operacyjna i bębnowa oraz pozostałe urządzenia zewnętrzne są tak dzielone, jak w reżimie A;

c) **Reżim C:** całość maszyny obejmuje zostaw 0, co oznacza pracę jedno-programową (nie licząc systemu operacyjnego).
O ile wybór dopuszczalnych reżimów następuje w momencie generowania systemu operacyjnego, to wybór właściwego reżimu pracy następuje w
momencie jego startowania.

Adresowanie pamięci operacyjnej jest niezależne od tego, czy program działa samodzielnie, czy dzieli pamięć z innymi programami. Wynika to z automatycznego dodawania w określonych warunkach, zawartości rejestru dolnej blokady do adresu rozkazu. Niezależne adresowanie pamięci bębnowej oraz niezmienny sposób odwoływania się do urządzeń zewnętrznych realizuje system operacyjny. Dzięki temu programista, pisząc program musi się jedynie troszczyć o ilość dostępnych słów w pamięci operacyjnej i bębnowej oraz liczbę urządzeń zewnętrznych,natomiast nie interesuje go, w którym zestawie program będzie działał.

Język operacyjny maszyny (JOM)

JOM (język operacyjny maszyny) służy do opisywania wykonywanych na maszynie problemów. Jednostką przetwarzania, tzn. taką jednostką, którą można oddzielnie operować, jest tzw. problem. Problem dzieli się na kroki, z których każdy oznacza- bądź wykonanie programu stanowiącego element biblioteki systemu, bądź programu będącego wynikiem działania translatora. W języku tym opisuje się zadania wchodzące w skład problemu oraz podaje się inne informacje o problemie, takie jak tytuł problemu, urządzenia, podprogramy i dekodery stosowane dla symbolicznych wejść i wyjść, ograniczenia czasu oraz wyjścia itp.

Parametry wykonywanego problemu opisuje się w specjalnym języku, zwanym Językiem Operacyjnym Maszyny (JOM). Opis problemu stanowi tzw. czołówkę, która musi być wyperforowana na taśmie papierowej w kodzie M-2 i musi być wprowadzona na początku problemu z pierwszego czytnika
taśmy papierowej w zestawie, w którym dany problem ma być wykonywany. Czołówka składa się ze zdań, które mają następujące znaczenie:

a) zdanie **PROBLEM** określa kolejne kroki, które w ramach danego problemu będą wykonywane; w szczególności jednym z kroków może być JOM, co oznacza, że po wykonaniu poprzednich kroków należy powtórnie wczytać czołówkę, która może zmieniać poprzednio ustalone parametry problemu;
b) zdanie **WEJSCIA** (Programów, Translatorów) określa symboliczne wejście, stosowane w danym problemie;

c) zdanie **WYJSCIA** (Programów, Translatorów) określa symboliczne wyjście, stosowane w danym problemie;
d) zdanie **OGRANICZ** uniemożliwia zbyt długie działanie programów w ramach danego problemu i wyprowadzanie zbyt dużej liczby wyników; ograniczenia te stosuje się przede wszystkim wówczas, gdy program nie jest jeszcze uruchomiony i istnieje obawa jego "zapętlenia";
e) zdanie **TYTUL** powoduje wypisanie określonego tekstu na początku wyprowadzanych za pomocą symbolicznego wyjścia O wyników.

Język PJP

Język PJP (Podstawowy Język Programowania) jest niewątpliwym poprzednikiem języków wymienionych niżej. Był on, prawdopodobnie, pierwszą generacją implementacji języka maszyny umożliwiającą dalszy rozwój oprogramowania maszyn rodziny ZAM. Pierwotnie opracowany dla maszyny ZAM-21.

Języki PJES i PJEG

Języki PJES (Podstawowy Język Symboliczny) i PJEG (Podstawowy Język Generowania) są językami symbolicznymi, w których każdy rozkaz odpowiada jednemu słowu maszyny. Języki te umożliwiają dostęp do każdego słowa lub nawet bitu, zarówno w pamięci operacyjnej, jak i bębnowej.

Obydwa Języki dopuszczają różne sposoby adresowania symbolicznego, przy czym PJEG, który jest rozszerzeniem języka PJES, stwarza w tym zakresie szczególnie dużo udogodnień. Liczne dyrektywy (szczególnie w języku PJEG) umożliwiają optymalną gospodarkę pamięcią, wypisywanie informacji o programie wynikowym, omijanie części programu w czasie translacji itp.

Język PJEG umożliwia tworzenie makrodefinicji i wykonywanie makrorozkazów i własności te ułatwiają tworzenie programów wynikowych.

Język SAS

Język SAS (Sytem Adresów Symbolicznych), formalnie zwany Makro SAS. Tak jak powyższe języki stosowane były w produkcji oprogramowania, stopniowo przechodzącej z języka PJES na język PJEG tak SAS przeznaczony był do pisania programów użytkowych. Jak poprzednio w przypadku maszyny ZAM-2, programy w języku SAKO mogły być zintegrowane z programami w języku SAS. Język jest wyposażony w rozbudowany aparat wbudowanych makrodefinicji i odpowiadających im makroodwołań. Aparat ten

został pomyślany jako narzędzie umożliwiające łatwe korzystanie z obszernej biblioteki funkcji języka. Ponadto SAS posiada własny system gospodarki pamięcią operacyjną, co w połączeniu z podziałem programu na sekcje, w dużej mierze ułatwia programowanie.

Język SMAD

Język SMAD (System Magazynowania i Aktualizacji Dokumentów) znany również pod nazwą POPR lub POPRAWEK służy do magazynowania programów na taśmie magnetycznej. Programy te, traktowane jak teksty, mogą być uzupełniane i poprawiane, co w istotny sposób upraszcza proces programowania. Ponadto SMAD umożliwia wykonanie dodatkowych czynności jak np. sporządzenie dokumentacji programu.

Języki BIB1 i BIB2

Języki BIB1 i BIB2 (Bibliotekarz 1 i 2) służą do wprowadzania nowych zbiorów do biblioteki systemu. Dla programisty nieprzewidującego włączenia programów użytkowych do biblioteki systemu, języki te nie mają zastosowania.

Translatory WEBIN, WYBIN i WYSE

Translatory WEBIN (wejścia binarne1) i WYBIN (wyjście binarne) służą do operowania tzw. taśmą binarną, tj. taśmą z programem całkowicie lub częściowo przetłumaczonym. Taśmy binarne stosuje się przede wszystkim dla programów napisanych w językach o stosunkowo długim procesie translacji (np. SAS, COBOL). Taśma taka jest wprowadzana do maszyny znacznie szybciej, a zapisane na niej sumy kontrolne zapewniają ponadto prawidłowość jej wprowadzenia.

Ponieważ translator języka SAS prowadzi własną gospodarkę pamięcią bębnową, rozmieszczenie programu na bębnie jest uzależnione od aktualnej konfiguracji systemu sprowadzonego na bęben. W tej sytuacji programy binarne z języka SAS mogłyby nie działać prawidłowo.

Aby temu zaradzić, wprowadzono translator WYSE, który przed wypisaniem taśmy binarnej przenosi program wynikowy na ustalone miejsce oraz dołącza do niego sekcję startową. Po wyprowadzeniu taśmy binarnej i wprowadzeniu jej ponownie zaczyna działać sekcja startowa, która z powrotem przenosi program na właściwe miejsce.

Język SMAO

Język SMAO (System Magazynowania Opisów) służy do przechowywania opisów poszczególnych translatorów, programów itp. na taśmie magnetycznej.
Zmagazynowane opisy mogą być bieżąco uzupełniane i poprawiane, co gwarantuje ich aktualność, nawet jeśli zmiany w dokumentacji następują często. Za pomocą Języka SMAO produkowana jest tzw. informacja ekspresowa dotycząca maszyny ZAM-41, a obejmująca te fragmenty opisów oprogramowania, które jeszcze nie zostały opublikowane.

Pogramy użytkowe

Oprócz wymienionych wyżej translatorów w skład biblioteki systemu wchodzi kilka programów użytkowych, które można podzielić na następujące trzy kategorie:

- programy manipulacyjne,
- programy do przetwarzania danych,
- testy.

Programy manipulacyjne

Programami manipulacyjnymi nazwano często używane programy, służące do wykonywania pomocniczych czynności typu organizacyjnego. Czynności te dotyczą na ogół operowania taśmą magnetyczną, sporządzania dokumentacji, uruchamiania programów itp.

Programy manipulacyjne są podzielone na cztery grupy: MAN, MAN1, VMAN oraz inne. W skład tych grup m.in. wchodzą (podane poniżej nazwy, chociaż żargonowe, zostały już zaakceptowane przez środowisko programistów i użytkowników i dlatego są przytoczone w oryginalnym brzmieniu):

a) **Grupa MAN:**

- EDYTOR-PJES (CIA) - dzieli programy napisane w języku PJES na strony, zgodnie z wymaganiami SMAD, *(KB - ciachator)*
- SPRAWDZATOR (SPR) - porównuje i wykrywa niezgodności między dwoma taśmami z tym samym programem napisanym w języku PJES,
- WYPISYWANIE TM (WTM) - wypisuje określone bloki z taśmy magnetycznej,
- WYSZUKIWANIE TEKSTU (SZUK) - wyszukuje określoną sekwencję znaków w tekście zbudowanym zgodnie z wymaganiami Języka SMAD.

b) **Grupa MAN1:**

- WYPISYWACZ SPISU STRON SMAD (STR) - wypisuje spis stron z tekstu zapisanego w języku SMAD na taśmie magnetycznej,
- POWIELACZ TMS (TMS) - powiela taśmę magnetyczną systemu,
- WYPISYWACZ SMAD (WYPS) - wypisuje kolejne strony tekstu zapisanego w języku SMAD na taśmie magnetycznej,

- DOKUMENT (DOK) - sporządza dokumentację tekstu wejściowego,
- METRYKOWACZ TAŚMY MAGNETYCZNEJ (MKZ) - metrykuje taśmę magnetyczną zgodnie z przyjętym standardem,
- WYPISYWACZ SPISU STRON SMAO (STR8) - wypisuje spis stron z tekstu zapisanego w języku SMAO na taśmie magnetycznej;

c) **Grupa VMAN:**

- METRYKOWACZ TM W STANDARDZIE (VMKZ) - metrykuje taśmę magnetyczną zgodnie z przejętym standardem,
- UNIWERSALNY PROGRAM WYDAWNICZY (VWYD) - redaguje tekst, wypisując go na różnych urządzeniach,
- METRYKOWACZ TM W STANDARDZIE SMAD (VMEP VMKD) - zapisuje na początku lub w miejscu zatrzymania taśmy magnetycznej znacznik końca dokumentu, zgodnie z wymaganiami języka SMAD,
- SCALATOR TAŚM ZAPISANYCH W STANDARDZIE SMAD (VSCP) - łączy dwie taśmy magnetyczne z zapisanymi tekstami w języku SMAD,
- WYPISYWACZ SPISU stron TM zapisanej w STANDARDZIE SMAD (VWYP) - wypisuje spis stron z taśmy magnetycznej zapisanej w języku SMAD,
- PERFORATOR ZAWARTOŚCI TM ZAPISANEJ W STANDARDZIE SMAD (VPPO) - perforuje zawartość taśmy magnetycznej zapisanej w języku SMAD,
- WYPISYWACZ TAŚMY SYSTEMU (VWTS) - powiela lub sprawdza taśmę systemu, wypisuje katalogi, perforuje wybrane zbiory z taśmy systemu,
- KONTROLA ZAPISU PLIKU (VKZP) – kontroluje i reprodukuje pliki zapisane na taśmie magnetycznej,
- ZAPIS BINARNEJ NA MAGNETYCZNEJ (VZBT) - zapisuje informacje z binarnej taśmy papierowej na taśmie magnetycznej zgodnie z wymaganiami języka SMAD;

d) **Inne:**

- EDYTOR NA TAŚMĘ PAPIEROWĄ - SAS-EOL (PTSE) - dzieli programy napisane w języku SAS-EOL na strony, zgodnie z wymaganiami języka SMAD; wyniki wypisuje na taśmie papierowej,
- EDYTOR NA DRUKARKĘ - SAS-EOL (DWSE) - dzieli programy napisane w języku SAS-EOL na strony, zgodnie z wymaganiami języka SMAD; wyniki wypisuje na drukarce wierszowej, uzupełniając je skorowidzem etykiet,
- SPRAWDZATOR - SAS-EOL (SPSE) – porównuje i wykrywa niezgodności między' dwiema taśmami z tym samym programem, napisanym w języku SAS-EOL,

- EDYTOR-ALGOL (EDAL) - dzieli programy napisane w języku ALGOL na strony, zgodnie z wymaganiami języka SMAD; wyniki wypisuje na perforatorze oraz na drukarce wraz e skorowidzem zadeklarowanych nazw,
- EDYTOR-COBOL (EDCO) - dzieli programy napisane w języku COBOL na strony, zgodnie z wymaganiami języka SMAD; wyniki wypisuje na taśmie papierowej.

Programy do przetwarzania danych

W bibliotece systemu znajduje się grupa programów parametryzowanych, znanych pod nazwą OPUS, służących do wykonywania niektórych typowych czynności z zakresu przetwarzania danych. Najważniejszymi programami z tej grupy są:

- PP01 - czyta dane z kart perforowanych, kontroluje ich poprawność, poprawne dane zapisuje na taśmie magnetycznej,
- PP03 - czyta dane z taśmy perforowanej, kontroluje ich poprawność, poprawne dane zapisuje na taśmie magnetycznej,
- PF09 i PP12 - czytają dane z taśm magnetycznych, przekształcają je i wynik zapisują na taśmach magnetycznych,
- PP10 - kontroluje zbiory na taśmie magnetycznej,
- PP11 - powiela zbiory na taśmie magnetycznej,
- PP13 - czyta dane z taśmy magnetycznej,redaguje je, wykonuje odpowiednie operacje arytmetyczne i wyniki wypisuje na drukarce wierszowej,
- PP15 - sortuje dane zapisane na taśmie magnetycznej oraz łączy dwa posortowane zbiory danych.

(J. Dańda)

System Opus

Pakiet programów parametryzowanych Opus byt wynikiem prac teoretycznych nad metodami przetwarzania (technologia) danych do zastosowań administracyjno-gospodarczych.

Prace nad pakietem Opus rozpoczęto w 1967 r., a więc wtedy, gdy idea programowania parametryzowanego dopiero zaczynana zyskiwać popularność. Przy opracowywaniu pakietu przyjęto następujące założenia:

- poszczególne programy powinny wykonywam typowe czynności z zakresu przetwarzania danych;
- działanie każdego programu powinno zależeć od parametrów, wczytywanych przez program na początku działania;
- struktura parametrów dla równych programów powinna byt możliwie jednolita;
- dla programu o prostych funkcjach parametry powinny być proste, w miarę komplikowania się funkcji programu, struktura parametrów może się stawać bardziej złożona;
- w pierwszej wersji programów nalewy ograniczyć się do zbiorów sekwencyjnych i działań metoda interpretacji - w

dalszych wersjach należy rozważyć metodę generowania i dziamanie na zbiorach niesekwencyjnych.

Jako czynności typowe dla przetwarzania danych przyjęto:

- kontrolę danych na maszynowych nornikach informacji (papierowych);
- konwersję danych z papierowych nośników informacji na taśmę magnetyczną;
- wielotaśmową konwersja zbiorów;
- sortowanie, łączenie;
- redagowanie i drukowanie wyników,
- czynności pomocnicze jak np. zakładanie, kontrola oraz powielanie zbioru danych.

Realizacje pakietu Opus zakończono w 1971 r. Składa się on z 13 programów obejmujących w sumie ok. 40 tys. rozkazów. Pakiet stosuje się m.in. w Systemie Ewidencji i Rozliczeń Wyrobów Gotowych opracowanym dla Zakładów Przemysłu Odzieżowego "Cora". System ten posiada trzy ewidencje, aktualizowane w cyklu
miesięcznym i tworzy ponad 20 sprawozdań. Składa się on z ponad 80 programów, z których tylko 3 trzeba byto napisać specjalnie dla tego systemu, a pozostałe 90% wygenerowano przy użyciu systemu Opus.

Zastosowanie programów Opus w tym systemie byto praktycznym sprawdzeniem ich przydatności, umożliwiało jednocześnie istotne skrócenie łącznego czasu projektowania i wdrażania systemu. Praktyka wykazana, że przygotowanie parametrów i sprawdzenie działania programu pochodzącego z pakietu Opus zajmuje ok. 5 dni roboczych. Łatwo też można rozbudowywać system i modyfikować go, gdy potrzeby przedsiębiorstwa dyktuje konieczność wprowadzenia zmian.

Testy

Włączone do biblioteki systemu testy tworzą tzw. system testów K-500. Składa się on z dyrygenta oraz z autonomicznych testów, sprawdzających poszczególne urządzenia. Ponieważ K-500 pracuje pod nadzorem systemu operacyjnego zakłada się, że w pewnym zakresie maszyna jest sprawna, szczególnie część pamięci operacyjnej, podstawowe operacje arytmetyczne i logiczne itp.

K-500 sprawdza więc tylko część pamięci operacyjnej i bębnowej oraz urządzenia zewnętrzne. Ponieważ K-500 pracuje pod nadzorem systemu operacyjnego, sprawdzenie urządzeń zewnętrznych nie jest zupełne. W celu pełniejszego sprawdzenia maszyny należy zastosować istniejące testy działające poza systemem operacyjnym.

W skład systemu testów K-500 wchodzą:

- test arytmometru zmiennego przecinka,
- test pamięci operacyjnej,
- test pamięci bębnowej,
- test pamięci taśmowej,
- test perforatora i czytnika taśmy papierowej,
- test czytnika taśmy papierowej,
- test drukarki DW-1,
- test drukarki DW-2,
- test czytnika kart.

Język LOGOL dla maszyny ZAM-2

(Jerzy Mysior)

LOGOL – była to jedna z nielicznych publikacji przez renomowane wydawnictwa zachodnie dot. opracowań wykonanych w IMM (przed odejściem S. Waligórskiego na Uniwersytet).

Późniejsza była publikacja w „Ada Letters, November/December 1989” pp. 85-90 Jerzy Mysior, Andrzej Paprocki „An Eight-Bit Character Set in Ada Programs” związana z realizowanym przez IMM kompilatorem szerokiego podzbioru języka Ada dla komputerów SM EMC (PDP 11).
Nb. ciekawostką było, że będąc sprzedawany z komputerami przez fabrykę ERA krążył w sąsiednich krajach również w wersji pirackiej, wysoko uznawany przez instytucje komputerowe.

Język i translator SAKO dla maszyny XYZ oraz maszyn ZAM 2, ZAM 21 i ZAM 41

Język SAKO jest opracowaniem w znacznej mierze oryginalnym, a pierwszy jego translator uruchomiono w roku 1962. Było to opracowanie w Polsce pionierskie i przez wiele la t w dziedzinie języków proceduralnych i w tej skali — jedyne.

Następne translatory SAKO uruchomiono dla maszyn ZAM 21 oraz ZAM 41. Łącznie SAKO znalazło w Polsce dość szerokie rozpowszechnienie, aczkolwiek obecnie jest już w znacznej mierze wyparte przez język ALGOL 60.

Język SAKO przystosowany był pierwotnie do maszyn stałoprzecinkowych, przy czym programista miał pełną możność sterowania skalą obliczeń. W realizacjach tego języka dla ZAM 21 oraz dla ZAM 41 obliczenia można również wykonywać w zmiennym przecinku.

Rozkazy SAKO mogą też być przeplatane z rozkazami maszyny, zapisanymi w języku maszyny SAS dla ZAM 2 przy zachowaniu tych, samych identyfikatorów.

SAKO przewiduje podział program u na rozdziały, z których każdy może być kompilowany oddzielnie.

Wprowadzone też zostały operacje bulowskie na słowach binarnych maszyny. Ponadto SAKO posiada prawie wszystkie operacje występujące w FORTRAN II, zakodowane w polskiej wersji językowej, przy czym niektóre cechy tego języka stawiają go na równi z FORTRANEM IV.

Translator ALGOL 60 na ZAM 21

Zakończony w Instytucie Maszyn Matematycznych w roku 1966, jest to translator prawie pełnej wersji ALGOLu z niewielkimi tylko ograniczeniami. Translator jest 5-przebiegowy i wraz z funkcjami i procedurami standardowymi zajmuje około 8000 rozkazów.

W translatorze tym na uwagę zasługuje podział programu wynikowego na segmenty bez wnikania w jego strukturę. Program wynikowy zapisany jest na bębnie; wymiana segmentów między bębnem a pamięcią operacyjną odbywa się według oryginalnej metody, która zdała praktyczny egzamin.

Translator ALGOL 60 dla ZAM 41

Translator ten został zakończony w roku 1968 w Instytucie Maszyn Matematycznych. Jest to translator prawie pełnej wersji ALGOL 60, wzorowany na Wheatstone Compiler, opracowanym dla maszyny angielskiej KDF9.

Tłumaczenie programu źródłowego na program wynikowy odbywa się w dwóch przebiegach. Program wynikowy zapisany jest w języku makrorozkazów
wykonywanych przez interpreter. Zarówno program wynikowy, jak i stos podzielone są na segmenty po 256 słów i umieszczone w pamięci bębnowej. Wymianą segmentów pomiędzy pamięcią bębnową a operacyjną steruje interpreter.

Język ALGOL 60 uzupełniony został przez dogodne procedury wejścia-wyjścia, zgodne ze standardami ALGAMS.

Translator języka COBOL na ZAM 41

Translator ten największy ze względu na liczby rozkazów i stopień złożoności ze wszystkich translatorów wykonanych dotychczas w Polsce zakończono w Instytucie Maszyn Matematycznych w roku 1970. Jest on dziesięcioprzebiegowy i zawiera około 18 000 rozkazów.

W skład przyjętej reprezentacji Języka wchodzą następujące elementy standardowego opisu języka COBOL 1:

- jądro poziom 2,
- dostęp sekwencyjny poziom 2,

- sortowanie poziom 1,
- segmentacja poziom 1.

Jak widać więc jest to dość duży podzbiór języka COBOL. Do wzorcowego opisu języka COBOL dołączono szereg zwrotów deklaracyjnych, dzięki którym umożliwiono przetwarzanie informacji zapisanej na nośnikach zewnętrznych w sposób niepozycyjny. Takie rozwiązanie jest szczególnie wygodne w przypadku stosowania taśmy perforowanej jako jednego z nośników. W trakcie wczytywania do pewnego pola pamięci, informacja niepozycyjna jest automatycznie przekształcana do postaci zgodnej z opisem tego pola w programie.

W realizacji translatora warto zwrócić uwagę na trzy sprawy:

a) Organizacja danych na taśm ach magnetycznych została przyjęta zgodnie z zaleceniem ISO.

b) Duża część translatora napisana została w języku EOL, dzięki czemu można było precyzyjnie zdefiniować języki wejściowe i wyjściowe poszczególnych przebiegów, szybko napisać i uruchomić programy przebiegów oraz sporządzać czytelną dokumentację translatora.

c) Programista może posługiwać się językiem assemblera SAS, którego instrukcje mogą być włączane w dowolnych miejscach **PROCEDURE DIVISION**, przy czym można się jednocześnie odwoływać do zmiennych opisanych w **DATA DIVISION**.

CEMMA — Język modelowania symulacyjnego procesów ciągłych

Zarówno język, jak i translator zostały utworzone w Zakładzie Sterowania Instytutu Maszyn M atematycznych dla maszyny ZAM 41. Translator oddany został do eksploatacji w pierwszym kwartale 1970 roku.

Język CEMMA 2 (Cyfrowe Modelowanie Maszyny Analogowej) służy do symulowania cyfrowego procesów ciągłych, tradycyjnie rozwiązywanych na maszynach analogowych. Uniwersalność techniki cyfrowej umożliwiła znaczne rozszerzenie zestawu operatorów (jest ich 39). M. in. włączono do języka takie operacje nietypowe dla maszyn analogowych jak np. funkcje nieliniowe $\sqrt{x}$ oraz log(x), elementy logiczne, funkcje zadane tablicą itp.

W jednym zadaniu może występować 500 operatorów, co także przewyższa możliwości stosowanych obecnie maszyn analogowych. Ponadto wprowadzono do języka procedurę optymalizacji, która

umożliwia optymalizację statyczną i dynamiczną badanych układów.

Programy w języku CEMMA 2 wprowadzane są do maszyny na taśmie perforowanej lub na kartach. Wyniki wypisywane są na drukarkę w postaci tablic albo w postaci graficznej.

EOL — Język do przekształcania symboli

Język EOL jest językiem do przekształcania symboli i był już opracowany w kilku kolejnych wersjach. EOL-1 oraz EOL-2 zostały opracowane koncepcyjnie w latach 1965—66 w Instytucie Maszyn Matematycznych.

EOL-2 został zrealizowany w roku 1966 na maszynie ZAM 41. EOL-3 opracowany -został w roku 1967 na Uniwersytecie ILLINOIS oraz zrealizowany w roku 1968 na maszynach IBM 7094, oraz IBM 360.

Następujące cechy są charakterystyczne dla wszystkich wersji języka EOL:

- są to języki uniwersalne do przetwarzania symboli, przystosowane jednak przede wszystkim do pisania translatorów ;
- są koncepcyjnie proste i łatwe do nauczenia i realizacji;
- zawierają operacje niskiego szczebla, a jednocześnie pozwalają użytkownikowi na jego rozszerzenia w dowolnym kierunku za pomocą zarówno podprogramów, zapisanych w języku EOL lub też w języku maszyny.

Programy napisane w języku EOL prowadzą do sprawnych programów wynikowych.

Realizacja powyższych punktów została ułatwiona tm. in. tym , że języki EOL mają zarówno środki dla przetwarzania krótkich, lecz złożonych informacji w pamięci operacyjnej maszyny, jak również środki do przetwarzania plików umieszczonych w pamięci masowej maszyny.

Doświadczenia zdobyte do tej pory potwierdzają użyteczność EOLu przy pisaniu translatorów. W szczególności za pomocą EOL-2 napisano znaczną część translatora języka COBOL,

Języki EOL są stosowane w celu kształcenia w zakresie wiedzy komputerowej zarówno w Polsce, jak i na niektórych uniwersytetach amerykańskich.

Język GPSS

Język GPSS (General Purpose System Simulator wersja na maszynie ZAM 41, znana również pod nazwą MODESI) jest wzorowany na rozpowszechnionym za granicą, szczególnie w USA, języku o tej samej nazwie. GPSS służy do symulacji różnych procesów dyskretnych, zależnych od przypadkowych lub zdeterminowanych zdarzeń zewnętrznych.' Wynikiem symulacji są różne dane statystyczne o zachowaniu się badanego procesu. W trakcie działania symulacji zapewniona jest wygodna komunikacja operatora z systemem.

Język ASTEK

Język ASTEK (Analiza Statystyczna Eksperymentów) służy do statystycznego opracowania zbiorów danych eksperymentalnych,uzyskiwanych z różnego rodzaju doświadczeń technicznych,przemysłowych, laboratoryjnych itp.
W szczególności w języku tym można testować statystycznie zbiory danych, liczyć średnie, wariancje i dystrybuanty, przeprowadzać analizę regresji i korelacji, wyznaczać gęstość widmową itp.

Języki BIGRAF i PLEXOL

Języki BIGRAF i PLEXOL są przeznaczona do rozwiązywania zagadnień z zakresu sztucznej inteligencji, np. pisania fragmentów translatorów dla Języków wyższych rzędów. Obiektem działania programów w Języku BIGRAF są dane w postaci grafów o dwóch gałęziach wychodzących z każdego wierzchołka. Język PLEX0L Jest uogólnieniem BIGRAF-u i dopuszcza dowolną liczbę gałęzi wychodzących z poszczególnych wierzchołków.

Obydwa Języki są iteracyjne i mają strukturę podobną do języka ALG0L-60, natomiast nie są językami rekursywnymi.

System Diagnostyczny MC ZAM-41

Jacek Sobaniec, Wolf Gendelman, Jan Klimowicz

Instytut Maszyn Matematycznych

(KB jednym z przykładów jak dobrze oprogramowana była maszyna ZAM-41 jest poniższy dokument opisujący niezależny od SO-141 system testów kontrolnych, stosowany, gdy testy wbudowane w SO-141 zawodziły lub gdy SO-141 nie dał się załadować)

WSTĘP

W eksploatacji systemów maszyn cyfrowych, szczególnie w rozbudowanych systemach, istotny jest z punktu widzenia ekonomicznego problem strat czasu maszyny, związany z czasem przeznaczonym na detekcję błędów, lokalizację uszkodzeń, konserwację i naprawy oraz badania profilaktyczne. Dla dużych zestawów maszyn cyfrowych współczynnik czasu nieużytecznego maszyny, uzależniony głównie przyczynami strat czasu wymienionymi wyżej, osiąga wartości od 0,1 dla efektywnych systemów maszyny cyfrowej do 0,3 dla systemów mniej sprawnych. Doniosłą rolę w kształtowaniu wartości współczynnika czasu nieużytecznego maszyny odgrywa obok niezawodności ‘hardware’ wyposażenie systemu maszyny cyfrowej w odpowiednio sprawne środki diagnostyczne i to zarówno w zakresie diagnostyki stanu, jak i lokalizacji uszkodzeń.

W większości systemów maszyn cyfrowych środki diagnostyczne zabezpieczone są przez specjalne wbudowane w hardware dla tego celu układy oraz poprzez mniej lub bardziej rozbudowane systemy programów diagnostycznych.

Maszyna cyfrowa ZAM-41, poza nielicznymi jej modułami, ogólnie rzecz biorąc nie ma wbudowanych układów kontrolnych. Fakt ten zaważył na rozwiązaniu systemu diagnostycznego maszyny ZAM-41 głównie poprzez realizację na drodze programowej.

Maszyna ZAM-41 wyposażona jest w zespół programów kontrolno-diagnostycznych, na który składają się dwa systemy testów:

- System Testów Kontrolnych K-400,
- System Testów Lokalizujących L-141.

W dalszym ciągu niniejszej pracy omówione będą te dwa systemy.

System Testów Kontrolnych K-400

Ogólne założenia Systemu Testów Kontrolnych K-400

System Testów Kontrolnych jest jednym ze środków programowej kontroli poprawności działania maszyny cyfrowej ZAM-41. Zadaniem tego Systemu jest detekcja i automatyczna przybliżona lokalizacja uszkodzeń na poziomie modułu maszyny, grup rozkazów lub poszczególnych rozkazów. System K-400 pozwala na wydatne skrócenie przeciętnego czasu usuwania uszkodzeń.

W hierarchii Systemu diagnostycznego MC ZAM-41 System Testów Kontrolnych K-400 zajmuje pośrednie miejsce między zadaniami kontrolnymi a Systemem Lokalizującym L141. System K-400 jest przystosowany do wykrywania uszkodzeń w Centralnej Jednostce Przetwarzania i pamięciach operacyjnych oraz w modułach urządzeń zewnętrznych.

System zaprojektowany został na wykrywanie uszkodzeń pojedynczych i wielokrotnych, trwałych i nietrwałych. Uszkodzenia nietrwałe wykrywane są, o ile zaistnieją, w trakcie wykonywania się odpowiedniej sekwencji programowej i w układach, które objęte są w tym czasie kontrolą przez tę sekwencję programową.

Określenie 1. Uszkodzenie trwałe jest to uszkodzenie nieprzemijające, powstałe z przyczyn trwałego uszkodzenia fragmentu sieci, bramki logicznej itp.

Określenie 2. Uszkodzenie nietrwałe jest to uszkodzenie przemijające, powstałe na skutek zakłóceń zasilania, przypadkowych i starzeniowych zmian parametrów elementów sieci logicznych.

Podstawowym założeniem Systemu jest wykrywanie wyłącznie uszkodzeń zdeterminowanych programowo, tzn. takich, które umożliwiają realizację określonej kontrolnej sekwencji rozkazowej. W każdym innym przypadku mogą powstać tzw. stany niezidentyfikowane.

Metoda Programowej Kontroli MC

Wybór metody

Istnieją dwie podstawowe metody kontroli programowej maszyny. Pierwsza metoda polega na sprawdzeniu stanu maszyny za pomocą tzw. zadań kontrolnych, druga za pomocą tzw. testów kontrolnych.

Sprawdzenie maszyny za pomocą zadania kontrolnego sprowadza się do wykonania tego zadania wg założonych danych i porównania uzyskanych wyników ze z góry znanymi wynikami. Wyniki diagnozy za pomocą zadań kontrolnych są zazwyczaj zbyt ogólne ze względu na brak lub ograniczone informacje o lokalizacji błędu oraz ze względu na zawężony zakres sprawdzania maszyny.

Innymi słowy, zadanie kontrolne przeprowadza podział na maszynę sprawną i niesprawną dla danego problemu o konkretnych argumentach bez żadnej pewności co do prawidłowego wykonania innego zadania lub nawet tego samego o zmienionych argumentach.

Z drugiej strony istnieją obiektywne trudności w określaniu doboru zadań kontrolnych i ich argumentów, koniecznych do skontrolowania całej maszyny.

Kontrola programowa za pomocą testów, aczkolwiek jest metodą też nie w pełni doskonałą i nieuwzględniającą wszystkich możliwych stanów w maszynie, ma następujące cechy dodatnie:

1) umożliwia zgrubną lokalizacją uszkodzenia;
2) realizacja programowa metody wydaje się mniej skomplikowana;
3) zbiór tekstów może objąć kontrolą całą maszyną.

Metoda testowa sprawdzania maszyny polega na próbie wykonania operacji badanej i analizie skutków wykonania operacji w odniesieniu do założeń na daną operacją.

W oparciu o powyższą analizą i biorąc pod uwagą specyfiką MC ZAM-41, zdecydowano się na wybór testowej metody kontroli stanu maszyny z uwzględnieniem:

1) wyodrębnienia w maszynie mniej lub więcej niezależnych modułów,
2) kontroli poszczególnych modułów za pomocą niezależnych od siebie testów jak najmniej angażujących moduły nieobjęte sprawdzaniem przez dany test,
3) zautomatyzowanego sterowania działaniem testów i łatwością komunikacji pomiędzy operatorem i Systemem Testów,
4) stosowania rozszerzalności kontroli, tzn. obejmowania kontrolą stopniowo większej ilości układów w maszynie z wykorzystaniem do tych celów układów (operacji) uprzednio sprawdzonych,
5) wygodnej dla operatora sygnalizacji o wykrytych błędach z podaniem maksymalnych możliwych i dostępnych dla testu informacji o warunkach, w jakich wykryto błąd,
6) zautomatyzowanej ewidencji pracy testu.

Do sprawdzania poprawności działania poszczególnych układów maszyny wykorzystywane są w testach argumenty bądź w postaci stałych - tzw. szablonów, bądź w postaci zmiennych generowanych przez generator liczb pseudolosowych.

Struktura Testów Kontrolnych K-400

Działanie Systemu Testów Kontrolnych i prowadzenie przez niego kontroli programowej możliwe jest przy założeniu sprawności maszyny w pewnym minimalnym zakresie, obejmującym pewien obszar pamięci operacyjnej, niektóre rejestry, operacje wykonywane ze stolika operatorami kilka podstawowych rozkazów Jednostki Centralnej (PZA, STO, CTA).

Całość programów Systemu K-400 składa się z dwóch podstawowych części, z których każda jest zbiorem testów.

Pierwszy zbiór testów nazwano "Maszyną Minimalną". Drugi zbiór testów jest właściwym Systemem Kontrolnym o umownym symbolu K-400.

"Maszyna Minimalna" jest to zbiór krótkich testów. Poszczególne testy "Maszyny Minimalnej" sprawdzają w zasadzie pojedyncze operacje maszyny. Najpierw sprawdzana jest operacja SOB, umożliwiająca zbudowanie programowej pętli do automatycznego wczytywania systemu z nośnika na taśmie papierowej, następnie kontrolowany jest obszar pamięci operacyjnej przeznaczony na zmagazynowanie Systemu K-400, potem zaś kontrolowane jest prawidłowe wykonywanie sekwencji operacji współpracy Jednostki Centralnej z pamięcią bębnową albo pamięcią taśmową. Każdy test zbioru testów K-400 sprawdza działanie określonego modułu MC ZAM-41 Z. K-400 zawiera następujące testy:

- test sprawdzający rozkazy Sterujące,
- test sprawdzający Arytmometr,
- test sprawdzający Operacje Zmiennoprzecinkowe,
- test sprawdzający Pamięć Operacyjną,
- test sprawdzający Pamięć Bębnową,
- test sprawdzający Taśmy Magnetyczne,
- test sprawdzający Monitor dalekopisowy,
- test sprawdzający Perforatory i czytniki taśmy papierowej,
- test sprawdzający Czytnik kart,
- test sprawdzający Drukarkę wierszową.

Pracą Systemu Kontrolnego K-400 steruje program „Dyrygent”, który funkcjonalnie dzieli się na dwie części składowe. Pierwsza część zwana "M-Dyrygent" jest programem organizującym przejście od "Maszyny Minimalnej" do K-400. Program "M-Dyrygent" zapamiętywany jest w pamięci operacyjnej jako ostatni test "Maszyny Minimalnej".

Korzystając z operacji sprawdzonych już przez testy "Maszyna Minimalna", program "M-Dyrygent" wczytuje z nośnika na taśmie papierowej cały zbiór testów K-400 do jednej z pamięci pomocniczych, przygotowuje i produkuje krótki odcinek taśmy papierowej z programem manipulacyjnym Nr. 1.

Program manipulacyjny Nr 1 zawiera pętlę programową służącą do przepisywania Systemu Kontrolnego K-400 z pamięci pomocniczej do pamięci operacyjnej oraz danych do samokontroli systemu (sumy kontrolne).

Po jednorazowym wczytaniu Systemu Kontrolnego K-400 z nośnika na taśmie papierowej do pamięci pomocniczej, każdorazowo system kontrolny może być uruchomiony za pomocą programu manipulacyjnego Nr 1 z pominięciem testów "Maszyny Minimalnej".

Wymieniony wyżej sposób "rozruchu" Systemu Kontrolnego jest znacznie szybszy, wymaga jednak założenia z góry poprawności wykonywania operacji np. CKA, SOB, itp.

Druga część "Dyrygenta" zwana "D-Dyrygent" jest właściwym programem sterującym wykonywaniem testów K-400. Podczas pracy Systemu Kontrolnego K-400 "Dyrygent" znajduje się w Pamięci Operacyjnej. Zadania "D-Dyrygenta" są następujące:

1. Wprowadzanie do pamięci operacyjnej z pamięci pomocniczej kolejnych testów K-400 i inicjowanie ich działania wg parametrów ustalonych przez operatora na początku działania Systemu K-400.
2. Prowadzenie ewidencji o pracy Systemu Kontrolnego i poszczególnych testów K-400 z podawaniem tych danych operatorowi poprzez wypis tabulogramu na monitorze dalekopisowym.
3. Umożliwianie operatorowi ingerencji w trakcie pracy systemu.

Charakterystyka konstrukcji Systemu Testów kontrolnych K-400

"Maszyna Minimalna"

Pogramy (testy) Systemu Kontrolnego K-400 znajdują się w odpowiedniej kolejności na nośniku w postaci taśmy papierowej. Informacją z tego nośnika wczytywane są do maszyny za pomocą czytnika ze stolika operatora.

Jak wspomniano wyżej System Kontrolny K-400 zapamiętywany jest w pamięci pomocniczej. Zapamiętywanie w pamięci pomocniczej właściwego zbioru testów kontrolnych wchodzących w skład K-400, poprzedza wykonywanie testów "Maszyny Minimalnej". Poszczególne testy M-M, znajdujące się na początku taśmy papierowej, są krótkimi odcinkami programów wczytywanymi etapami do maszyny. Każdy etap wczytywania obejmuje zapamiętanie w pamięci operacyjnej pojedynczego testu "Maszyny Minimalnej" i zainicjowanie wykonywania tego programu. Jeśli wynik testu jest dodatni tzn. gdy nie wykryto błędu, wtedy w sposób automatyczny wczytywany jest do maszyny kolejny test "Maszyny Minimalnej".

Jeżeli zostanie wykryty błąd w czasie wykonywania określonego testu "Maszyny Minimalnej", praca maszyny jest wstrzymana rozkazem STO, którego część adresowa służy do odszukania dodatkowych informacji o błędzie w Instrukcji Eksploatacyjnej Systemu K-400. Pierwsza lewa cyfra dziesiętna tego adresu stopu, określa numer testu w zbiorze testów "Maszyny Minimalnej". Niektóre informacje o błędzie (wzorce i uzyskane wyniki operacji) podawane są również w rejestrach maszyny dostępnych operatorowi.

Wspomniana już wyżej możliwość wydzielenia z K-400 właściwego systemu kontrolnego (D-Dyrygent i Testy), dokonana została ze względów eksploatacyjnych. Dzięki temu możliwa jest znacznie szybsza, kontrola całości maszyny za pomocą systemu ulokowanego w pamięciach pomocniczych o znacznie krótszym czasie dostępu w porównaniu do koniecznego czasu wczytania systemu z taśmy papierowej. Każdy test systemu K-400 jest przeznaczony do sprawdzenia określonego modułu maszyny. Metody programowej kontroli poszczególnych modułów maszyny podane są w poniższym opisie testów.

Test Arytmometru oraz Test Zmiennego Przecinka

Zadaniem testu jest sprawdzenie poprawności działania wszystkich operacji dotyczących Arytmometru oraz sygnalizacja trwałych i przypadkowych błędów urządzenia. Poszczególne operacje są sprawdzane za pomocą argumentów w postaci liczb pseudolosowych oraz na niektórych stałych argumentach tzw, szablonach zero-jedynkowych.

Przy sprawdzaniu poprawności wykonywania operacji arytmetycznych, na specjalnie dobranych argumentach-szablonach. wynik operacji porównuje się ze znanym szablonem wzorcowym. Kontrola wykonywania operacji zmienno i stałoprzecinkowych oparta jest na metodzie porównania dwóch wyników uzyskanych z zastosowania prawa przemienności do sumy, różnicy i iloczynu argumentów wygenerowanych w postaci liczb pseudolosowych. W czasie kontroli wykonywania tych operacji korzysta się również ze stałych argumentów wybranych jako przypadki szczególne.

Sprawdzanie poprawności wykonywania operacji dzielenia odbywa się przez wykorzystanie wcześniej sprawdzonych operacji dodawania, odejmowania i mnożenia. Operacje logiczne i inne sprawdzane są w oparciu o specjalnie dobrane szablony i uprzednio skontrolowane operacje arytmetyczne.

Test Pamięci Operacyjnej PA0-5

Zadaniem testu jest sprawdzenie pamięci operacyjnej poprzez kontrolę jej podstawowych funkcji zapisu, pamiętania i odczytu informacji. Metoda kontroli oparta jest na zapisie specjalnych liczb pseudolosowych, następnie na ich odczycie i porównaniu tych informacji z informacjami wzorcowymi.

Pamięć operacyjna sprawdzana jest w blokach po 2048 słów. Test ten umożliwia:

- dokonanie zapisu i odczytu "0" i "1" pod każdym adresem i w każdym bicie pamięci operacyjnej, ze sprawdzeniem informacji będącej adresem, a następnie negacją adresu wybranej komórki,
- zapis i wielokrotny odczyt z każdej komórki pamięci w warunkach szczególnie niekorzystnych dla odczytu, poprzez wywoływanie zakłóceń wybieraniem informacji z sąsiednich adresów,
- wykrycie nietrwałych uszkodzeń i sprzężeń w układach pamięci,
- sprawdzanie zachowania się kolejnych linii słów pamięci poprzez zapisanie w nich liczb pseudolosowych w warunkach maksymalnie niekorzystnych dla pamięci operacyjnej. Lokalizacja testu w pamięci w czasie jego działania jest automatycznie zmienna,dla umożliwienia kontroli całego obszaru pamięci.

Test Pamięci Bębnowej

Zadaniem testu jest sprawdzenie podstawowych funkcji zapisu, odczytu i przechowywania informacji w Pamięci Bębnowej. Metoda kontroli polega na zapisie, a następnie odczycie i Sprawdzeniu specjalnych szablonów i liczb pseudolosowych.

Test umożliwia:

- Zapis wyznaczonego zakresu bębna liczbami pseudolosowymi, a następnie odczyt i sprawdzenie ze stałym blokiem wzorcowym.
- Zapis wyznaczonego zakresu bębna własnymi adresami, odczyt i sprawdzenie z generowanym wzorcem.
- Zapis wyznaczonego zakresu bębna słowem, w którym występuje i cyklicznie na przemian 12 jedynek i zer, następnie odczyt i sprawdzenie ze wzorcem. Sekwencja operacji zapis-odczyt-sprawdzenie powtarza się 12-krotnie, z tym że po każdej kolejnej sekwencji następuje przesunięcie cykliczne o 1 zapisywanego szablonu. Próba ma na celu sprawdzenie przemagnesowania nośnika.

Test Taśmy Magnetycznej

Zadaniem testu jest sprawdzenie stacji taśm magnetycznych pod kątem prawidłowości zapisu, odczytu bloków informacji oraz współpracy stacji taśm z jednostką centralną (kanałami przesyłania).

Metoda kontroli oparta jest na zapisie, a następnie odczycie i sprawdzeniu bloków informacji w postaci liczb pseudolosowych. Długość bloków oraz przyrosty bloków informacyjnych deklarowane są przez operatora na kluczach rejestru kluczy MC. Test umożliwia jednocześnie sprawdzenie różnej ilości stacji pamięci taśmowych, jednakże nie większej niż 8.

Testy urządzeń zewnętrznych

Do tych testów zaliczamy:

- **Test Przerwań**, zadaniem którego Jest sprawdzenie poprawności działania układu przerwań, układu priorytetu i wynikającej z tego możliwości pracy wieloprogramowej maszyny. Metoda sprawdzania polega na jednoczesnym uruchomieniu zadeklarowanych przez operatora urządzeń we-wy i analizie kolejności zgłaszanych i przyjmowanych przez jednostką centralną przerwań. Praca testu dokumentowana jest na dalekopisie.

- T**est Perforator i Czytnik taśmy papierowej**. Zadaniem testu jest kontrola poprawności działania modułów maszyny z czytnikami i perforatorami taśmy papierowej. Zastosowano metodą polegającą na symulowaniu programu wypisującego lub czytającego pseudolosowe znaki w zmiennych odstępach czasu z jednoczesnym sprawdzaniem poprawności wczytywanych znaków. Rozkład pseudolosowych odstępów czasu między kolejnymi operacjami pisania lub czytania jest taki,aby znaczna część tych operacji mogła się wykonywać z maksymalną dopuszczalną przez urządzenie szybkością.

- **Test Monitor dalekopisowy**. Zadaniem testu jest sprawdzenie poprawności działania modułu Monitora dalekopisowego przy współpracy z maszyną. Przyjęto metodę polegającą na wypisywaniu na tabulogramie Monitora wszystkich znaków dalekopisowych (litery i cyfry). Wypisywane są bądź wzorcowe testy, bądź dowolne testy uprzednio wczytane wg dyspozycji operatora. Drogą ingerencji operatora istnieje możliwość badania poprawności działania czytnika i perforatora dalekopisowego.

- **Test Czytnik Kart**. Test ten sprawdza prawidłowość czytania informacji zapisanej na kartach perforowanych. Przyjęto metodę, że informacja na kartach 80-kolumnowych jest zbiorem liczb pseudolosowych, generowanych wg zależności:

gdzie: Xi - informacja zapisana w i-tej kolumnie.
Obok dróg informacyjnych test sprawdza funkcjonowanie dróg sterujących.

- **Test Drukarka Wierszowa**. Zadaniem testu jest sprawdzenie działania drukarki podczas jej współpracy z maszyną. Test umożliwia sprawdzenie pracy wszystkich czcionek drukarki, układów sterujących młoteczkami i buforów informacyjnych. Przyjęto metodę kontroli optycznej prawidłowości druku wszystkich znaków drukarki dla następujących kombinacji:

- wydruk wszystkich znaków w wierszu, z tym że każdy następny wiersz jest cyklicznie przesunięty o jeden znak w stosunku do poprzedniego,
- wydruk 5 wędrujących spacji na tle wybranych znaków (24 wiersze),
- wydruk wędrującego alfabetu na tle wybranych znaków (120 wierszy).

Ponadto sprawdza się specyficzne sytuacje w układach sterujących.

- T**est Wskaźniki i Test Rozkazy Sterujące** ma za zadanie sprawdzenie niektórych wskaźników i rejestrów nieobjętych sprawdzaniem w wyżej wymienionych testach. Jest on jak gdyby testem uzupełniającym. Test kontroluje pracą zegara, wskaźniki kanałów oraz wskaźniki pamięci bębnowej. Zadaniem testu badającego rozkazy sterujące jest sprawdzenie Sieci Centralnego Sterowania w zakresie poprawności wykonywania operacji:

- modyfikacji,
- rozkazów skokowych warunkowych i bezwarunkowych,
- przeskoków,
- rozkazów programowych
- nielegalnych.

Metoda kontroli polega na próbie wykonania rozkazu i analizie wyników tej próby w odniesieniu do zakładanych efektów. Ponieważ każdy inny test Systemu K-400 korzysta z wyżej wymienionych operacji, D--Dyrygent systemu uruchamia przede wszystkim ten test dla określenia stanu Sieci Centralnego Sterowania. Sprawność Sieci Centralnego Sterowania w założeniach Systemu K-400 jest niezbędna do badania pozostałych modułów maszyny cyfrowej.

Charakterystyka własności kontrolno-lokalizujących Systemu Testów Kontrolnych K-400

Jak już było powiedziane we wstępie, System Kontrolny K-400 wykrywa i zgrubnie lokalizuje szkodzenia, dające tylko przejawy programowe. Dokładność lokalizacji ogranicza się bądź do modułu,bądź do rozkazu, w zależności od typu uszkodzenia i możliwości rozgraniczenia funkcji sieci logicznych na podstawie objawów programowych. Uszkodzenia w Sieci Centralnego Sterowania najczęściej objawiają się brakiem końca operacji. Wówczas z reguły niemożliwe jest bliższe określenie przyczyny uszkodzenia.

W innych przypadkach sygnalizacja uszkodzeń przez różne testy pozwala bliżej określić ich miejsce.
Jednakże wnioski lokalizujące na podstawie wyników tylko jednego testu mogą być niejednoznaczne. Na przykład sygnalizacja błędu w czasie działania testu arytmometru może być spowodowana uszkodzeniem pamięci operacyjnej. Dopiero działanie dwóch testów -arytmometru i pamięci operacyjnej - dostarczy właściwych danych identyfikujących o hardware'owe źródło błędu.

Bliższa lokalizacja uszkodzenia jest możliwa po zastosowaniu Systemu Lokalizującego L-141. Pełna lokalizacja wymaga angażowania środków technicznych i udziału obsługi konserwatorskiej.

System Testów Kontrolnych K-400 przewidziany jest do określania stanu technicznego maszyny cyfrowej ZAM-41 Z przy jej codziennej i okresowej kontroli profilaktycznej.

Niezależnie od wyżej wymienionego przeznaczenia System Testów Kontrolnych K-400 może służyć jako narzędzie do badań maszyny ZAM--41 w różnych warunkach technicznej i użytkowej eksploatacji oraz na pewnym szczeblu procesu jej produkcji i uruchamiania.

System Testów Lokalizujących L-141

Ogólne założenia Systemu Testów Lokalizujących L-141

System Testów Lokalizujących jest jednym ze środków służących zwiększeniu eksploatacyjnej niezawodności maszyny cyfrowej ZAM-41. Niezawodność eksploatacyjną maszyny cyfrowej można poprawić drogą stosowania w konstrukcji elementów o większej niezawodności lub poprzez skrócenie czasu naprawy w wypadku powstania jakiegoś uszkodzenia.

W czasie niezbędnym dla dokonania naprawy większą część stanowi czas potrzebny na zlokalizowanie uszkodzonych elementów. Zadaniem systemu L-141 jest wykrywanie i automatyczna lokalizacja uszkodzeń. Jednocześnie zautomatyzowanie procesu lokalizacji uszkodzeń pozwala zmniejszyć wymagania co do kwalifikacji obsługi technicznej maszyny ZAM-41.

System L-141 wykrywa i lokalizuje uszkodzenia w części maszyny cyfrowej ZAM-41 obejmującej jednostką centralną i moduły pamięci operacyjnej. Kontrola działania urządzeń zewnętrznych realizowana jest przez System Kontrolny K—400. System L—141 zbudowany został w zasadzie w celu lokalizacji jedynie uszkodzeń pojedynczych.

Uszkodzenia wielokrotne mogą być lokalizowane przez system, jeżeli są niezależne, tzn. takie, które nie kompensują się i nie dają innych objawów niż każde z nich oddzielnie. Uszkodzenia lokalizowane przez system L-141 muszą być trwałe, tj. nieprzemijające w czasie pracy systemu lokalizującego (ok. 15 minut). Jeżeli uszkodzenie nie jest trwałe, może, ale nie musi, pozostać nie wykryte, bądź też system dostarczyć może fałszywych wyników.

Ograniczenia odnośnie do niezależności uszkodzeń wielokrotnych i trwałości uszkodzeń mogły być w znacznym stopniu zredukowane w przypadku lokalizacji uszkodzeń w modułach pamięci operacyjnej. Osiągnięte to zostało przez zastosowanie statystycznej analizy objawów błędnej pracy pamięci operacyjnej PAO-5.

Metoda lokalizacji uszkodzeń w MC

Na wybór metody wpływ miały następujące okoliczności, w których system L-141 był realizowany.

1) W Części Centralnej oraz pamięci operacyjnej maszyny cyfrowej ZAM-41 nie zostały przewidziane żadne układy kontroli poprawności działania. Stąd jedynymi objawami niesprawności, pozwalającymi na automatyczną lokalizację uszkodzeń, są objawy programowe polegające na błędnym wykonywaniu się rozkazów.

2) Brak opisu formalnego maszyny ZAM-41 wynikający z niestosowania przy jej projektowaniu rozbudowanego systemu automatycznego projektowania oraz wynikający stąd brak odpowiedniego symulatora maszyny ZAM-41. W tej

sytuacji można było zastosować jedynie metody, w których lokalizacji uszkodzeń dokonuje się w oparciu o objawy programowe. Stworzenie systemu diagnostycznego opartego o metodę tabeli diagnostycznej uznane zostało za przedsięwzięcie nierealne. Wypełnienie takiej tabeli ze względu na jej wielkość wynikającą z dużej liczby uszkodzeń, jakie należałoby uwzględnić, wydaje się możliwe tylko przy automatyzacji tej pracy w oparciu o odpowiedni symulator maszyny.

Z powyższych przyczyn wybrano jako najodpowiedniejszą metodę opartą na zasadzie rozszerzania kontroli od pewnego minimalnego poziomu sprawności maszyny. Zakłada się więc sprawność maszyny w pewnym minimalnym zakresie obejmującym kilkadziesiąt miejsc pamięci operacyjnej, kilka rozkazów, niektóre rejestry i operacje wykonywane ze stolika operatora. W tym zakresie maszyna musi być sprawdzona przed uruchomieniem systemu. Następnie, po uruchomieniu, System L-141 sprawdza pozostałe operacje, sygnalizując stwierdzenie błędnego działania maszyny zatrzymaniem pracy i podaniem w rejestrach MC informacji o niesprawności.

Generalna metoda stosowana w Systemie L-141 dla sprawdzenia kolejnych rozkazów polega na wykonaniu ich przy danych odpowiednio dobranych argumentach i porównaniu wyniku z wyznaczonym wynikiem poprawnym. Jeśli wynik nie jest prawidłowy, następuje porównanie z przewidywanymi dla konkretnych uszkodzeń wynikami błędnymi. Może być ono wykonywane programowo lub też częściowo przez obsługę na podstawie wskazówek zawartych w instrukcji eksploatacyjnej.

Argumenty operacji i przewidywane wyniki określa się tak, aby uwzględnione zostały możliwie wszystkie uszkodzenia wszystkich bramek sieci logicznej.

Odmienna jest metoda lokalizacji uszkodzeń w modułach pamięci operacyjnej PA0-5. Polega ona na zebraniu informacji o poprawności wykonywania się operacji zapisu-odczytu we wszystkich komórkach bitowych i przeprowadzeniu statystycznej analizy tych wyników w celu określenia uszkodzeń w układach adresowych i informacyjnych. Zasadą stosowaną przy tym jest, że uszkodzony układ powoduje niesprawny zapis - odczyt w większości tych wszystkich komórek bitowyoh, przy wybieraniu których bierze czynny udział.

Struktura Systemu Testów Lokalizujących L-141

System zakłada sprawność pewnego minimalnego zbioru operacji zwanego "Maszyną Minimalną". Są to operacje takie jak "Ładuj", "Wprowadź", PZA, STO, niezbędne dla uruchomienia systemu i sygnalizacji przez ten system faktów wykrycia uszkodzeń. Przewidziana jest jednak wstępna kontrola "Maszyny Minimalnej" przez operatora. W tym celu operator powinien przed uruchomieniem systemu wykonać następujące wstępne czynności:

1) Sprawdzenie przesyłania do rejestrów przez kolejne załadowanie "zer" i "jedynek" do rejestrów R, L, A, M, B
2) Wykonanie rozkazu PZA O , gdy w A znajdują się same "jedynki".
3) Wykonanie UMA O.
4) Wykonanie punktów 2) i 3), gdy zawartość A - same "zera".
5) Przesyłanie do miejsca pamięci operacyjnej o adresie 1 i 2 rozkazu UMA 0.

6) Wykonanie 1 cyklu.

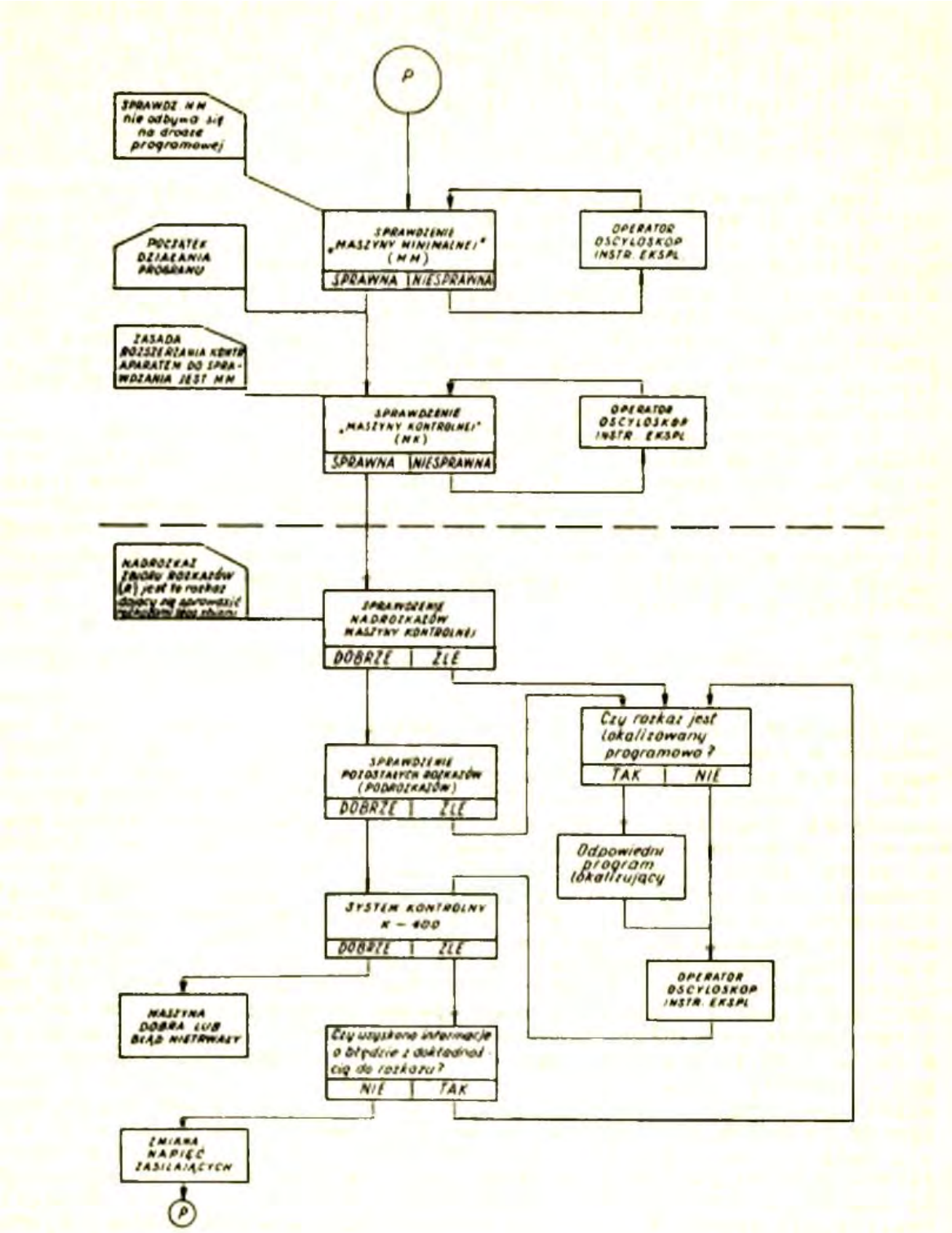
W czasie tego sprawdzania zachodzi nieunikniona dwuznaczność: obszar pamięci operacyjnej "Maszyny Minimalnej" i rozkaz PZA można sprawdzić tylko za pomocą rozkazu UMA, który może być niesprawny. Czynności powyższe poleca się wykonywać tylko w przypadku zaistnienia błędu maszynowego, którego wykrycie daje informacje niedostatecznie sprecyzowane lub dwuznaczne. W innych przypadkach zakłada się, że "Maszyna Minimalna" jest sprawna i przy jej pomocy sprawdza się rozkazy ze zbioru rozkazów tzw. "Maszyny Kontrolnej".

Stosowana jest przy tym zasada rozszerzania kontroli. Polega ona na tym, że każdy kolejny rozkaz sprawdzany jest za pomocą rozkazów sprawdzonych w poprzednich krokach. W ten sposób do sprawdzania można używać coraz większej liczby rozkazów wykonujących się poprawnie. Przede wszystkim sprawdzane są rozkazy CTA 58.NAS 6, a następnie UMB, SOB i B-modyfikacja, pozwalająca zautomatyzować czynność wprowadzania informacji z taśmy papierowej i utworzyć pętlę programową. Dalej sprawdzane są rozkazy UMA, PMB, OSL, LCD, SZA, SKO pozwalające na utworzenie zwrotnicy w programie i analizę rezultatów operacji sprawdzanej. W wypadku wykrycia niesprawności na etapie sprawdzania rozkazów "Maszyny Kontrolnej" działanie systemu zostaje zatrzymane aż do usunięcia wykrytego uszkodzenia.

Prawidłowo wykonujące się w maszynie rozkazy "Maszyny Kontrolnej" służą do sprawdzania swoich "nadrozkazów" (rozkazów dających się sprawdzić za pomocą rozkazów "Maszyny Kontrolnej"), wchodzących w skład zbioru rozkazów tzw. "Maszyny Sprawdzanej". Na tym etapie zasada rozszerzania kontroli nie jest już stosowana. Wykrycie uszkodzenia nie zatrzymuje więc działania systemu aż do jego usunięcia. Wykrycie niesprawności sygnalizowane jest rozkazem STO powodującym wstrzymanie pracy systemu (co najmniej na czas potrzebny operatorowi dla odczytania zawartości rejestrów, których stan dostarcza informacji o wykrytym uszkodzeniu).

Po sprawdzeniu nadrozkazów "Maszyny Kontrolnej" system przechodzi do sprawdzenia jej "podrozkazów", tj. tych pozostałych rozkazów "Maszyny Sprawdzanej", które nie mogły być sprawdzone przez "Maszynę Kontrolną", ale które można sprawdzić dotychczas sprawdzonym aparatem. Między innymi jako podrozkaz "Maszyny Kontrolnej" sprawdzany może być całkowity obszar bloku pamięci operacyjnej.

Na rys. 2 przedstawiono schemat blokowo-ideowy Systemu Testów Lokalizujących L-141.



Rys. 2. Schemat blokowo-ideowy Systemu Testów Lokalizujących L-141

Charakterystyka konstrukcji Systemu Testów Lokalizujących L-141

Programy lokalizujące znajdują się we właściwej kolejności na nośniku w postaci taśmy papierowej, z której informacje są wczytywane przez czytnik ze stolika operatora. W czasie pracy systemu taśma automatycznie przesuwa się pod czytnikiem.

W wypadku stwierdzenia błędnego wykonywania się aktualnie sprawdzanej operacji praca systemu zostaje zatrzymana rozkazem STO, którego numer (część adresowa) służy do odszukania informacji o miejscu uszkodzenia. Informacje te są zebrane w Instrukcji Eksploatacyjnej Systemu L-141. Dodatkowe informacje niezbędne dla dokonania szczegółowej lokalizacji uszkodzenia napisane są bezpośrednio na taśmie papierowej w miejscu, gdzie zatrzymuje się ona pod czytnikiem w wypadku stwierdzenia niesprawności.

Niektóre informacje podawane są również w rejestrach maszyny i mogą być odczytywane przez operatora w rejestrze lampek na stoliku operatora lub na symulatorze. Na taśmie i w rejestrach podawane są przede wszystkim informacje o sprawdzanej operacji, jej argumentach oraz wynikach przewidywanych i rzeczywiście otrzymanych. W niektórych przypadkach podane są w ten sposób bezpośrednie informacje lokalizujące.

Cały System L-141 został podzielony na dwie części, z których pierwsza służy do lokalizacji uszkodzeń w Części Centralnej maszyny ZAM-41, a druga ma lokalizować uszkodzenia w modułach Pamięci Operacyjnej PAO-5.

Wydzielenie testu lokalizującego pamięci operacyjnej z całości systemu L-141 zostało dokonane ze względów użytkowych. Dzięki temu możliwa jest lokalizacja uszkodzeń w pamięci operacyjnej również wtedy, gdy ze względu na jej niesprawność nie można uruchomić w całości systemu L-141. Możliwe jest również pominięcie tej części Systemu, która dotyczy lokalizacji w części maszyny ZAM-41, co do której nie ma podejrzeń o niesprawności.

Charakterystyka własności lokalizujących Systemu L-141

Jak już było powiedziane w rozdziale poprzednio, System L-141 sprawdza i lokalizuje tylko uszkodzenia dostatecznie długo nie przemijające, w zasadzie pojedyncze w Części Centralnej i modułach pamięci operacyjnej maszyny ZAM-41. Wykrywane i lokalizowane są tylko uszkodzenia dające przejawy programowe i to tylko takie, które są wynikiem niesprawnego działania określonej bramki. Ogólnie można więc określić wykrywane przez system uszkodzenia jako stabilne niesprawności bramek w sieci logicznej.

Nieco większej skuteczności wykrywania uszkodzeń można oczekiwać od testu lokalizującego w modułach pamięci operacyjnej, a to dzięki zastosowaniu metody analizy statystycznej. Dzięki temu mogą być wykrywane również uszkodzenia nietrwałe, niestabilne i nie polegające na krańcowej niesprawności elementu. Wymagane jest jedynie, aby element dawał objawy niesprawności w dostatecznie dużej części przypadków, w których jest angażowany w pracy układu.

Ponieważ System L-141 sprawdza kolejno (z dokładnością do mikrooperacji) rozkazy maszyny ZAM-41 i ponieważ lokalizacja opiera się o analizę programowych jedynie objawów uszkodzenia,przeto dokładność lokalizacji zależna jest wyraźnie od ilości sprzętu i charakteru sieci angażowanej przez sprawdzany rozkaz. Najczęściej lokalizacja może być przeprowadzona z dokładnością do pakietu lub grupy pakietów.

W wielu jednak przypadkach dotyczących przede wszystkim uszkodzeń sieci centralnego sterowania jedynym objawem jest brak sygnału końca operacji. W takim przypadku bliższa lokalizacja jest z reguły niemożliwa 1 wymaga bezpośredniego zaangażowania obsługi lokalizującej uszkodzenia z pomocą środków technicznych (oscylograf, woltomierz, symulator).

System Testów Lokalizujących L-141 przewidziany jest jako środek ułatwiający diagnozę uszkodzeń w przypadku podejrzeń o niesprawną pracę maszyny ZAM-41. Po zlokalizowaniu i usunięciu wszystkich zlokalizowanych uszkodzeń maszyna powinna być sprawdzona Systemem Testów Kontrolnych K—400. Jeżeli w wyniku działania tego Systemu można przyjąć, że wszystkie uszkodzenia zostały usunięte, tzn. że maszyna jest sprawna, wówczas rolę Systemu L-141 można uważać za zakończoną. Jeżeli natomiast System K-400 wykaże niesprawność, wówczas należy powtórzyć proces lokalizacji od początku.

Jeżeli System K-400 wykazuje niesprawność, a System L-141 uszkodzenia nie znajduje, można powtórzyć lokalizację z ewentualną zmianą napięć zasilających.

Niezależnie od przeznaczenia Systemu L-141 jako środka zwiększenia niezawodności eksploatacyjnej może on służyć jako narzędzie kontrolne przy uruchomianiu maszyny w procesie jej produkcji.

Uwagi o wynikach badań skuteczności systemu diagnostycznego MC ZAM-41

Przeprowadzono badania skuteczności Systemu Diagnostycznego MC ZAM-41. Metoda badań Systemu kontrolnego K-400 polegała na tym, że w poszczególnych modułach maszyny, sprawdzanych przez testy K-400, zasymulowano pewną liczbę uszkodzeń trwałych i parametrycznych (proporcjonalną do liczby pakietów), a następnie uruchomiono odpowiedni test Systemu K—400. Działanie testu oceniane było przez zespół prowadzący badania wg 4-punktowej skali ocen.

W wyniku badań określono skuteczność kontroli modułów przez poszczególne testy. Ogółem w maszynie zasymulowano 372 uszkodzenia. Ogólna skuteczność kontroli Systemu Testów Kontrolnych K-400 wynosi:

90,1 +/- 3%.

Przykładowo dla systemu kontrolno-diagnostycznego maszyny ESS Nr 1 skuteczność diagnozy stanu maszyny wynosi:

91,6 +/- 3,5%.

Dla oceny Systemu Testów Lokalizujących L-141 w poszczególnych modułach symulowano wyłącznie pewną liczbą wylosowanych uszkodzeń trwałych, a następnie lokalizowano je za pomocą Systemu L-141. Działanie Systemu oceniane było przez zespół prowadzący badanie wg 6-punktowej skali ocen.

Skuteczność lokalizacji dla testów lokalizujących w jednostce centralnej maszyny ZAM-41 i Pamięci Operacyjnej wynosi odpowiednio:

JC = 49,96 +/- 10%
PAO = 60 +/- 10%

Dokładniejsze omówienie metody badań skuteczności Systemu Diagnostycznego MC ZAM-41 i uzyskanych wyników przewidziane jest jako temat następnej publikacji.

W Instytucie Maszyn Matematycznych w Warszawie w pracach nad Systemem Diagnostycznym MC ZAM-41 obok autorów niniejszego opracowania udział wzięli:
mgr inż. Bożena Przyborowska, mgr inż. Ewa Zawadzka, mgr inż. Stanisław Choromański, mgr inż. Zdzisław Michalski i mgr inż. Waldemar Romaniuk.

System pracy w czasie rzeczywistym - TRAN

(KB)

Niestety, nie udało mi się znaleźć żadnej informacji na ten temat. O ile mi wiadomo, opracowany w Instytucie Technicznym Wojsk Lotniczych, był to system śledzenia przestrzeni powietrznej PRL (czy tylko?), opracowany na maszynę ZAM-41, czy też może jeszcze na jeden z prototypów ZAM-21, nie jestem pewien. Wszelkie próby dojścia do jakiejkolwiek informacji nie powiodły się.

Eksperymentalny system transmisji danych z maszyną cyfrową ZAM-41

Stanisław Chrobot, Andrzej Janik, Maciej Żebrowski

Instytut Maszyn Matematycznych

Pierwsze systemy przetwarzania informacji, wykorzystujące sieci telegraficzną i telefoniczną do transmisji danych alfanumerycznych na odległość, powstały na początku lat pięćdziesiątych. Dzisiejsze maszyny cyfrowe trzeciej generacji są już wszystkie wyposażone w tzw. kanały transmisji danych.

Jeżeli urządzeniem końcowym będziemy nazywali czytnik i dziurkarkę taśmy papierowej, czytnik kart dziurkowanych, maszynę do pisania (dalekopis), monitor ekranowy, jednostki pamięci na taśmie magnetycznej, to transmisja danych może odbywać się pomiędzy:

a) dwoma urządzeniami końcowymi,
b) urządzeniem końcowym a maszyną cyfrową,
c) dwiema maszynami cyfrowymi.

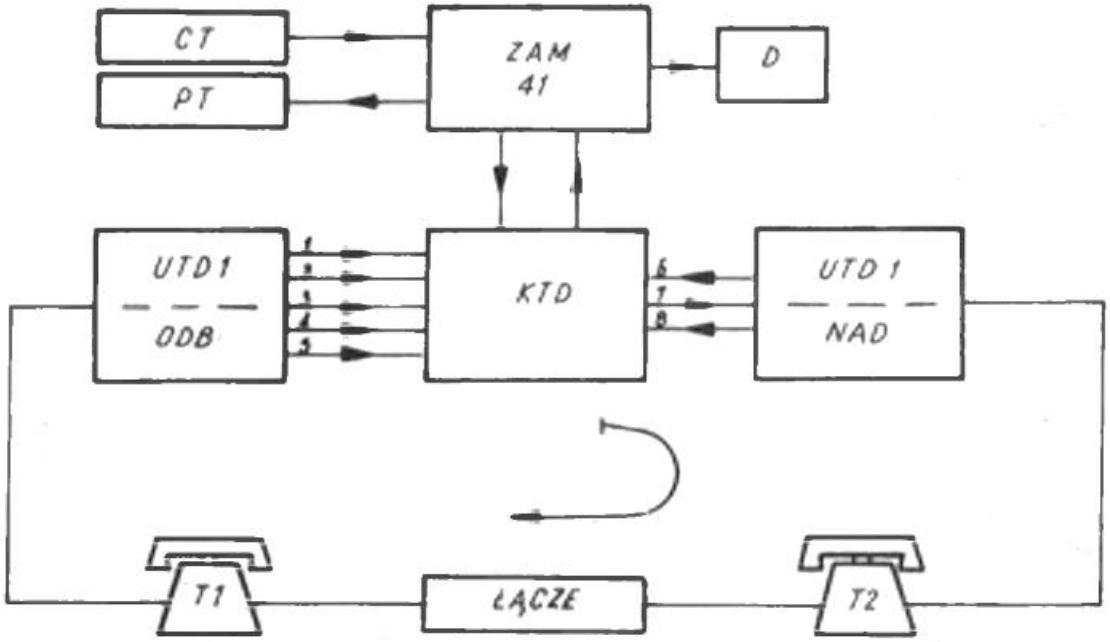
Jeżeli zadanie transmisji danych określimy jako przesłanie pewnej ilości informacji w określonym czasie przy jak najmniejszej stopie błędów, to przy opracowaniu systemu transmisji należy wybierać rodzaj łącza transmisyjnego (telegraficzne, telefoniczne, komutowane lub stałe, specjalne łącza szerokopasmowe, łącza bezprzewodowe), typ transmisji (synchroniczna lub asynchroniczna,równoległa lub szeregowa), typ modulacji (amplitudowa, fazowa lub częstotliwościowa) i szybkość modulacji (50, 100, 200, 600,1200,2400 bodów lub jeszcze wyższe - dla łączy specjalnych). Spełnienie warunku jak najmniejszej wrażliwości na zakłócenia osiąga się poprzez stosowanie układów detekcji lub korekcji błędów (kontrolę parzystości znakowej lub blokowej, różnego rodzaju kody cykliczne).

Budowa systemu

W latach 1966-1967 opracowano w Instytucie Maszyn Matematycznych eksperymentalny system transmisji danych z maszyną cyfrową ZAM-41 alfa. Celem pracy było zbadanie możliwości budowy w oparciu o maszynę cyfrową ZAM-41 alfa systemu transmisji danych wykorzystującego komutowane łącza miejskie i pozwalającego na synchroniczne, bezpośrednie wprowadzenie z łącza telefonicznego informacji binarnej do jednostki centralnej maszyny cyfrowej oraz na przesłanie w łącze przetworzonej informacji.

W skład systemu transmisji wchodziły (rys. 1):
- maszyna cyfrowa ZAM-41 alfa stanowiąca źródło i ujście danych
- urządzenie transmisji danych1(uTD1) o dwóch przełączalnych szybkościach modulacji 600 i 1200 bodów oraz układy stykowe pomiędzy maszyną cyfrową i urządzeniem transmisji danych, tworzące kanał transmisji danych (KTD; maszyny ZAM-41. Maszyna cyfrowa była wyposażona

w czytnik (CT) i dziurkarkę (PT) taśmy papierowej oraz szybką drukarkę wierszową (DW).



Rys. 1

System był połączony z miejską siecią telefoniczną za pośrednictwem dwóch telefonów (T1 i T2), informacja przesłana między urządzeniem transmisji danych (UTD1) a KTD miała postać słów trzydziestobitowych oddzielonych przerwami o długości odpowiadającej 9 bitom informacyjnym.

Po stronie odbiorczej UTD-1 było pięć par przewodów, na których odpowiednio pojawiały się:

przewód 1 - impulsy synchronizujące na wejściu KTD,
przewód 2 - słowa informacyjne wejściowe,
przewód 3 - impulsy, "brak błędu",
przewód 4 - impulsy, "błąd",
przewód 5 - sygnał "przerwa w transmisji".

Po stronie nadawczej KTD były 3 pary przewodów, na których pojawiały się:

przewód 6 - impulsy wywołania kolejnego słowa KTD,
przewód 7 - słowa informacyjne wyjściowe,
przewód 8 - impulsy synchronizujące na wyjściu KTD.

"* Urządzenie transmisji danych zostało opracowane w Katedrze Urządzeń Teletransmisyjnych i Telegraficznych Politechniki Warszawskiej.

Działanie systemu przy wprowadzaniu do maszyny cyfrowej słowa informacyjnego było następujące:

- Impulsy synchronizujące (1) przesuwały zawartość rejestru wejściowego KTD, do którego szeregowo wprowadzana była informacja (2).
-
- Po słowie informacyjnym urządzenie UTD1 generowało alternatywnie sygnały "błąd" (3) lub "brak błędu" (4), w zależności od poprawności przyjętej informacji.
-
- Pojawienie się sygnału "brak błędu" powodowało przerwanie programu maszyny i wczytanie słowa do pamięci operacyjnej.
-
- Jeżeli więcej niż połowa przychodzącej informacji była błędna lub miał miejsce zanik fali nośnej w łączu, pojawiał się stały poziom napięcia na przewodzie (5), sygnalizując przerwę w transmisji.
-
- Analogicznie działo się na wyjściu KTD, z tym że pojawienie się impulsu (6) powodowało przerwanie pracy jednostki centralnej i wpisanie do rejestru wyjściowego kolejnego słowa do wysłania w łącze.

Urządzenie transmisji danych (UTD1) pracowało z modulacją częstotliwości (częstotliwości środkowe: 1500 i 1700 Hz, dla szybkości modulacji 600 i 1200 bodów). Do detekcji błędów zastosowano kod cykliczny.

Oprogramowanie systemu

Prace programowe polegały na:

- opracowaniu programu użytkowego pozwalającego na określenie stopy błędów transmisji,
- modyfikacji systemu operacyjnego S0-141 oraz,
- opracowaniu testu kontrolno-lokalizującego dla KTD.

Program użytkowy składał się z dwóch wyraźnie wyróżnionych części:

- organizującej transmisję danych i
- drukującej tablicę słów błędnych po zakończeniu seansu transmisji.

Szerszego omówienia wymaga część pierwsza. W sieci można wyróżnić:

- główną pętlę nadawczo-odbiorczą oraz
- pięć niezależnych fragmentów stanowiących reakcję na przerwania programowe.

Działanie głównej pętli nadawczo-odbiorczej zapewniało następujący tok transmisji. Po wykręceniu na tarczy telefonu T2 numeru telefonu T1 następowało nawiązanie łączności i po naciśnięciu przycisku "Zgłoszenie

operatora" na module KTD, startowała główna pętla nadawczo-odbiorcza. Po stronie nadawczej wysyłany był tzw. tekst początkowy, a następnie ciąg liczb naturalnych.

Nadawanie ciągu liczb naturalnych miało na celu uproszczenie algorytmu kontroli poprawności transmisji. Pozwalało to bowiem na proste określenie kolejnych znaków, które powinny być odbierane w przypadku poprawnej transmisji. Po odbiorze tekstu początkowego główna pętla badała poprawność napływających kolejno znaków zapamiętując liczbę i postać zaistniałych błędów. Seans transmisji kończył się:

- po ponownym nienaciśnięciu przycisku "Zgłoszenie operatora",
- po przekroczeniu arbitralnie wybranej liczby słów błędnych,
- po przyjęciu przewidzianej w programie liczby słów.

Przerwania programowe, na które przewidziana była reakcja w programie użytkowym wywoływane były następującymi zdarzeniami:

1. rozpoczęcie seansu transmisji,
2. koniec kolejnych pięciominutowych odcinków seansu transmisji,
3. pojawienie się sygnału "Przerwa w transmisji",
4. pojawienie się sygnału "Koniec przerwy w transmisji",
5. zakończenie seansu transmisji.

Reakcja na przerwania programowe wywoływane zdarzeniami 1,3, 4,5 sprowadzała się do wydrukowania nazwy zdarzenia i czasu jego pojawienia się. Zdarzenie 2 związane było natomiast z rejestracją stanu systemu. Polegała ona na drukowaniu:

- czasu bieżącego,
- liczby słów nadanych,
- liczby słów odebranych,
- liczby słów poprawnych,
- liczby słów błędnych.

Modyfikacja systemu operacyjnego polegała na:

- dołączeniu programów obsługi KTD zapewniających nadawanie i odbieranie kolejnych słów w sposób synchroniczny,
- stworzeniu aparatu programowego umożliwiającego:
  - a) automatyczne wykrywanie zdarzeń związanych z przerwaniami programowymi,
  - b) przerywanie działania głównej pętli nadawczo-odbiorczej w momencie pojawienia się przerwania programowego i przekazanie sterowania właściwemu fragmentowi programu związanemu z tym przerwaniem.
  - c) przywróceniu sterowania głównej pętli nadawczo-odbiorczej po
  - zakończeniu reakcji na przerwanie programowe.

Test kontrolno-lokalizujący pozwalał na zbadanie poprawności pracy KTD wraz z UTD1, w zamkniętej pętli z maszyną ZAM-41. W wyniku testowania poszczególnych podzespołów generowane było "słowo kontrolne" wskazujące na określone uszkodzenia w KTD i UTD.

Badania systemu

Badania systemu transmisji danych trwały łącznie około 100 godzin, w tym 15 godzin z komutowanym łączem miejskim (10-kilometrowy odcinek Ochota-Grochów-Ochota). Stopa błędów przesyłanej informacji była równa $4*10^{-4}$, co odpowiada wartościom spotykanym w literaturze dla komutowanyoh łączy telefonicznych, przy szybkościach transmisji od 600 do 1200 bodów.

Wnioski

Opisana wersja KTD pozwala na podłączenie równolegle do maszyny ZAM-41 do 6 linii telefonicznych (praca dwukierunkowa), przy czym szybkość transmisji na wszystkich kierunkach byłaby równa 1200 bodom, a czynności związane z obsługą linii zajęłyby 32% czasu jednostki centralnej.

Przyjęte rozwiązanie staje się technicznie i ekonomicznie nieuzasadnione z chwilą podłączenia do maszyny ZAM-41 większej liczby linii transmisji danych (do kilkudziesięciu). Techniczną koniecznością staje się wówczas opracowanie bardziej rozbudowanego urządzenia stykowego wyposażonego w pamięć buforową, ze złożonym układem sterowania dokonującego wstępnej obróbki przyjmowanej informacji. Potrzebny jest również bezpośredni dostęp do pamięci operacyjnej przy pomocy tzw. kanału przesyłania, tak aby jednostka centralna inicjowała tylko proces przesłania do lub z pamięci szybkiej.

(J. Dańda)

Drugim, dużo większym przedsięwzięciem programowym w tej dziedzinie było opracowanie w latach 1967-1968 systemu oprogramowania maszyny ZAM-41 do pracy na bieżąco w sieci transmisji danych, w której dominującą rolę odgrywało zbieranie danych (ang. data acquisition).

System ten wykazywał wszystkie cechy istotne dla tej klasy zastosowań, a przede wszystkim:

- możliwość bezpośredniej współpracy z odległymi urządzeniami końcowymi;
- możliwość manipulowania informacją przesyłaną między urządzeniami końcowymi a jednostka centralną w postaci zwartych "paczek", zwanych dalej meldunkami, napływającymi w momentach przypadkowych;
- możliwość wyróżniania równych stopni ważności przesyłanych meldunków.

Generowanie konkretnego systemu oprogramowania - Język PJEG

Stefan SAWICKI
Instytut Maszyn Matematycznych

Język przystosowany do kodowania zagadnień określonego typu może w znacznym stopniu przyspieszyć wykonanie zadania, a sposób kompilacji zdań przez translator może decydować o strukturze organizacyjnej programu wynikowego i o jego efektywności.

Spośród zadań kodowanych w językach symbolicznych, ze względu na złożoność zagadnienia i specyfikę pracy programów, należy wyróżnić podstawowy system oprogramowania nowoczesnej maszyny. Język symboliczny przystosowany przede wszystkim do kodowania tego zagadnienia będę nazywać podstawowym językiem symbolicznym.

Jedną z cech podstawowego języka symbolicznego, którą pragnę zająć się szerzej, jest możliwość generowania oprogramowania dla konkretnej maszyny. Aby można było wygenerować z ogólnego systemu, system oprogramowania dla danej maszyny, język powinien mieć następujące cechy:

- możliwość podania, na początku programu, parametrów opisujących konfigurację konkretnej maszyny;
- posiadanie zdań języka pozwalających na wykonywanie obliczeń w trakcie translacji i tworzenie parametrów pochodnych;
- parametry lub wielkości obliczone w trakcie translacji mogą występować jako parametry rozkazów lub parametry innych zdań języka decydujących o przebiegu dalszej translacji;
- posiadanie zdań języka decydujących o rozmieszczeniu programów w pamięci pomocniczej i operacyjnej (parametrami tych zdań mogą być m.in. wielkości obliczane na etapie translacji);
- możliwość operowania licznikami translatora (np. określającymi rozmieszczenie słowa programu wynikowego w pamięci) wraz z możliwością ich zmiany oraz używania jako parametrów zdań sterujących translacją;
- posiadanie zdań warunkowych pozwalających, w zależności od wartości parametrów, na warunkowe pomijanie odcinków tekstu programu;
- posiadanie aparatu generowania nowych identyfikatorów (tzw. etykiet formalnych lub zmiennych formalnych), których postać jest określana w procesie translacji na podstawie parametrów aktualnie obliczonych;
- posiadanie aparatu translatora pozwalającego na wielokrotną translację określonego odcinka programu (m.in. na rekursywną translację), w połączeniu z warunkową translacją i możliwością tworzenia identyfikatorów pozwala to na generowanie konkretnych programów z programów ogólnych;
- możliwość wprowadzenia identyfikatorów lokalnych (lub nielokalnych) w określonym odcinku programu.

Poniższy przykład, napisany w języku PJEG wykonanym dla maszyny ZAM-41, podaje realizację kilku z powyższych wymagań.

```
VFB [B = 4095 - T (ALFA), Q = 20, E1]
 I5 = Q,
 VMA [I2 ,
     1:
       VRZ[T (ALFA), (2)]
          VRO : 2
       SKO ET (ALFA)
     T (ALFA) = T(ALFA)-
     VSK [I2]
    :1]
VOL [I2]
E1) STO 1
I5 = Q - 15
VKB
```

Blok programu ograniczony dyrektywami **VPB** i **VKB** (początek bloku, koniec bloku) zależy od parametru zewnętrznego **ALFA**. Parametry dyrektywy **VFB** określają że program ma wejść na bęben od miejsca 4095 minus wartość parametru **ALFA**,

```
B = 4095 - T (ALFA)
```

- przewidziane jest, że program będzie się zaczynał od stałego miejsca w pamięci operacyjnej 20,

```
Q = 20
```

- etykiety lokalne w tym bloku są **E0** i **E1**,

Dyrektywa **VMA** informuje translator, że tekst zawarty między sekwencjami znaków **:1** i **:1** będzie podlegał wielokrotnej translacji (powinien być przechowany) i adres tego tekstu jest **I2**.

Po napotkaniu dyrektywy **VOL [I2]** translator przerywa pobieranie tekstu z bieżącego wejścia i przechodzi do opracowywania tekstu o nazwie **I2**. Powrót do pobierania tekstu z wejścia poprzedniego następuje po napotkaniu dyrektywy **VRO**.

Zwrot **E1) STO 1** oznacza etykietę **E1** wypisaną przed rozkazem maszyny **STO 1**.

Wartość licznika **Q** wzrasta o jeden po wprowadzeniu informacji zajmującej jedno słowo w programie wynikowym.

Zdanie **I5 = Q** nadaje etykiecie nielokalnej **I5** wartość równą aktualnej wartości licznika **Q**, a zdanie **I5 = Q - 15** powoduje nadanie etykiecie I5 wartości, która jest długością odcinka programu zawartego między tymi dwoma zdaniami.

Jeżeli pierwszy parametr dyrektywy **VRZ** jest różny od zera (ALFA≠0), to tekst programu występujący po tej dyrektywie będzie pominięty aż do

miejsca określonego drugim parametrem (:2), w przeciwnym przypadku bieżący tekst będzie podlegał analizie.

SKO ET(ALFA) - rozkaz maszyny **SKO** z adresem określonym jako etykieta formalna zaczynająca się od litery **E** i o numerze określonym przez wartość parametru **ALFA**.

T(ALFA) = T(ALFA)-1 - zdanie zmniejszające wartość parametru **ALFA** o jeden.

Dyrektywa **VSK [I2]** powoduje przejście do translacji tekstu o nazwie **I2**.

Jeżeli teraz poprzedzimy omawiany odcinek programu zdaniem nadającym wartość parametrowi **ALFA** np.

```
T (ALFA) = 3
```

to otrzymany program będzie równoważny programowi:

```
VPB [B = 4092, Q = 20, E1]
     SKO E3
     SKO E2
     SKO E1
E1)  STO 1
     I5 = 4
VKB
```

(KB) Globalne zmienne generacyjne, takie jak **ALFA** znane były jako T-etykiety.

Symboliczne Operacje Wejścia i Wyjścia

Jerzy MYSIOR, Jacek WITASZEK
Instytut Maszyn Matematycznych

Decydując się na zainstalowanie nowej EMC użytkownik posiadający już pewien zestaw urządzeń przygotowywania i tabulowania danych w wypadku, gdy kod posiadanych urządzeń nie odpowiada wymaganiom nabytej maszyny, staje przed kłopotliwym problemem, jak wykorzystać dotychczasowe wyposażenie ośrodka. Problem ten staje się Poważniejszy w wypadku, gdy dany ośrodek współpracuje z dużą ilością innych zakładów wyposażonych we własne urządzenia tabulujące, perforujące itp.

Świadomość tego problemu w trakcie projektowania oprogramowania dla maszyn ZAM-41 pozwoliła na taką koncepcję jego rozwiązania, która uwalnia przyszłego użytkownika maszyn ZAM-41 od wspomnianych kłopotów. U podstaw tej koncepcji leżą tzw. symboliczne operacje wejścia i wyjścia.

Realizując tę koncepcję otrzymano elastyczne i oryginalne narzędzie, które pozwala generalnie rozwiązać problem różnorodności wymagań programów oraz ich użytkowników.

Operacje Wejścia-Wyjścia

Szczególnym rodzajem czynności wykonywanych przez program jest przesyłanie danych pomiędzy pamięcią operacyjną, rejestrami itp., a urządzeniami peryferyjnymi maszyny. Proces ten dotyczy zawsze pewnych porcji danych, które nazywać będziemy dalej kwantami, a które można traktować jako zbiory znaków. Z procesem przesyłania związane jest zawsze pewne przekształcanie danych. Jeżeli nazwać kwanty danych umiejscowione na urządzeniach zewnętrznych maszyny kwantami zewnętrznymi danych, natomiast kwanty umiejscowione w pamięci operacyjnej lub innych rejestrach maszyny kwantami wewnętrznymi, wówczas przesyłanie jest zawsze związane z transformacją danych, w której trakcie bądź kwanty wewnętrzne przechodzą w zewnętrzne, bądź zewnętrzne w wewnętrzne.

Zarówno zewnętrzne, jak i wewnętrzne kwanty danych można scharakteryzować podając wartości trzech następujących parametrów: struktura kwantu (s), kod znaków stanowiących elementy kwantów (k) oraz lokalizacja kwantu (l). Na przykład kwantem może być:
(1) blok 32 słów, z których każde zawiera jeden znak w kodzie KW-6, a początek bloku położony jest w PAO w miejscu 7362 lub
(2) rządek taśmy perforowanej zawierający 1 znak, w kodzie M-2, położony pod czytnikiem nr 3.

W ogólnym przypadku z przesyłaniem związana jest zmiana wartości wszystkich parametrów kwantów. Ograniczymy się tutaj do tych przypadków, gdy przesyłaniu towarzyszy transformacja danych, która ciąg kwantów o ustalonych własnościach s, k, l przekształca w ciąg kwantów o ustalonych własnościach s', k', l’. Transformację taką można scharakteryzować podając szóstką wartości s, k, l, s', k', l’ o podanych wyżej znaczeniach, którą nazywać będziemy charakterystyką transformacji.

Charakterystyka ta jest oczywiście funkcją czynników tak hardware'owych, jak i software'owych, zaangażowanych w realizację określonego przesłania danych.

Z punktu widzenia programu wykonanie przesyłania sprowadza się do wykonywania operacji zwanych operacjami wejścia-wyjścia (we-wy).

Możliwe jest oczywiście wykonywanie przez ten sam program różnych typów przesyłań, tzw. przesyłań, z którymi związane są transformacje o różnych charakterystykach.

Istotną cechą operacji we-wy jest to, że konkretna operacja związana jest zawsze z tą samą transformacją, tzn. charakterystyka transformacji określa funkcjonalne własności operacji we-wy, oraz że wykonanie jakiejkolwiek operacji powoduje przesłanie, a więc i transformację porcji danych zawartej w dokładnie jednym kwancie wewnętrznym (niezależnie od tego, czy na zewnątrz odpowiada temu jeden, kilka lub fragment kwantu). Wymogiem poprawnego funkcjonowania programu jest związanie z konkretną operacją wejścia-wyjścia określonych własności kwantu wewnętrznego danych.

Poza zakresem zainteresowania programu leżą własności kwantów zewnętrznych. Własności te są natomiast przedmiotem zainteresowań użytkownika programu, który będąc dostarczycielem danych wejściowych i odbiorcą danych wyjściowych, może narzucać w tym względzie różne, na ogół zmieniające się w czasie wymagania.

Problem uwzględnienia tych zmiennych wymagań, jak wynika z przeprowadzonych tutaj rozważań, można sprowadzić do problemu definiowania poza programem funkcjonalnych własności użytych w programie operacji we-wy, a w szczególności do definiowania wartości tych parametrów charakterystyki transformacji związanych z tymi operacjami, które określają własności zewnętrznych kwantów danych.

Symboliczne operacje WE-WY w systemie oprogramowania ZAM-41

Przykładem rozwiązania, które stanowi realizację idei operacji definiowanych poza programem są Symboliczne Operacje we-wy (SOWW) zrealizowane w Systemie Oprogramowania maszyny ZAM—41.

W tej realizacji program może dysponować 32 operacjami symbolicznymi, które dzielą się na dwie klasy po 16 operacji w każdej. Do pierwszej z tych klas należą operacje wejścia typu

WPROWADZ (n),

do drugiej operacje wyjścia typu

WYPROWADZ (m)

gdzie n i m są liczbami z zakresu 0-15 i stanowią identyfikator operacji w ramach jednej klasy. Podział na klasy z teoretycznego punktu widzenia nie jest oczywiście istotny, wynika on ze względów praktycznych - wygody programisty.

Funkcjonalne własności tych operacji (poza klasą operacji, a zatem określeniem lokalizacji kwantów stanowiących przedmiot i wynik transformacji związanej z daną operacją) stają się określone dopiero z chwilą przejścia do wykonania programu.

Program lub tworzącą pewną całość sekwencją programów nazywamy problemem. Każdy problem poprzedza tzw. czołówka problemu zawierająca jego opis, a w szczególności definicje SOWW dla programów problemu.

Oczywiście sensowność procesu przetwarzania wymaga, ażeby definicje uwzględniały zarówno wymagania programu odnośnie do własności kwantów wewnętrznych, jak i aktualne wymagania użytkownika odnośnie kwantów zewnętrznych.

Jak już wspomniano, w wykonanie operacji we-wy zaangażowane są zarówno elementy software'u, jak i hardware'u. Czynnikami, które organizując pracę tych elementów na rzecz SOWW realizują tym samym przekształcenia danych o określonych charakterystykach, są procedury we-wy. Procedury te w stanie nieaktywnym rezydują w Bibliotece Systemu.

Z każdą procedurą związana jest określona charakterystyka zrealizowanego przez nią przekształcenia danych.

W tym ujęciu każda występująca w czołówce definicja SOWW przyporządkowuje określonej poprzez identyfikator symbolicznej operacji określoną, również poprzez pewien identyfikator, procedurę.

Sam fakt wystąpienia w czołówce definicji określonej SOWW nie musi być w sensie Systemu Operacyjnego ZAM-41 równoznaczny ze zdefiniowaniem zgodnie z tą definicją danej symbolicznej operacji.

W każdym wypadku wymagane jest jeszcze tzw. dołączenie procedury dla danej SOWW. Pod tym pojęciem rozumiemy sprowadzenie procedury do pamięci operacyjnej oraz wstawienie odpowiedniej informacji do tabeli wejść procedur.

W maszynie istnieje specjalna operacja, której wykonanie powoduje dołączenie procedury, a której parametrami są klasa,identyfikator operacji, której przyporządkowana jest sprowadzana procedura, oraz adres początkowy obszaru pamięci operacyjnej,do którego procedurą tą należy sprowadzić.

Konieczność takiego rozwiązania podyktowana została wymogami oszczędnej gospodarki pamięcią operacyjną. Operacją dołączania może wykonać nie tylko program, ale na specjalne żądane w czołówce również i system operacyjny. Można więc w wypadkach, gdy nie ma potrzeby ograniczać się rozmiarami pamięci, sprawę dołączania przekazać systemowi operacyjnemu i wówczas z punktu widzenia programu sam fakt wystąpienia definicji w czołówce jest równoznaczny ze zdefiniowaniem odpowiednich SOWW. Należy tutaj zaznaczyć, że w czołówce wystarczy wyszczególnić definicje tylko tych SOWW, które są wykorzystywane w programach problemu.

Brak definicji danej SOWW w czołówce lub niedołączenie odpowiedniej procedury nie oznacza,że operacja ta nie jest zdefiniowana, wręcz przeciwnie - jest ona zdefiniowana jako operacja nielegalna ,i jako taka

ma określone własności funkcjonalne - jej wykonanie powoduje przerwanie pracy programu, który ją wywołał.

Możliwości różnorakiego definiowania SOWW są oczywiście ograniczone liczbą procedur zmagazynowanych w Bibliotece Systemu.

Zachodzi zatem pytanie, w jakim stopniu realizacja koncepcji SOWW jest opłacalna? Z każdą procedurą związane są określone nakłady pracy, każda procedura zajmuje określony obszar pamięci pomocniczej przeznaczonej na biblioteką.

Praktyczne doświadczenia wykazały, że wymagania tak programów, jak i użytkowników programów odnośnie do własności kwantów danych można zaspokoić stosunkowo niewielką liczbą standardów.

Stanie się to oczywiste, jeżeli wziąć pod uwagą, że wykorzystuje się zaledwie kilka kodów zewnętrznych, liczba lokalizacji zewnętrznych jest ograniczona liczbą urządzeń peryferyjnych w zestawie maszyny, a liczba struktur zewnętrznych jest niewielka zważywszy, że związane są one na ogół z urządzeniami (np. znak w rządku - dla czytników taśmy papierowej, blok 80 kolumn zawierających po jednym znaku w kolumnie dla czytnika kart itp.).

Jeżeli chodzi o wymagania odnośnie do własności kwantów wewnętrznych, okazało się, że w większości wypadków wystarcza jeden standard: s = znak w słowie, k = kod 6-bitowy KW-6, l = B -rejestr.

Obecnie wprowadza się drugi standard, który od poprzedniego różni się tym, że kod KW-6 zastąpiono kodem 8-bitowym KW-8.

W sumie więc koszty związane z implementacją idei SOWW nie okazały się zbyt wielkie, tym bardziej że udało się opracować taką organizację procedur we-wy, która pozwala na komponowanie procedur z pewnych bardziej uniwersalnych elementów. W Bibliotece Systemu znajdują się jedynie wspomniane komponenty, które odpowiednio składane w trakcie dołączania tworzą procedurę o żądanych własnościach.

Na zakończenie należy podkreślić, że opracowano formalne reguły pisania elementów procedur, wobec czego rozszerzenie możliwości definiowania SOWW nie przedstawia większych trudności.

JOM - Realizacja

(KB) Praktycznym wdrożeniem powyższej teorii stał się JOM – Język Operacyjny Maszyny, parę przykładów poniżej:

```
PROBLEM:  POPR,JOM;
WEJSCIA:  0 = PCTM(TMx),
          1 = PCTM(TMx),
          2 = PCTP(CPA,DM2C);
WYJŚCIA:  0 = PDW(DWA,DDW1),
          1 = PPTP(DPA,DM2P),
          2 = PPTP(DPA,DJED);
TYTUŁ: WYPROWADZENIE TAŚMY BINARNEJ.

PROBLEM: BIB2;
WEJŚCIA:  0=PCTP(CPA,DM2C),
          1=PCTP(CPA,DJED),
          2=PCTM(TMx);
WYJSCIA:  0=PDW(DWA,DDW1),
          1=PDW(TMy),
          2=PDW(Tmz).
```

(KB) – Zwrócić tu trzeba uwagę na definicje We-Wy urządzeń, które w naturze swojej mają możliwość wprowadzania i wyprowadzania danych (głównie taśmy perforowanej) w różnych kodach zewnętrznych i postaciach fizycznych (taśma 5-cio lub 8-mio kanałowa). Definicje takich wejść-wyjść składają się z dwóch parametrów, definicji urządzenia np. CPA i dekodera DM2C. Okazało się to, w sytuacji szerokiej różnorodności urządzeń pracujących z taśmą papierową, niezwykle elastycznym rozwiązaniem.

Gdy zaszła potrzeba wsparcia koleżanki piszącej pracę magisterską na Wydziale Matematyki UW, wymagającą programowania na MC GIER, wystarczyło napisać dekoder wejścia i wyjścia 8-bitowego kodu taśmy papierowej, wczytać program źródłowy na taśmę magnetyczną w systemie SMAD. Pozwoliło to uniknąć kolejek do dalekopisów w COPAN, wielokrotnego przepisywania programu po zmianach i w ogóle zdecydowanie przyspieszyć zadanie.

Symbole urządzeń We-Wy

Urządzenie	#pisanie	#czytanie	urządzenie
CPA		1	czytnik taśmy perforowanej
CPB		3	
DPA	0		dziurkarka taśmy papierowej
DPB	2		
DWA	4		drukarka wierszowa
BBS		5	bęben systemu
BBM	6	7	bęben użytkownika
TMA	8	9	jednostka taśmy magnetycznej
TMB	10	11	
TMC	12	13	
TMD	14	15	

TME	16	17	
TMF	18	19	
CKA		21	czytnik kart perforowanych

Podprogramy Wejścia-Wyjścia

PPUS	pusty
PWE	wejściowy wewnętrzny
PCTP	czytania taśmy papierowej
PCKP	czytania kart Hollerith (prosty)
PCKA	czytania kart Aritma (prosty)
PCK	czytania kart perforowanych
PCM	czytania monitora
PCT8	czytania znaków 8-bitowych z taśmy magnetycznej
PCT6	czytania znaków 6-bitowych z taśmy magnetycznej (selektywny)
PCTM	czytania znaków 6-bitowych z taśmy magnetycznej
PPUS	pusty
PWY	wyjściowy wewnętrzny
PPTP	pisania taśmy papierowej
PDW	wyjścia na drukarkę DW-1
PDV	wyjścia na drukarkę; DW-2
PPM	pisania monitora
PPT6	pisania znaków 8-bitowych na taśmę magnetyczną
PPT8	pisania znaków 8-bitowych na taśmę magnetyczną

Niektóre dekodery wejściowe na KW6

Nazwa	**nośnik**	**kod**
DM2C	taśma	M2
DFEC	taśma	Ferranti (nietypowy)
DFE6	taśma	Ferranti (standardowy)
D0L6	taśma	Olivetti
DBUL	karty	BULL
DBL6	karty	BULL2
DARN	karty	Aritma numeryczny
DARA	karty	Aritma alfanumeryczny
DAR6	karty	Aritma
DIBM	karty	IBM
DSAM	karty	SAM
DS46	karty	S4
D86C		kod wewnętrzny KW8

Niektóre dekodery wyjściowe z KW6

DM2P	taśma	M2
DFEP	taśma	Ferranti (nietypowy)
D6FE	taśma	Ferranti (standardowy)
D60L	taśma	01ivetti
DDW1	drukarka	DW-1 (prosty)
DDW2	drukarka	DW-1 (złożony)
DDV1	drukarka	DW-2 (prosty)
DDV2	drukarka	DW-2 (złożony)
DDU1	drukarka	DW-2E (prosty)

```
DDU2      drukarka  DW-2E (złożony)
D68P                kod wewnętrzny KW8
```

Niektóre dekodery wejściowe na KW8

```
DM28      taśma     M2
DFE8      taśma     Ferranti
D0P8      taśma     Optima
D0L8           taśma     Olivetti
DBU8           karty     BULL
DBL8      karty     BULL2
DAN8      karty     Aritma numeryczny
DIL8      karty     ICL
DIB8      karty     IBM
DSA8           karty     SAM
D68C                kod wewnętrzny KW6
D8R8                     kod wewnętrzny KW8R
```

Niektóre dekodery wyjściowe z KW8

```
D8M2      taśma          M2
D80P      taśma          Optima
D80L           taśma          Olivetti
D8SE      taśma          Soemtron
DDW8      drukarka       DW-1
DDV8      drukarka       DW-2
DDU8      drukarka       DW-2E
D86P                     kod wewnętrzny KW6
D88R                          kod wewnętrzny KW8R
```

Zagadnienie generowania systemu oprogramowania

Ewa Zaborowska
Instytut Maszyn Matematycznych

Ten sam model maszyny cyfrowej może być dostarczony użytkownikowi, w zależności od indywidualnych jego potrzeb, z różnym zestawem pamięci pomocniczych i urządzeń zewnętrznych a musi być rzecz jasna zaopatrzony w System Operacyjny dostosowany do tych urządzeń i pamięci. To samo dotyczy translatorów języków programowych i innych elementów oprogramowania: na życzenie użytkownika dostarczony mu System Oprogramowania może zawierać tylko niektóre z oferowanych translatorów. Ponadto, już w czasie eksploatacji otrzymanego Systemu Oprogramowania może zaistnieć potrzeba innego niż standardowe rozmieszczenia pewnych elementów systemu w pamięciach pomocniczych, np. aby uzyskać szybszy dostęp do częściej wykorzystywanych translatorów.

Z wyżej wymienionych powodów mnożą się wersje Systemu Oprogramowania dla danej maszyny. Ponieważ sporządzanie dokumentacji dla wielu wersji Systemu, ich konserwacja i rozbudowa jest rzeczą bardzo niewygodną i pracochłonną - powstał problem automatycznej generacji Konkretnego Systemu Oprogramowania, dostosowanego do aktualnych potrzeb użytkownika na bazie - uzależnionego od parametrów pisujących konfigurację maszyny - Ogólnego Systemu Oprogramowania, który to Ogólny System jako całość jest szczegółowo udokumentowany oraz podlega konserwacji i rozbudowie.

Poniżej podane będzie rozwiązanie tego problemu dla maszyny ZAM—41. Produkowanie Konkretnego Systemu Oprogramowania (KSO; dzieli się na dwa zasadnicze etapy:

1. Generowanie KSO na etapie translacji Ogólnego Systemu Oprogramowania.
2. Adaptacja wygenerowanego KSO, zawartego w tzw. Bibliotece Systemu Oprogramowania do zadań aktualnie wykonywanych.

Problemy związane z punktem 1 poruszone będą osobno przy omawianiu podstawowego języka symbolicznego (na przykładzie języka PJEG), którego translator pełni rolę generatora KSO. Obecnie zajmiemy się zagadnieniem adaptacji już wygenerowanego ESO (pkt 2).

Wyprodukowany na etapie translacji Konkretny System Oprogramowania przechowywany jest w Bibliotece Systemu Oprogramowania (BSO). Elementami BSO są tzw. zbiory i ich metryki. Metryka zbioru określa jego nazwę, wielkość, położenie w pamięci operacyjnej lub pomocniczej i inne związane z tym zbiorem parametry. Ze względu na swoją strukturą zbiory dzielą się na zwarte - stanowiące jednolity blok informacji, i złożone - składające się z tzw. podzbiorów.

Podzbiory również mają nazwę i stanowią już jednolity blok informacyjny. Ze względu na zawartość istnieją zbiory typu program i zbiory typu tekst. Z BSO związany jest tzw. Katalog będący elementem biblioteki i zawierający jednocześnie nazwy i tzw, adresy biblioteczne wszystkich zawartych w niej zbiorów.

Bibliotekę SO tworzy program zwany Bibliotekarzem,który współpracując z translatorem PJEG lub innymi translatorami przetwarza teksty lub programy zapisane w języku odpowiednim dla tych translatorów na formę przewidzianą dla elementów biblioteki. Bibliotekarz korzysta dodatkowo z informacji o przetwarzanym materiale zapisanym w Języku Bibliotekarza (JB).

Proces tworzenia BSO ilustruje poniższy przykład: Program Bibliotekarz otrzymuje następujące informacje zapisane w języku JB:

```
ZBIOR : ORG - Y, TYP= PROGRAM, NZBIORU = ALGOL;
PODZBIÓR : NPZ= AL1;
TRANSLACJA : TRA = PJEG;
.
.
.
PODZBIOR : NPZ = AL7;
TRANSLACJA : TRA = PJEG;
ZBIÓR : ORG = X, TYP = TEKST, NZBIORU - DOKUMENT;
TRANSLACJA : TRA = POPR;
KONIEC;
```

W wyniku pracy Bibliotekarza w BSO znajdzie się zbiór złożony (ORGANIZACJA = Y) typu program, o nazwie ALGOL, składający się z siedmiu podzbiorów o nazwach AL1,..,AL7, przetłumaczonych na formę binarną przy pomocy translatora PJEG, oraz zbiór zwarty (ORG = X) o nazwie DOKUMENT będący tekstem opracowanym przez translator POPR. Oba zbiory będą poprzedzone metrykami zawierającymi charakterystykę tych zbiorów, przekazaną częściowo poprzez deklaracje języka JB, częściowo przez wywołane translatory. W Katalogu zostaną zapisane nazwy i adresy obu zbiorów.

Do przygotowanych w wyżej zilustrowany sposób elementów KSO, stanowiących zbiory lub podzbiory zawarte w BSO, można się odwoływać podając jedynie nazwę danego elementu. System Operacyjny i inne programy organizacyjne sprowadzające z BSO do pamięci operacyjnej lub bębnowej translatory, ich części lub inne elementy oprogramowania posługują się operacjami typu:

```
PODAJ ZBIOR (nazwa zbioru)
PODAJ PODZBIOR (nazwa zbioru, nazwa podzbioru)
PODAJ METRYKE (nazwa zbioru, adres pamięci op)
```

Operacje te na podstawie informacji zawartych w Katalogu oraz w metryce zbioru wybierają z BSO element o wskazanej nazwie i przesyłają go do pamięci operacyjnej (podzbiór, metryka zbioru) lub do pamięci bębnowej (zbiór).

Wszystkie operacje współpracy z BSO mają, jak widać, formę niezależną od rodzaju pamięci pomocniczej, w której zapisana jest BSO. Biblioteka zapisana w przeznaczonej dla Systemu Oprogramowania części pamięci bębnowej, zwanej Bębnem Systemu (BS), miałaby duże zalety ze względu na szybki dostęp do poszczególnych jej elementów.

Ponieważ jednak pamięć bębnowa jest stosunkowo mało pojemna (w maszynie ZAM-41 - 64 tys. słów), BSO jest tworzona na taśmie magnetycznej zwanej Taśmą Magnetyczną Systemu (TMS). Istnieje jednak możliwość umieszczenia, chwilowo, wybranych zbiorów na BS, co znacznie przyspiesza ich sprowadzanie do pamięci operacyjnej.

Dostosowywanie BSO do aktualnych potrzeb odbywa się na etapie inicjowania pracy Systemu Operacyjnego. Na dany przez System Operacyjny sygnał (odpowiedni tekst wypisany na monitorze) operator powinien wyklawiszować na monitorze nazwy zbiorów, które jak przewiduje, będą często w najbliższym czasie wywoływane (np. nazwę ALGOL, jeśli dostarczono mu w danym dniu wiele programów napisanych w ALGOL-u).

Odpowiedni program adaptujący przeniesie wówczas wskazane zbiory wraz z metrykami z TMS na BS (o ile tylko maksymalna pojemność BS na to pozwoli) odnotowując ten fakt w Katalogu i metrykach zbiorów. Zbiory te będą potem sprowadzane operacjami typu PODAJ ZBIÓR wprost z pamięci bębnowej. Pozostałe, czyli te, których nazwy nie zostały wymienione przez operatora, lub te, które nie zmieściły się na BS, będą sprowadzone z TMS.

Powstały w ten sposób układ, nadający priorytet pewnym elementom oprogramowania kosztem innych, jest chwilowy i jeśli zajdzie potrzeba, może być łatwo zmieniony przy ponownym wykonaniu programu adaptującego. Na zakończenie trzeba jeszcze podkreślić, że wspomniane wyżej problemy powstały w wyniku doświadczeń naszego zespołu (J. Swianiewicz, J. Mysior, S. Sawicki, J. Witaszek, E. Zaborowska) przy oprogramowywaniu maszyny ZAM-41 i wszystkie rozwiązania, jak to już było powiedziane, opracowywano pod kątem tej właściwie maszyny. Niemniej jednak wydaje się, że są to zagadnienia natury ogólniejszej i dlatego próbujemy je tutaj bardzo skrótowo zasygnalizować.

System Oprogramowania i jego Dokumentacja

Jerzy Swianiewicz
Instytut Maszyn Matematycznych

W niniejszym opracowaniu pragną zaproponować kilka podstawowych definicji dotyczących systemu oprogramowania jego dokumentacji. Chodzi mi przede wszystkim o ustalenie zakresów znaczeniowych dla omawianych pojąć, jakkolwiek ustalenie terminologii w tej dziedzinie wydaje się również bardzo ważne.

Mimo że omawiane niżej pojęcia odnoszą się głównie do systemów oprogramowania, to wydaje się, że mogłyby one być również zastosowane do wszelkich innych złożonych systemów programowych.

System oprogramowania

Przyjmuję następującą, roboczą definicję Systemu Oprogramowania: System Oprogramowania pewnej konkretnej maszyny cyfrowej jest to taki zbiór programów tej maszyny, które za pomocą programu System Operacyjny zostały połączone w jeden program stale rezydujący w specjalnie wydzielonych pamięciach tej maszyny i stale funkcjonujący podczas jej eksploatacji. Mowa o wydzielonych dla systemu pamięciach ma tu sugerować wymaganie niezniszczalności elementów tego systemu w czasie jego eksploatacji. Zdając sobie sprawę z wielu nieścisłości zawartych w powyższej definicji, przechodzę do właściwego tematu licząc na to, że intuicyjne rozumienie tej definicji będzie u większości słuchaczy podobne.

Dokumentacja systemu oprogramowania

Proponuję podział dokumentacji systemu oprogramowania na faktyczną i ideową. Treść tych pojęć opisana jest poniżej.

1. Dokumentacja faktyczna

1.1. Plik źródłowy

Zasadniczym elementem dokumentacji faktycznej jest treść oprogramowania. stanowiąca formalny opis oprogramowania w określonym języku dokumentacyjnym. Językiem dokumentacyjnym może być język, dla którego istnieje translator działający na maszynie cyfrowej. Treść oprogramowania jest zapisana na maszynowym nośniku informacji (np. taśmie papierowej, kartach perforowanych, taśmie magnetycznej itp.) tworząc plik źródłowy. Plik źródłowy jest centralnym elementem dokumentacji faktycznej.

Kryterium uznania pliku źródłowego jako elementu dokumentacji faktycznej jest posiadanie mechanizmu programowego pozwalającego

na automatyczne (tzn. za pomocą MC) przekształcenie danych zawartych w pliku źródłowym na funkcjonujący we właściwej maszynie System Oprogramowania.

1.2. Dokumenty związane

Oprócz pliku źródłowego do dokumentacji faktycznej zaliczamy wszelkiego rodzaju dokumenty powstające automatycznie (np. drogą programową) z danych zawartych w pliku źródłowym. Dokumenty te ogólnie nazywamy dokumentami związanymi.

Wśród dokumentów związanych, jako podstawowy, wyróżniamy tzw. tabulogram oprogramowania. Jest to dokument drukowany, zawierający zapis zawartości pliku źródłowego w języku dokumentacyjnym, otrzymany drogą programową (np. wydruk na drukarce wierszowej) bądź mechaniczną (wydruk za pomocą urządzenia tabulującego taśmę dziurkowaną bądź karty dziurkowane) z pliku źródłowego.

Inne dokumenty związane pełnią z zasady rolę pomocy ułatwiających operowanie tabulogramu oprogramowania. Będą to więc różnego rodzaju indeksy, słowniki etykiet, skorowidze etykiet określające rozmieszczenie w pamięciach poszczególnych elementów oprogramowania ich strukturę blokową rozmieszczanie na tabulogramie oprogramowania itp.

1.3. Programy manipulacyjne

Programy manipulacyjne są to programy, za pomocą których uzysku je się dokumenty związane z pliku źródłowego. Powinny być one dołączone do dokumentacji faktycznej wraz ze zwięzłymi instrukcjami ich obsługi.

1.4. Uwagi ogólne

Celem użytkowym dokumentacji faktycznej jest ułatwienie nanoszenia poprawek i uzupełnień do programów systemu oprogramowania.
Ze względu na konieczną wiarygodność tej dokumentacji istotne jest wyeliminowanie czynnika ludzkiego przy jej otrzymywaniu. Wartość dokumentacji faktycznej jest tym większa, im bardziej komunikatywny jest zastosowany język dokumentacyjny oraz przyjęta forma tabulogramu oprogramowania, im przejrzystszy jest opis komentarzowy poszczególnych programów oraz doskonalsze dokumenty związane.

2. Dokumentacja ideowa

2.1. Opis funkcjonalny

Podstawowym dokumentem dokumentacji ideowej jest opis funkcjonalny systemu oprogramowania. Opis ten zawiera wszystkie informacje interesujące użytkownika maszyny zaopatrzonej w dany system oprogramowania. Podaje on właściwości, jakie uzyskuje maszyna w efekcie zaopatrzenia jej w system oprogramowania, nie musi natomiast określać sposobu ich realizacji. Opis taki powstaje zwykle w wyniku konfrontacji założeń na System Oprogramowania z efektem uzyskanym w drodze realizacji.

2.2. Opisy realizacji

Do dokumentacji ideowej zaliczane są tzw. opisy realizacji. Są

to dokumenty określające sposób realizacji właściwości systemu oprogramowania opisanych w opisie funkcjonalnym. Dokumenty takie stanowią ogniwo łączące między opisem funkcjonalnym a tabulogramem oprogramowania. Powinny one umożliwić odnalezienie w tabulogramie oprogramowania elementów realizujących poszczególne funkcje.

(KB)

Koncepcja ta, zwana później Technologią Woluminów Dokumentacyjnych została adaptowana i rozszerzona dla systemów IBM i JS-ES i kolejnych wersji systemów operacyjnych OS, MVS, VS1 i z/OS IBM przez zespół (BKS): K.Bytnerowicz, Marian Skupiński i Zbigniew Kosowski.

ZAM-41 - LISTA INSTALACJI MASZYN

A	B	C	D	E	F	G	H	I
#	Instytucja	Miasto	PAO (K)	PB (K)	#PT	CK	DW	Uwagi
1	Biuro Projektów Przemysłu Syntezy Chemicznej "Prosynchem"	Gliwice	20	64	7	CK3	DW1, DW2E	
2	Zakład Elektronicznej Techniki Obliczeniowej Instytutu Zootechniki	Kraków, Balice	20	64	7	CK3 BETA	DW2	
3	Miejskie Przedsiębiorstwo Komunikacji, Ośrodek Elektronicznej Techniki Obliczeniowej	Łódź	20	32	7	CK3 BETA	DW2E	
4	Zakład Elektronicznej Techniki Obliczeniowej, 'ZETO'	Łódź	20	64	7	CK3 BETA*2	DW2	
5	Branżowy Ośrodek Zastosowań Elektronicznej Techniki Obliczeniowej Zjednoczenia Przemysłu Farmaceutycznego POLFA	Warszawa	20	64	7	CK3 BETA	DW2	
6	Dział Przetwarzania Informacji Huty Warszawa	Warszawa	20	64	7	CK3	DW2E	
7	Ośrodek Obliczeniowy Instytutu Technicznego Wojsk Lotniczych, 'ITWL'	Warszawa	20	64	7	CK3	DW2E	
8	Ośrodek Obliczeniowy Wojskowej Akademii Technicznej	Warszawa	20	128	7	CK3, CK3 BETA	DW1, DW2E	
9	Stołeczny Ośrodek Elektronicznej Techniki Obliczeniowej, 'SOETO'	Warszawa	20	64	7		DW2	
10	Stołeczny Ośrodek Elektronicznej Techniki Obliczeniowej, 'SOETO'	Warszawa	20	64	7	CK3 BETA*2	DW2E	
11	Zakład Elektronicznej Techniki Obliczeniowej Instytutu Ekonomiki Rolnej	Warszawa	20	96	5	CK3 BETA	DW2	
12	Ministerstwo Spraw Wewnętrznych	Warszawa, Pyry	20	64	7		DW2	Przypuszczalna konfiguracja
13	Instytut Informatyki, Wydział Elektroniki, Politechnika Gdańska	Gdańsk	20	64	7		DW2	Przypuszczalna konfiguracja
14	Zakład Elektronicznej Techniki Obliczeniowej, 'ZETO'	Wrocław	20	64	7		DW2	Przypuszczalna konfiguracja
15	Politechnika Gdańska	Gdańsk	20	64	7		DW2	Przypuszczalna konfiguracja
16	??							c

(KB) Jak wynika z powyższej listy, maszyny ZAM-41 znalazły zastosowanie w kilkunastu ośrodkach obliczeniowych różnych branż. Opracowano tam wiele zastosowań z dziedziny obliczeń inżynierskich, naukowych, przetwarzania danych oraz pracy w czasie rzeczywistym (ITWL).

Wymienienie tego oprogramowania wykracza poza ramy tego opracowania, nie wspominając już o trudności uzyskania informacji.

Centralna Kartoteka Programów publikowana przez Centrum Obliczeniowe PAN zawiera, między innymi, trochę informacji o programach opracowanych na maszyny ZAM-41. Jest to, niewątpliwie informacja niepełna.

TROCHĘ OSOBISTYCH UWAG

Terminologia

Włoszczyzna

Programiści pracujący nad systemem SO-41,14 posiadali, jak zwykle, swoistą terminologię. Nieraz w dyskusjach słyszałem pojęcie ‘włoszczyzna’. Nie miało ono jednak nic wspólnego z gotowaniem, był to termin określający fragmenty Dyrygenta obsługujące poszczególne typy urządzeń zewnętrznych. Sam wprowadzałem zmiany do włoszczyzny taśmowej. Coś jak ‘device driver’.

Tracący czas

Gówna pętla Dyrygenta, czekająca na przerwania opatrzona była komentarzem ‘tracący czas’.

Post Mortem

Awaryjne zakończenie programu użytkownika, o ile było zdefiniowane wyjście 0 na Drukarkę Wierszową powodowało wydrukowanie informacji diagnostycznej zatytułowanej ‘Post Mortem’, inaczej ‘Po Śmierci’.

Ściągaczka systemu

Był to odcinek pięciokanałowej taśmy perforowanej, zawierający binarny zapis programu działającego na ‘surowej maszynie’, wczytywany do maszyny za pomocą stolika operatora powodujący załadowanie systemu SO-141 z taśmy systemu lub bębna magnetycznego, o ile był na nim uprzednio zapisany. System SO-41 był ładowany z taśmy papierowej pięcio lub ośmio kanałowej.

Podprogram za Prince Polo

Gdy okazało się w SOETO, że opanowaliśmy z kolegami programowanie w SASie na tyle dobrze, że nasze programy i podprogramy były szybsze i bardziej zwarte niż firmowe, otworzyły się perspektywy handlowe. Zaczęliśmy przyjmować zamówienia na podprogramy, cena: batonik Prince Polo za podprogram. Zaczęliśmy ten proceder na ZAM-2 i przenieśliśmy go na ZAM-41 do czasu przejścia do producenta (IMM).

Sobie ukochanemu ten fragment kodu poświęcam

Pracując kiedyś nad zmianami w Dyrygencie SO-141 natrafiłem na fragment kodu, chyba we włoszczyźnie taśmowej, na następujący komentarz: „Sobie ukochanemu ten fragment kodu poświęcam”. Utkwiło mi to, jak widać, na długo w pamięci. Czyj był to komentarz, ma na ten temat pewne podejrzenia, ale, oczywiście mogę się mylić, więc przemilczę.

Oczywiście, chyba każdemu programiście zdarzało się napisać wesoły lub osobisty komentarz, ten jednak jest wyjątkowy.

IRIS 50 a sprawa polska

Dostałem kiedyś do przejrzenia dokumentację systemu operacyjnego SIRIS francuskiej maszyny IRIS 50, w oryginale, oczywiście. Posiadałem ograniczoną znajomość języka, więc sobie poczytałem. Nie pamiętam czym było dyktowane zainteresowanie IMM tą maszyną, ale nasunęły mi się następujące wnioski:
- Francuzi, jak i IMM, niezwykle dbali o terminologię w swoim języku, chyba wszystkie terminy były francusko-języczne.
- Francja, podobnie jak wtedy Polska, próbowała opracować swoje własne maszyny cyfrowe, niezależnie od większej skali i, niewątpliwie większych środków finansowych, projekt ten (Plan Calcul zatwierdzony przez Prezydenta De Gaulle w roku 1966) jednak po pewnym czasie padł. Podobnie jak ZAM, maszyny IRIS miały stanowić rodzinę, wyprodukowano modele 50 i 80. Plan zakończył się jak ZAM, firma CII (producent) została wchłonięta przez Honywell-Bull.

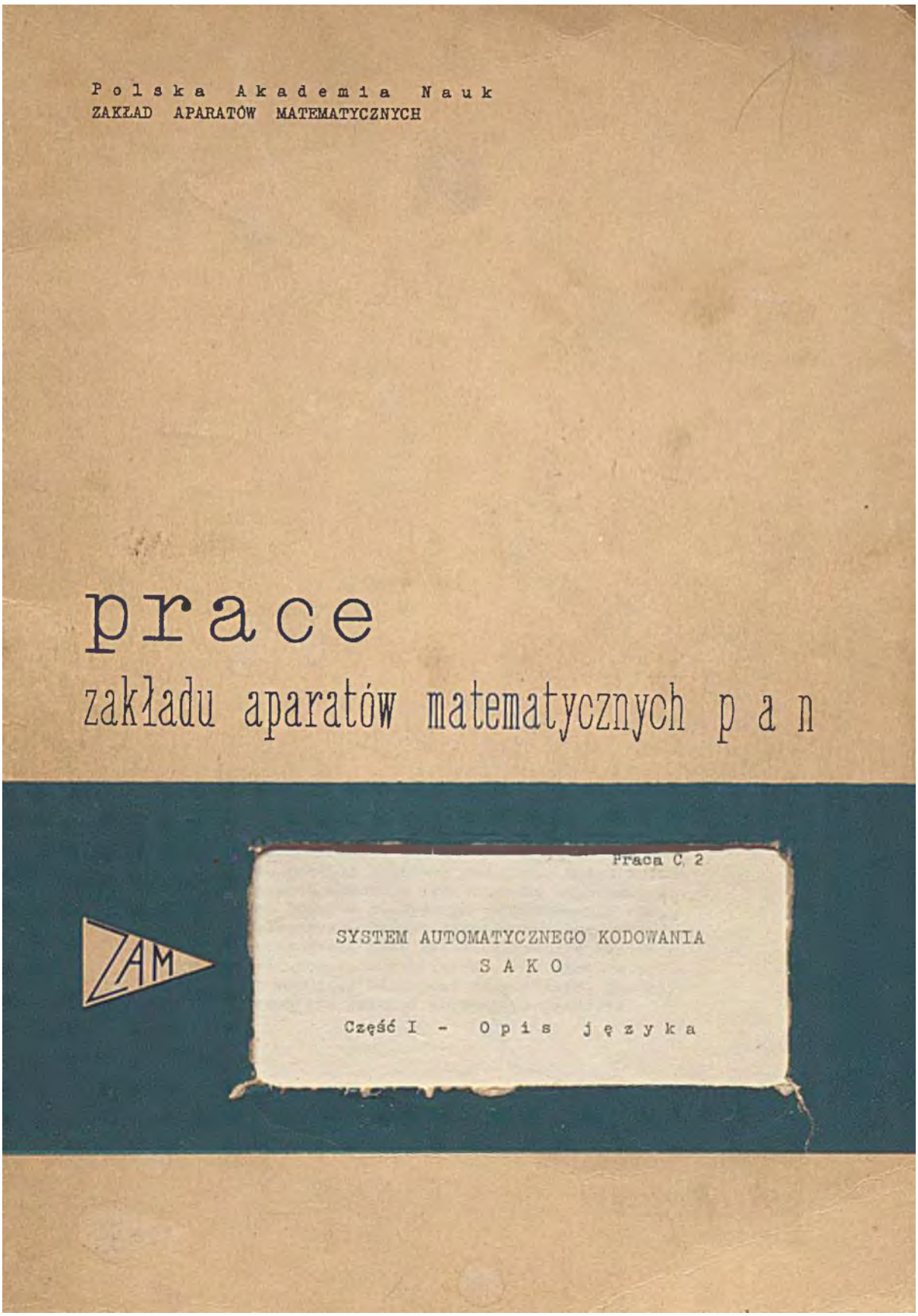
KISR i W.M. Turski – świat jest mały

Siedzieliśmy sobie kiedyś ze Swianem w KISR (Kuwait Institute of Scientific Research) w mojej ‘piwnicznej izbie’, miałem pół pokoju w podziemiach, chałturzyłem tam wieczorami. O czymś tam rozmawialiśmy, miałem drzwi otwarte na korytarz, gdy zauważyliśmy przechodzącą znajomą sylwetkę. Zawołaliśmy zgodnie ‘panie profesorze’, sylwetka się cofnęła i weszła na chwilę. Był to Profesor W.M. Turski, były (między innymi) zastępca dyrektora IMM. Zdziwił się trochę skąd myśmy się wzięli, porozmawiał z nami przez chwilę i poszedł dalej, był w składzie jakiejś międzynarodowej delegacji.

Gdyby ktokolwiek z czytających miał jakąś uwagę, wspomnienie lub anegdotkę, autor będzie niezwykle wdzięczny za podzielenie się ją.

DODATKI

System Automatycznego Kodowania – SAKO



System Automatycznego Kodowania SAK0 opracowany został w latach 1959 - 1960 przez zespół następujących pracowników naukowych Zakładu Aparatów Matematycznych PAN

Leon Łukaszewicz
Antoni Mazurkiewicz
Jan Borowiec
Jerzy Swianiewicz
Jowita Koncewicz
Piotr Szorc
Maria Łącka
Alfred Szurman
Stefan Sawicki
Andrzej Wiśniewski

Ponadto w pracach nad SAKO udział wzięli:

Ludwik Czaja
Jacek Witaszek
Danuta Kosecka
Ewa Zaborowska

Podręcznik opracowali:

Leon Łukaszewicz
Antoni Mazurkiewicz

SAKO umożliwia automatyczne kodowanie programów, ułożonych dla maszyn XYZ oraz ZAM-2. Programy te, napisane w łatwym do opanowania języku

SAKO,

są następnie sprowadzane do postaci ostatecznej przez odpowiedni program kodujący, zapisany na stałe w pamięci bębnowej maszyny. W ten sposób sama maszyna wykonuje pracochłonne czynności kodowania, które w klasycznym programowaniu obciążały człowieka. Dzięki temu łączny czas przygotowania programu może być wielokrotnie skrócony.
Sprawność programu wynikowego, uzyskanego przez program kodujący SAKO jest na ogół taka, jak gdyby program ten wykonał sprawny programista.

WSTĘP

System Automatycznego Kodowania - SAKO opracowany został celem znacznego zmniejszenia wysiłku i czasu, potrzebnych dla przygotowania programów. Jego zasadniczą rolą jest pełnienie funkcji tłumacza pomiędzy programistą a maszyną. Z jednej strony programista zapisuje program w formie podobnej do ogólnie przyjętej w matematyce, z drugiej strony maszyna otrzymuje ścisłe instrukcje dotyczące realizacji rozwiązania zadanego problemu. PROGRAM bowiem jest to jednoznaczny opis czynności maszyny, potrzebnych dla przetworzenia informacji wejściowych - danych w informacje wyjściowe - wyniki.

Dla rozwiązania na maszynie cyfrowej każdego problemu należy wykonać dwa zadania:

- sformułować metodę rozwiązania problemu,
- zakodować tę metodę w języku zrozumiałym dla maszyny.

O ile pierwsze z tych zadań wymaga od programisty umiejętności i inteligencji w stopniu uwarunkowanym trudnością problemu, zadanie drugie wymaga tylko wiadomości o charakterze raczej formalnym i odpowiedniej wprawy w kodowaniu.

Programem zakodowanym w języku maszyny nazywamy taki jego zapis, w którym wyszczególniony jest każdy rozkaz maszyny składający się na ten program. Przykładem takiego języka jest np. język SAS, opracowany dla maszyny XYZ i maszyny ZAM-2.

Z takim systemem kodowania związane są liczne trudności, a mianowicie:

- duża ilość czasu, potrzebna na wykonanie zadania i przekraczająca najczęściej czas potrzebny na sformułowanie metody,
- wynikający z powyższego długi okres czasu pomiędzy momentem postawienia zadania a jego rozwiązaniem,
- niedostateczna /znikoma/ przejrzystość zakodowanego programu,
- niemal nieuniknione, liczne omyłki w programie,
- konieczność bezproduktywnej pracy maszyny przy uruchamianiu programu.

Wobec tych trudności nic dziwnego, że problemy kodowania stanowiły dotąd jeden z najpoważniejszych czynników, hamujących szersze stosowanie elektronowych maszyn cyfrowych.

Stworzenie Systemu Automatycznego Kodowania SAKO, podobnie jak innych autokodów, stanowi próbę przezwyciężenia tych trudności. W systemie takim programista zapisuje metodę rozwiązania problemu przez maszynę w specjalnym języku sformalizowanym , zbliżonym do zapisu stosowanego powszechnie w matematyce. Nauka i posługiwanie się tym językiem nie przedstawia specjalnych trudności. Na podstawie zapisu w SAKO maszyna sama koduje program w języku maszyny.

Programem zakodowanym w języku autokodu nazywamy jego zapis przy pomocy symboli i rozkazów uogólnionych, niemających na ogół bezpośredniego znaczenia dla maszyny. Program zapisany w autokodzie

musi być dopiero przetłumaczony na program wynikowy, zapisany w języku maszyny. Dokonuje tego specjalny program kodujący, zapisany na stałe w pamięci bębnowej maszyny. W stosunku do programu kodującego program zapisany w autokodzie stanowi dane wejściowe, natomiast wynikiem jest ten sam program, zapisany w języku maszyny.

Program kodujący dla systemu SAKO składa się z około 5000 rozkazów w języku maszyny. Jego obszerność i złożoność spowodowana jest faktem, że zastępuje on całkowicie doświadczonego i sprawnego programistę.

Autokod pozwala zatem na szerokie stosowanie maszyn cyfrowych np. w przemyśle. Czas bowiem potrzebny na wyszkolenie personelu programującego maszynę w systemie SAKO jest wielokrotnie krótszy od czasu potrzebnego do wyszkolenia programistów, programujących w języku maszyny. Pozwala to specjalistom z różnorodnych dziedzin na przyswojenie sobie zasad pracy w autokodzie w ciągu kilku dni i tym samym otwiera szerokie perspektywy stosowania maszyn cyfrowych do analizy problemów powstających w trakcie prac badawczych.

W podręczniku zawarty jest opis języka SAKO, ułożonego dla maszyn XYZ i ZAM-2.

Ogólna struktura programów w języku SAKO

Programy, napisane w języku SAKO, składają się z ciągu zdań. Rozróżniamy kilkadziesiąt różnych typów zdań SAKO, z których każdy ma określone znaczenie oraz ustaloną formę, a posługiwanie się każdym zdaniem ujęte jest w szereg reguł.
Język SAKO jest więc językiem sformalizowanym.

Przetwarzanie informacji przez maszynę składa się z dwóch zasadniczych etapów:

- wprowadzenie programu do maszyny,
- wykonanie wprowadzonego programu przez maszynę.

Z tego punktu widzenia dzielimy wszystkie zdania SAKO na następujące kategorie:

- **Deklaracje** - podają informacje dotyczące budowy programu oraz ustalają znaczenie używanych dalej symboli. Deklaracje spełniają swą rolę przy wprowadzaniu programu do maszyny, są natomiast pomijane przy wykonywaniu programu przez maszynę.

- **Rozkazy** - określają czynności maszyny w trakcie wykonywania programu.

- **Komentarze** - mają jedynie charakter objaśnień poszczególnych fragmentów programu i są pomijane zarówno przy wprowadzaniu, jak i przy wykonywaniu programu przez maszynę.

Należy rozróżniać kolejność wypisania rozkazów przez programistę od kolejności wykonania tych rozkazów przez maszynę. Na ogół różnią się one między sobą.

Kolejność wypisania rozkazów, obok innych zdań SAKO zgodna jest z kolejnością ich wprowadzania do maszyny.

Kolejność wykonania rozkazów określona jest następująco:

1. pierwszym rozkazem, wykonanym przez maszynę jest pierwszy rozkaz, wypisany w programie;
2. po wykonaniu dowolnego rozkazu, który nie jest rozkazem sterującym, maszyna przechodzi do rozkazu następnego, to jest najbliższego w kolejności wypisania;
3. przejście do rozkazu o innej kolejności wypisania nastąpić może tylko po rozkazach sterujących, przy spełnieniu określonych warunków i w sposób właściwy takiemu rozkazowi.

Ostatnim wypisanym zdaniem programu jest zawsze deklaracja KONIEC, sygnalizująca koniec wprowadzania programu do maszyny. Ostatnim wykonanym rozkazem jest rozkaz STOP, sygnalizujący zakończenie wykonywania programu i zatrzymanie się maszyny.

Wszystkie programy SAKO dzielimy na rozdziały, stanowiące odcinki programu, które wraz z przyporządkowanymi im danymi mieszczą się jednocześnie w pamięci wewnętrznej maszyny.
Składnia zdań SAKO jest następująca.

Zdania SAKO mogą być zbudowane z następujących znaków, dostępnych na dalekopisie /tablica 1/:

1. Alfabet łaciński, składający się z 26 liter

A, B, C, D, E, F, G •••, Z

2. Cyfry

0 1 2 3 4 5 6 7 8 9

3. Znaki działań i relacji

+ - x / * = =>

4. Separatory i nawiasy

. , : ()

5. Spacje /odstępy między znakami/.

Pewne określone ciągi znaków /nie licząc spacji/ nazywać będziemy zwrotami języka SAKO. Przykładami zwrotów są:

CZYTAJ
NASTEPNY
GDY

Pewne ciągi znaków o określonej budowie nazywać będziemy wyrażeniami. Przykładami wyrażeń są:

liczby
zmienne
funkcje
listy zmiennych
wyrażenia arytmetyczne
wyrażenia bulowskie.

Pewne ciągi znaków o dowolnej budowle lecz jednoznacznie ograniczone, nazywać będziemy tekstami:

X1 = **PIERWIASTEK=**

Zdania SAKO zbudowane są jako określone ciągi zwrotów, wyrażeń i tekstów. Rodzaj zwrotów, rodzaj wyrażeń, ograniczenia tekstów i kolejność ich występowania muszą być zgodne z jednym z dopuszczalnych typów zdań SAKO.

Każde wyrażenie SAKO rozpoczyna się od nowego wiersza. Ilość wierszy zajętych przez zdanie wynika wyłącznie z jego budowy. Na ogół zdania SAKO zamykają się w jednym wierszu.

Metodyka przygotowania programu

Dla uzyskania pełnej przejrzystości programu zalecane jest, aby każdy program, przygotowany do pracy na maszynę, był zaopatrzony w podane poniżej pozycje:

1. pełne sformułowanie problemu,
2. metoda rozwiązania uwzględniająca specyfikę pracy maszyn cyfrowych,
3. dyskusja skali występujących w problemie danych, wartości pośrednich i wyników,
4. dyskusja dokładności uzyskanych wyników,
5. postać danych, wprowadzanych do maszyny,
6. postać, w jakiej chcemy uzyskać wyniki,
7. sieć działań,
8. objaśnienia sieci działań,
9. właściwy program w języku SAKO,
10. objaśnienia programu,
11. metoda sprawdzenia poprawności programu,
12. metoda sprawdzenia poprawności wyników,
13. sprawdzenie ilości miejsc pamięci zajętych przez program i jego poszczególne rozdziały.
14. obliczenie czasu wykonywania programu w różnych wariantach jego wykonania.

Po sprawdzeniu programu, jego wykonaniu na maszynie i sprawdzeniu wyników następuje jeszcze

15. ostateczne opracowanie dokumentacji rozwiązania problemu.

Jak widać z powyższego, napisanie właściwego programu stanowi tylko jedną z wielu pozycji, jakie powinien opracować programista.

FORMA I OPIS POJĘĆ PODSTAWOWYCH W JĘZYKU SAKO

Poniżej podany jest opis i forma tych wyrażeń, które będą wykorzystane w opisie rozkazów SAKO /rozdz. 3/ i które posiadają jednakowe znaczenie, niezależnie od rozkazu, w którym będą one używane.

Liczby

Liczba ułamkowa

Przykłady:

```
       45.678
       .007
    -2345.
       +2.7182818284
123456789.1
```

Znaczenie:

Ogólnie przyjęte w matematyce. Kropka, zwana kropką pozycyjną, rozdziela część całkowitą i ułamkową liczby.

Forma:

Ciąg, złożony z co najwyżej dziesięciu cyfr dziesiętnych oraz kropki, która może być umieszczona na początku, na końcu lub też pośrodku liczby.

Nie są liczbami ułamkowymi w sensie SAKO:

3.1415926536	jedenaście cyfr dziesiętnych
-34,56	znak ,zamiast .

Reguły:

1. Każdej liczbie ułamkowej w sensie SAKO odpowiada pewna skala, dziesiętna lub binarna, określająca sposób jej zapisania w komórce pamięci maszyny. Ogólnie biorąc, zapis w skali dziesiętnej N składać się może z N cyfr przed kropką pozycyjną oraz z 10-N cyfr po kropce. Ściśle biorąc, wartość bezwzględna liczby ułamkowej, zapisanej w skali dziesiętnej N nie może osiągać lub przekraczać liczby a_N; tabelkę a_N w zależności od N podajemy poniżej:

N	a_N
0	1
1	16
2	128
3	1024
4	16384
5	131072
6	1048576
7	16777216
8	134217728
9	1073741824
10	34359738368

Skalę dziesiętną dla liczb ułamkowych, występujących explicite w programie, podaje deklaracja **SKALA DZIESIĘTNA PARAMETRÓW.**
W przypadku przekroczenia zakresu a_N odpowiadającego zadeklarowanej skali, następuje błędne wprowadzenie liczby do pamięci maszyny.

2. Każdą liczbę całkowitą, nie więcej jak dziesięciocyfrową, można zapisać jako ułamkową w sensie SAKO. Musi być ona wpisana do pamięci maszyny w skali dziesiętnej, zgodnej z wyżej podaną tabelką.

3. Wyniki działań arytmetycznych, dokonywanych przez maszynę, są wpisywane do pamięci maszyny w określonej skali. Skala ta, dziesiętna lub binarna, ustalona jest przez rozkaz **USTAW SKALE DZIESIETNIE** /lub **USTAW SKALE BINARNIE**/. Obliczanie wartości wyrażeń arytmetycznych oraz wczytywanie liczb rozkazem **CZYTAJ** przed ustawieniem skali prowadzi z zasady do błędnych wyników.

4. Gdy w wyniku działania otrzymamy liczbę ułamkową, o większej ilości cyfr znaczących przed kropką niż na to pozwala przyjęta skala, powstaje tzw. nadmiar połączony z błędnym zapisem liczby w odpowiedniej komórce pamięci. Powstanie nadmiaru jest w maszynie sygnalizowane przez automatyczne wpisanie liczby 1 do specjalnego rejestru, zwanego wskaźnikiem nadmiaru. Stan wskaźnika nadmiaru może być wykryty w maszynie przez rozkaz sterujący **GDY NADMIAR**. Wykonanie tego rozkazu powoduje m.in. wyzerowanie wskaźnika nadmiaru.

Liczba całkowita

Przykłady:

```
 321
 -7
45678
+15
-7654
```

Znaczenie:

Ogólnie przyjęte w matematyce.

Forma:

Ciąg co najwyżej pięciu cyfr dziesiętnych. Liczby ujemne są poprzedzone znakiem - (minus). Liczby dodatnie mogą /ale nie muszą/ być poprzedzone znakiem + (plus).

Nie są liczbami całkowitymi w sensie SAKO:

```
987654 zapis sprzeczny z formą: 6 cyfr
345.   zapis sprzeczny z formą: obecność w ciągu
                                kropki pozycyjnej.
```

Reguły:

1. Wynikiem działań arytmetycznych /z wyjątkiem dzielenia/ na liczbach całkowitych jest zawsze liczba całkowita.

2. Jeśli w wyniku pewnych działań powstanie w maszynie liczba całkowita niemieszcząca się w komórce pamięci /jest nią każda liczba, większa od 131071/, powstaje nadmiar z wszystkimi jego konsekwencjami.

Słowo bulowskie

Przykłady:

123.456.701.234
777000174000
012345.670123
137.237
7773.41
566331

Znaczenie:

Przez słowo bulowskie rozumiemy ciąg 36 lub 18 cyfr 0, lub 1, które traktujemy jako elementy dwuelementowej algebry Boole'a. Dla zapisu tych słów przyjmujemy formę oktalową. Słowo bulowskie długie zawiera 36 zer lub jedynek, słowo bulowskie krótkie zawiera 18 zer lub jedynek.

Forma:

Słowo bulowskie długie:
Ciąg dwunastu cyfr od 0 do 7, przedzielonych kropkami.

Słowo bulowskie krótkie:
Ciąg sześciu cyfr od 0 do 7 przedzielonych kropkami. Kropka nie może stać na początku ani na końcu słowa.

Blok liczbowy

Znaczenie:

Przez blok rozumiemy skończoną i uporządkowaną grupę liczb, oznaczoną wspólnym symbolem. Liczby, wchodzące w skład bloku noszą nazwę elementów bloku. Każdemu elementowi bloku odpowiada układ indeksów, wyznaczających jego położenie w bloku. Ilość indeksów nazywa się wymiarem bloku, ilość kolejnych /począwszy od zera/ liczb naturalnych, przebieganych przez pewien indeks, nosi nazwę zakresu tego indeksu.

Przykłady:

1. Ponumerowana grupa 40 liczb:

$$a_0, a_1, a_2, \ldots, a_{39}$$

stanowi przykład jednowymiarowego bloku zwanego wektorem. Każdy element wektora jest opatrzony jednym indeksem, którego zakres jest tutaj równy 40.

2. Układ liczb, tworzący tablicę o Trzech wierszach i czterech kolumnach

$$\begin{matrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \end{matrix}$$

stanowi dwuwymiarowy blok, zwany macierzą 3×4. Każdy element macierzy jest opatrzony dwoma indeksami, których zakresy w tym przypadku wynoszą 3 i 4.

W języku SAKO jest możliwe posługiwanie się blokami o dowolnej liczbie wymiarów. Każdy blok w programie musi być określony deklaracją BLOK, **STRUKTURA, TABLICA** lub **TABLICA OKTALNA,** które określają jego wymiar i zakresy indeksów.

Rozmieszczenie bloków w pamięci maszyny.

Każdy blok jest wprowadzany lub wczytywany do pamięci maszyny "wierszami" w kolejności, którą można określić, jak następuje: niech blok A będzie blokiem n-wymiarowym i

$$A_{i1},\ i_2,\ \ldots,\ i_n$$

pewnym jego elementem. Niech ponadto zakresami indeksów

$$i_1,\ i_2,\ \ldots,\ i_n$$

będą odpowiednio liczby

$$d_1,\ d_2,\ \ldots,\ d_n.$$

Wówczas element ten będzie wprowadzony lub wczytany do pamięci maszyny jako k-ty z kolei, licząc od zera, gdzie

$$k = \left(\ldots\left(\left(i_1 \cdot d_2 + i_2\right) \cdot d_3 + i_3\right) \cdot d_4 + \ldots + i_{n-1}\right) \cdot d_n + i_n.$$

Na przykład, w przypadku czterowymiarowego bloku o zakresach indeksów 3, 2, 4, 5, element np. wyznaczony przez indeksy

$$1,\ 1,\ 2,\ 4$$

będzie wczytany do pamięci maszyny jako

$$((1 \cdot 2 + 1) \cdot 4 + 2) \cdot 5 + 4 = 74$$

z kolei, licząc od zera.

Znajomość kolejności wczytywania elementów bloków do pamięci maszyny jest konieczna przy przygotowywaniu danych wejściowych dla maszyny.

Ogólne uwagi o blokach

1. Blok może być utworzony albo wyłącznie z liczb całkowitych, albo wyłącznie z liczb ułamkowych, albo słów bulowskich krótkich, albo słów bulowskich długich.

2. W języku SAKO rozpatrujemy tylko bloki tzw. „prostokątne”, to znaczy takie, że każdy indeks przebiega ustalony zbiór wartości niezależnie od wartości innych indeksów.

3. Nazwa bloku jest zbudowana zgodnie z zasadami budowy zmiennych prostych.

Zmienne

Zmienna prosta

Przykłady:

A
A12
EPSILON
R MALE
X3
SUMA KWADRATÓW

Znaczenie:

Jest to symbol, przyjmujący w trakcie obliczeń różne wartości liczbowe.

Te same zmienne, użyte w różnych rozdziałach, posiadają całkowicie niezależne znaczenia. W tym samym rozdziale, zmienne, użyte w programie głównym i poszczególnych podprogramach są również od siebie niezależne.

Uwaga:
Każda zmienna prosta w danym programie ma przypisaną sobie komórkę pamięci maszyny, w którą są wpisywane kolejne wartości tej zmiennej.

Forma:

Ciąg liter, cyfr i spacji, rozpoczynający się od litery. Dwie zmienne języka SAKO są traktowane jako identyczne, jeśli ich pierwsze cztery znaki, nie licząc spacji, są identyczne.

Np. zmienne:

WARIACJA
WARIANCJA

są w języku SAKO identyczne.

Rozróżnialne są natomiast zmienne:

R MALE
R MACIERZY

gdyż różnią się one czwartym z kolei znakiem: L i C.

Tak więc własność identyfikującą posiadają cztery pierwsze znaki.

Nie są w SAKO zmiennymi:

A.B3 \ w ciągu występują nie tylko
C/8 / litery i cyfry
8DF \ ciąg rozpoczyna
3H8 / się od cyfry

Zmienna indeksowana

Przykłady:

A(3)
BETA(0,8)
C8(2,BETA)
TLX(l,0,1,3,21)

Znaczenie:

Zmienna indeksowana jest to symbol elementu bloku, wyznaczony wartościami indeksów. Własności zmiennej indeksowanej nie różnią się od własności zmiennych prostych.

Forma:

Symbol zmiennej indeksowanej jest identyczny z nazwą bloku, którego element zmienna ta określa. Po symbolu tym występuje para nawiasów, wewnątrz której znajdują się indeksy rozdzielone przecinkami. Własność identyfikującą posiadają cztery pierwsze znaki symbolu zmiennej /nie licząc spacji/.

Reguły:

1. Posługiwanie się zmiennymi indeksowanymi w programie wymaga uprzedniej deklaracji BLOK lub STRUKTURA, odnoszącej się do właściwych bloków.
2. Indeksy przyjmować mogą jedynie wartości całkowite. Indeksami zmiennej indeksowanej mogą więc być tylko:
   - liczby całkowite nieujemne,
   - zmienne proste lub indeksowane o wartościach całkowitych nieujemnych,
   - wyrażenia arytmetyczne o wartościach całkowitych nieujemnych /patrz wyrażenie arytmetyczne poniżej/.

Numery

Przykłady:

1
1 WYJŚCIE
2BC
789
3XG5

Znaczenie:

Jest to symbol, służący do oznaczania rozkazów SAKO. Numer wypisuje się z lewej strony rozkazu, oddzielając go od rozkazu nawiasem zamykającym. O rozkazie, przy którym został napisany numer α mówimy, że jest on opatrzony numerem α lub że jest to rozkaz o numerze α. Nie jest konieczne, aby wszystkie rozkazy programu były opatrzone numerami. Dwa rozkazy mogą być opatrzone tymi samymi numerami jedynie wówczas, gdy rozkazy te nie występują wewnątrz tego samego paragrafu lub rozdziału.

Forma:

Ciąg liter, cyfr i spacji, rozpoczynający się od cyfry. Własność identyfikującą posiadają cztery pierwsze znaki, nie licząc spacji.

Funkcje

Przykłady:

SIN (X + A)
TRY (ALFA)
TNG (X)

Znaczenie:

Znaczenie ogólnie przyjęte w matematyce. W języku SAKO dopuszczalne są dwa typy funkcji

a. Funkcje języka
b. Funkcje definiowane

W SAKO wyróżniamy następujące funkcje języka:

Tablica 2

Skrót	Nazwa	Znaczenie
SIN(X)	Sinus	
COS(X)	Cosinus	
TNG(X)	Tangens	
ASN(X)	Arcus sinus	
ACS(X)	Arcus cosinus	
ATG(X)	Arcus tangens	
ARC(X,Y)	Arcus	Oznacza kąt, leżący w ćwiartce kątowej wyznaczonej przez znak X i znak Y, którego tangens jest równy Y/X.
PWK(X)	Pierwiastek kwadratowy	
PWS(X)	Pierwiastek sześcienny	
LN(X)	Logarytm naturalny	
EXP(X)	Eksponent	
MAX(X,Y, ...,Z)	Maksimum	Oznacza największą z liczb X,Y,.,Z
MIN(X,Y, ...,Z)	Minumum	Oznacza najmniejszą z liczb X,Y,.Z
MOD(X,Y)	Moduł	Oznacza resztę powstałą z dzielenia X przez Y /tylko dla całkowitych A i B/
SGN(X,Y)	Signum	Oznacza liczbę |X|· sgn Y
ABS(X)	Wartość bezwzględna /absolutna/	Oznacza wartość absolutną X
CNT(X)	Część całkowita	Oznacza część całkowitą liczby X

Inne funkcje jako funkcje definiowane wprowadza się rozkazem **PODPROGRAM.**

Forma:

Funkcje języka zapisuje się symbolem podanym obok funkcji. Argumenty podaje się w nawiasach, oddzielając je przecinkami.

Nazwy funkcji definiowanych są tworzone zgodnie z zasadami przyjętymi dla zmiennych prostych, z tym że własność identyfikującą posiadają tutaj trzy pierwsze litery nie licząc spacji. Argumenty funkcji definiowanych podaje się w sposób identyczny jak dla funkcji języka SAKO.

Uwagi:
1. Trzy pierwsze litery nazwy funkcji definiowanej nie mogą pokrywać się z trzema pierwszymi literami funkcji języka.
2. Sześć ostatnich funkcji, podanych w tablicy 2 można używać jedynie w wyrażeniach arytmetycznych; na przykład:

A = CALKA (A,B,ABS( ),EPSILON) źle
A = CALKA (A,B+ABS(c),SIN( ),0.01) dobrze

Wyrażenia arytmetyczne

Przykłady:

A+1
A+ALFA/X15+.7321
GAMMA-D3(A+B,9.8l)
LOG(SIN(X+Y)/PWK(Z)
A(3,B(4,J),ENT(C+D))+S
SIGMA/B + SIGMA*.33+1

Znaczenie:

Wyrażenie arytmetyczne posiada sens ogólnie przyjęty w matematyce. Znaki + - x / posiadają konwencjonalne znaczenia dodawania, odejmowania, mnożenia i dzielenia. Znak * jest symbolem potęgowania:

A*B oznacza A^B

Nawiasy określają - jak zazwyczaj - kolejność wykonywania działań, zamykają argumenty funkcji lub też zamykają indeksy zmiennych indeksowanych.

Każde wyrażenie arytmetyczne ma jednoznacznie określoną wartość, powstałą w wyniku wykonywania działań, wymienionych w wyrażeniu. Wartością wyrażenia może być liczba całkowita lub ułamkowa.

Forma:

Sposób budowy wyrażenia arytmetycznego określamy rekurencyjnie w sposób następujący:

1. Zmienna bądź liczba jest wyrażeniem.
2. Jeśli A jest wyrażeniem, wówczas (A) jest również wyrażeniem.
3. Jeśli A jest wyrażeniem nierozpoczynającym się od znaku + lub - , wówczas -A jest również wyrażeniem.
4. Jeśli A, B są wyrażeniami, B nie rozpoczyna się od znaku + lub - , wówczas

A + B
A - B
A x B
A / B
A * B

są również wyrażeniami.

5. Jeśli G jest nazwą bloku, A, B, …,Z są wyrażeniami przyjmującymi wartości całkowite i ilość tych wyrażeń jest równa ilości indeksów danej zmiennej indeksowanej G, wówczas

G(A , B , ..., Z)

jest również wyrażeniem.

6. Jeśli f jest nazwą funkcji, A, B, ..., Z są wyrażeniami w ilości i rodzaju zgodnym z argumentami tej funkcji, wówczas

f(A, B, .., Z)

jest również wyrażeniem.

Powyższa definicja formy wyrażenia pozwala budować wyrażenia drogą składania ich z wyrażeń mniej skomplikowanych. bądź też pozwala sprawdzać poprawność budowy konkretnych wyrażeń arytmetycznych.
Nie są wyrażeniami:

A + / B
A + -4.5678
A* *C
(A + b)c

Kolejność wykonywania działań w wyrażeniu arytmetycznym:

W SAKO przyjmujemy następującą listę mocy działań:

- Obliczanie wartości funkcji i wartości zmiennych indeksowanych.
- Wykonywanie działań * /potęgowanie/.
- Wykonywanie działań x.
- Wykonywanie działań /.
- Wykonywanie działań -.
- Wykonywanie działań +.

Działaniem o większej mocy nazywać będziemy działanie, stojące na podanej liście pod mniejszym numerem.

Mając określoną moc działań, możemy teraz podać ogólne reguły kolejności ich wykonywania.

Z dwu działań wykonuje się najpierw to,które jest zamknięte w większej ilości nawiasów.

W przypadku równej ilości nawiasów wykonuje się najpierw działanie o większej mocy.

Działania o równej mocy zamknięte w jednakowej ilości nawiasów wykonuje się kolejno od lewej do prawej strony wyrażenia arytmetycznego.

Wyrażenia całkowite i ułamkowe

Każde wyrażenie arytmetyczne SAKO posiada wartość całkowitą lub ułamkową, innymi słowy - jest całkowite lub ułamkowe. Decydują o tym następujące reguły, które stosuje się przy obliczaniu wartości wyrażenia:

- Zmienne proste, indeksowane oraz funkcje mają wartości całkowite /lub krócej: są całkowite/ wtedy i tylko wtedy, gdy są wymienione w deklaracji **CALKOWITE.**
- Funkcje **ENT** i **SGN(A,B)**, gdy A jest całkowite, są z definicji funkcjami o wartościach całkowitych.
- Jeśli A jest całkowite, to (A) i -A są całkowite. Analogiczna reguła dotyczy wyrażeń ułamkowych.
- Wyrażenie A/B jest zawsze ułamkowe. Natomiast A + B, A-B, A x B, A * B są tylko wtedy całkowite, gdy zarówno A i B są całkowite.

Wynika stąd, że wyrażenie ma wartość całkowitą wtedy i tylko wtedy, gdy wszystkie występujące w nim:

- liczby ułamkowe
- zmienne ułamkowe
- znaki dzielenia
- funkcje ułamkowe

znajdują się w wyrażeniu pod znakami funkcji o wartościach całkowitych.

Skala obliczania wartości wyrażenia

Jeśli wartość wyrażenia jest liczbą ułamkową, to jest ona obliczana zgodnie z ostatnio wykonanym rozkazem USTAW SKALE, ustalającym skalę obliczanych wartości.

Wszystkie stałe lub zmienne ułamkowe, występujące w wyrażeniu, muszą być wprowadzone, wczytane lub obliczone w tej samej skali, zgodnej z ostatnio wykonanym rozkazem **USTAW SKALE DZIESIETNIE** (**BINARNIE**) bądź ostatnio napisaną deklaracją **SKALA DZIESIETNA** (**BINARNA**) **PARAMETROW.**

Uwagi:

1. Wszystkie zmienne, występujące w wyrażeniu muszą nieć uprzednio określone wartości przez wprowadzenie, wczytanie lub obliczenie.
2. Para dodatkowych nawiasów, niekoniecznych dla jednoznacznego odczytania wyrażenia, nie stanowi błędnego zapisu.
3. Znak x dla mnożenia jest istotny. Nie można go zastępować znakiem • lub pomijać.
4. Wszystkie wartości pośrednie, jak i końcowy wynik obliczeń, określonych w wyrażeniu, muszą mieścić się w zakresie aktualnie przyjętej skali. Przekroczenie zakresu skali w trakcie obliczania wartości wyrażenia arytmetycznego powoduje

wpisanie liczby 1 do rejestru **WSKAZNIK NADMIARU** i może być wykryte w programie za pośrednictwem rozkazu sterującego **GDY BYL NADMIAR.**

Wyrażenia bulowskie

Przykłady:

A + BETA
A + (-C) X D28
C * (-2) + KSI
G x 000.077.777.760
A x (-B) + (-A) x B

Znaczenie:

Wyrażeniem bulowskim jest ciąg znaków, określający rodzaj 1 kolejność wykonywania działań bulowskich na słowach, których symbole występują w wyrażeniu. Każde wyrażenie bulowskie ma jednoznacznie określoną wartość, będącą również słowem bulowskim.

Forma:

Poniżej podajemy rekurencyjne określenie formy wyrażenia bulowskiego, które pozwala tworzyć i sprawdzać poprawność budowy wyrażenia:

- Każda zmienna prosta lub słowo bulowskie jest wyrażeniem bulowskim.
- Jeśli A jest wyrażeniem bulowskim, nierozpoczynającym się od znaku - , wówczas -A jest również wyrażeniem bulowskim.
- Jeśli A jest wyrażeniem bulowskim, wówczas (A)jest również wyrażeniem bulowskim.
- Jeśli A i B są wyrażeniami bulowskimi, zaś B nie rozpoczyna się od znaku - , wówczas A + B , A x 3 są wyrażeniami bulowskimi.
- Jeśli A jest wyrażeniem, I - zmianą całkowitą lub całkowitą nieujemną, wówczas A*1 , A*(-1) są wyrażeniami bulowskimi.

Reguły:

1. Wartość wyrażenia bulowskiego oblicza się według następującej hierarchii działań:

wykonywanie przesunięć * ,
wykonywanie iloczynów bulowskich x ,
wykonywanie sum i negacji + -

Z dwu działań wykonuje się najpierw to, które j-est zamknięte w większej ilości nawiasów. W przypadku równej ilości nawiasów wykonuje się to, które jest wymienione w liście hierarchii we wcześniejszej kolejności. Działania, zamknięte w jednakową ilość nawiasów i stojące na liście hierarchii w tym samym punkcie, wykonuje się kolejno od lewej do prawej strony wyrażenia bulowskiego.

2. Każda liczba może być traktowana jako słowo bulowskie i odwrotnie. Wartością wyrażenia bulowskiego jest słowo bulowskie, które może być traktowane w dalszym ciągu programu jako liczba całkowita lub ułamkowa.

Lista

Przykłady:

```
A, B, C, D, E
1, 2, 3, 4, 5, 6, 7
AX4, BT, *ALFA, *GAMMA»
1 WYJŚCIE, 2CA, 81CTB, -
  3BX, 8
A(K+1) , A(K+2) , A(K+3) , A(K+4)
A, B, F(), GX8(), *BVN, ASD
```

Znaczenie:

Lista jest to ciąg zmiennych, liczb, numerów, nazw bloków lub podprogramów, oddzielonych przecinkami. Listę stosuje się w rozkazach lub deklaracjach, które mogą odnosić się do więcej niż jednej zmiennej, liczby bloku lub podprogramu.

Forma:

Symbole, występujące na liście, pisane są zgodnie z zasadami, dotyczącymi ich fermy. Nazwy bloków, występujących na liście, poprzedzone są symbolem * ; np.

```
*GAMMA
```

Nazwy podprogramów, występujących na liście, uzupełnione są parą nawiasów:

```
F()
```

Zasada ta nie obowiązuje w tych rozkazach lub deklaracjach, które odnoszą się z reguły do bloków lub podprogramów, tzn, gdy z góry wiadomo, jakiego charakteru symbole występują na liście.
Np. w deklaracji STRUKTURA, na której liście występują z reguły tylko nazwy bloków, nie uzupełnia się tych nazw gwiazdkami:

```
STRUKTURA: (N, M, 6) : A, 3, C
```

Listę można przenosić /i to wielokrotnie/ z jednego wiersza do drugiego. Stosuje się w tym celu znak - , który stawia się po przecinku, oddzielającym ostatni symbol w górnym wierszu od pierwszego symbolu w wierszu poniżej, np.:

```
WE, RTZ, Ul, 0?, ASDF, -
  GHJK, KL, YXC
```

FORMA I OPIS POSZCZEGÓLNYCH ROZKAZÓW SAKO

Deklaracje

ROZDZIAL : nazwa

Przykłady:

ROZDZIAL : GAUSS - SEIDEL
ROZDZIAL : LICZENIE PIERWIASTKÓW
ROZDZIAL : L2A

Znaczenie:

Część programu, jaką możemy w całości pomieścić w pamięci wewnętrznej maszyny, nazywamy rozdziałem.

Deklaracja ROZDZIAL określa wypisaną pod nią część programu aż do najbliższej deklaracji ROZDZIAL lub KONIEC jako rozdział i jednocześnie nadaje mu nazwę.

Forma:

nazwa - składa się z liter, cyfr i spacji, przy czym pierwszym znakiem jest litera. Własność identyfikująca posiada 10 pierwszych znaków /nie licząc spacji/.

Reguły:

1. Przed pierwszym wypisanym rozdziałem nie potrzeba dawać deklaracji ROZDZIAL, o ile nie jest potrzebny powrót do tego rozdziału /rozkaz sterujący IDZ DO ROZDZIALU/. W szczególności deklaracja ROZDZIAL jest zbędna w programach, mieszczących się w jednym rozdziale.

2. Wszystkie deklaracje tracą swe znaczenie przy przejściu z jednego rozdziału do drugiego. Wartości zmiennych w jednym rozdziale przestają być aktualne w obrębie drugiego rozdziału. To samo dotyczy numerów rozkazów oraz nazw podprogramów. Tak więc zmiennym, użytym w jednym rozdziale muszą być nadane wartości w tym samym rozdziale. Podobnie podprogramy, używane w pewnym rozdziale muszą być w nim określone. Ostatni wypisany rozdział musi być zakończony deklaracją KONIEC, która sygnalizuje koniec wprowadzania programu.

3. Wykonywanie programu rozpoczyna się od pierwszego wypisanego rozdziału. Kolejność przechodzenia do dalszych rozdziałów określona jest rozkazami IDZ DO ROZDZIALU, a więc nie musi następować w kolejności ich wypisania.

Uwaga:

Wszystkie rozdziały, przełożone na język maszyny, zapisane są w pamięci bębnowej. Przy przejściu do nowego rozdziału rozdział ten zostaje przepisany z pamięci bębnowej do wewnętrznej na miejsce poprzednio wykonywanego rozdziału. Kosztuje to pewną ilość czasu, dlatego też należy w miarę możności unikać przechodzenia z jednego rozdziału do drugiego.

PODPROGRAM : lista wyników = nazwa (lista argumentów)

Przykłady:

```
PODPROGRAM: (U, V, W) = TRY(X, Y, Z)
PODPROGRAM: (*A) = MN MACIERZY (*B, *C, N, M, L)
PODPROGRAM: CALKA  (A, B, F( ), H)
```

Znaczenie:

Deklaracja ta określa ciąg następujących po niej rozkazów aż do najbliższej deklaracji **PODPROGRAM**, R**OZDZIAL** lub **KONIEC** jako podprogram o danej nazwie, argumentach i wynikach.

Forma:

Nazwa podprogramu jest budowana zgodnie z zasadami tworzenia zmiennych prostych, z tym że własność identyfikującą posiadają tutaj trzy pierwsze znaki, nie licząc spacji. Lista wyników jest zamknięta w nawiasy i stoi po lewej stronie znaku = . Lista argumentów, również w nawiasach, znajduje się po prawej stronie znaku = .

Jako argumenty występować mogą symbole:

liczb,
bloków,
podprogramów,
funkcji.

Jako wyniki występować mogą symbole:

liczb,
bloków.

Aby odróżnić nazwy bloków od zmiennych, na liście przed nazwami bloków stawiamy gwiazdkę. Aby odróżnić zmienne od nazw podprogramów i funkcji, po ich nazwach stawiamy parę nawiasów, np.

```
TRY()
```

W przypadku, gdy podprogram służy do określenia funkcji F, w deklaracji **PODPROGRAM** nie występuje lista wyników ani znak = /patrz ostatni z podanych przykładów/. W tym przypadku ostatni wykonywany rozkaz arytmetyczny podprogramu musi mieć postać

```
F() = . . . .
```

gdzie prawa strona formuły powoduje obliczenie ostatecznej wartości funkcji, lewa strona formuły dla przykładu trzeciego ma postać

```
CALKA()= . . .
```

Reguła:

Kolejność występowania na listach argumentów i wyników jest ustalona i nie może ulegać zmianom.

Uwagi:

1. Miejsce w pamięci maszyny na bloki, będące argumentami lub wynikami podprogramu, rezerwuje program główny. W podprogramie podaje się tylko strukturę tych bloków.

2. Funkcjami, które są argumentami podprogramu, mogą być zarówno funkcje języka /z wyjątkiem MAX, MIN, MOD, AB3, SGN, ENT/ jak i funkcje określone przez podprogramy.

BLOK (n, m, …,k) : lista

Przykłady:

```
BLOK (15) : A, B
BLOK (3, 8) : ALFA, G2
BLOK (2, 2, 2, 5, 3) : CX2
```

Znaczenie:

Deklaracja ta stwierdza, że symbole, wymienione w liście są nazwami bloków o wspólnej strukturze, określonej przez parametry deklaracji.

Forma:

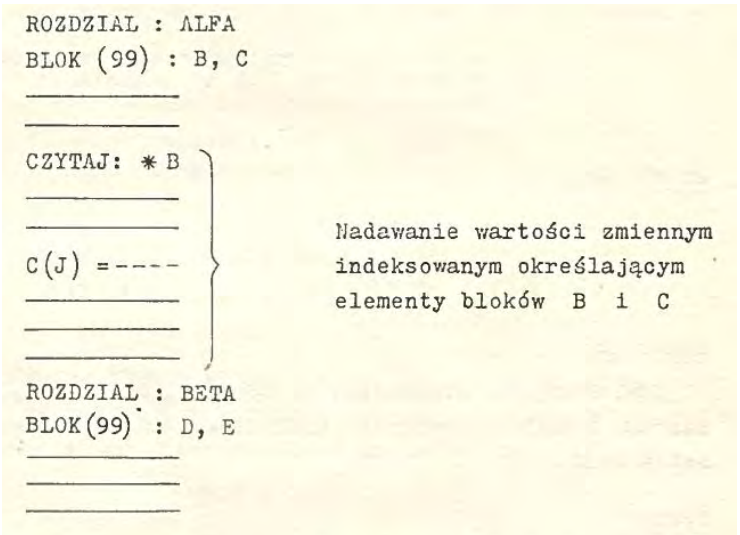
n, m, ..., k - liczby naturalne
lista - jest utworzona z nazw bloków .

Uwagi:

1. Deklaracja BLOK powoduje zarezerwowanie w pamięci wewnętrznej maszyny odpowiedniej ilości miejsc na pomieszczenie elementów bloków, wymienionych w liście.
2. Użycie każdej zmiennej indeksowanej i każdej deklaracji STRUKTURA musi być poprzedzone w programie właściwą deklaracją BLOK.
3. Jeżeli dwa różne rozdziały rozpoczynają się od deklaracji BLOK, określających analogiczne ciągi bloków o tej samej strukturze /chociaż nawet o różnych nazwach/, to wartości elementów tych bloków określone w jednym rozdziale, zachowują się przy przejściu do drugiego rozdziału.

Reguła ta staje się zrozumiała, gdy uwzględnimy, że deklaracje BLOK rezerwują miejsca pamięci na dane zawsze w ten sam sposób i zawartość tych miejsc nie ulega zmianie przy przejściu z jednego rozdziału do drugiego.

Wyjaśnia to następujący przykład:



W rozdziale ALFA deklaracja BLOK określa strukturę bloków B, C oraz rezerwuje 100 + 100 = 200 miejsc w pamięci maszyny na pomieszczenie liczb, wchodzących w skład tych bloków. Wykonanie rozkazów: CZYTAJ : B oraz formuł arytmetycznych

C(J)= . . .

dla J = 0, 1, ...,99 powoduje wypełnienie zarezerwowanych miejsc liczbami /czyli powoduje nadanie zmiennym B(I), C (I) określonych wartości/.

W rozdziale BETA deklaracja BLOK powoduje również zarezerwowanie 200 miejsc w pamięci, określając jednocześnie strukturę bloków D i E. Jeżeli teraz, po wykonaniu rozdziału ALFA rozkazem IDZ DO ROZDZIALU : BETA przejdziemy do tego rozdziału, to końcowe wartości bloków B i C rozdziału ALFA staną się początkowymi wartościami bloków D i E rozdziału BETA.

STRUKTURA (i, J, ..., K) : lista

Przykłady:

```
STRUKTURA (ALFA) : A, B
STRUKTURA (n, 5) : BETA: G3
STRUKTURA (K, L, m) : CX2, B, D
```

Znaczenie:
Deklaracja ta zmienia strukturę bloków, wymienionych w liście, a zadeklarowanych napisaną we wcześniejszej kolejności deklaracją BLOK. Nowa struktura bloków jest utworzona przez podanie liczb lub zmiennych, określających wymiar bloków i zakresy ich indeksów.

Forma:
I, J, ..., K - zmienne lub liczby całkowite, określające wspólne zakresy indeksów wymienionych na liście bloków.
lista - jest utworzona z nazw bloków.

Uwagi:
1. Rezerwowanie miejsc w pamięci na pomieszczenie elementów bloków odbywa się jedynie za pośrednictwem deklaracji BLOK; deklaracja STRUKTURA nie posiada tej właściwości.
2. Ilość elementów bloku w zmienionej strukturze noże być co najwyżej równa ilość miejsc pamięci, zarezerwowanych przez właściwą deklarację BLOK.

TABLICA (n, m,..., k) : nazwa

Przykłady:

```
TABLICA (2, 3) : A
3.7182    45.13     .1508
-.35      9.83      32.01
          *
```

```
TABLICA (7) : DZETA
21 4 694  37  0          293     23
                *
```

Znaczenie:

Deklaracja ta określa nazwę i strukturę bloku, którego elementy są wypisane począwszy od następnego wiersza formularza. Blok ten wprowadzany jest do maszyny równocześnie z programem i znajduje się w tej części pamięci, w której mieści się program obliczeń.

Forma:

Struktura wymienionego bloku określona jest przez podanie po słowie TABLICA zakresów indeksów w ilości i kolejności odpowiadającej indeksowaniu elementów tego bloku. Zakresy te są zamknięte w nawiasy i przedzielone przecinkami.

Poczynając od następnego po deklaracji TABLICA wiersza formularza wypisuje się listę liczb według zasad przygotowywania danych wejściowych /patrz rozkaz CZYTAJ/.

Zakresy indeksów:

n, m, ..., k

są w deklaracji TABLICA podane w postaci nieujemnych liczb całkowitych.

Nazwa w deklaracji TABLICA jest napisana zgodnie z przyjętymi zasadami pisania nazw bloków.

Uwagi:

1. Deklaracja TABLICA wraz z następującym po niej układem liczb powoduje nadanie liczbowych wartości elementom wymienionego bloku w trakcie wprowadzania programu. Jest to istotna różnica w stosunku do deklaracji BLOK i STRUKTURA. Blokom, określonym tymi deklaracjami, wartości nadawane są dopiero w trakcie wykonywania programu.
2. Gdy tablica składa się z liczb ułamkowych, skala, w jakiej mają być wprowadzone jej elementy, musi być ustalona za pośrednictwem poprzednio wypisanej deklaracji, dotyczącej skali parametrów.
3. W trakcie wykonywania programu można zmieniać pierwotne wartości elementów tablicy.

TABLICA OKTALNA (n, m, ..., k) : nazwa

Przykłady:

```
TABLICA OKTALNA(5) : ALFA

074.562.321.777
7777.0000.7777
252525.252525
400.000.000.000
333.333.333.333
000.760.003770
     *
```

```
TABLICA OKTALNA(1, 2) : KAPPA

 123456.710111
 34.72.27.61.23.41
 121126.123365
 7321.0012.375
 111 .000.111 .000
 721641002831
      *
```

Znaczenie:
Deklaracja ta jest identyczna z deklaracją TABLICA z tym, że tablica oktalna utworzona jest ze słów bulowskich, zapisanych oktalnie.

Forma:
Analogiczna, jak w przypadku TABLICY z tym że zamiast liczb występują słowa bulowskie.

Uwaga:
Patrz 1 i 2 uwaga dla deklaracji TABLICA.

CALKOWITE : lista

Przykłady:

```
CALKOWITE: A, GAMMA, X3
CALKOWITE: L, L1, *K2, TRY()
```

Znaczenie:
Deklaracja ta stwierdza, że wartości zmiennych prostych, elementów bloków lub wyników podprogramów, których nazwy występują na liście, są liczbami całkowitymi.

Forma:
Lista deklaracji CALKOWITE jest listą zmiennych, nazw bloków i nazw podprogramów. Słowo CALKOWITE oddzielone Jest od pierwszego symbolu na liście dwukropkiem.

Reguły:

- Zmienne, elementy bloków, wyniki podprogramów, których nazwy nie są wymienione w deklaracji CALKOWITE, przyjmują wartości ułamkowe. Tak więc deklaracja CALKOWITE dokonuje podziału zmiennych na całkowite i ułamkowe.
- Deklaracja CALKOWITE traci znaczenie przy przejściu z jednego rozdziału do drugiego, a w obrębie jednego rozdziału, przy przejściu z programu głównego do podprogramu, lub z jednego podprogramu do drugiego.

SKALA DZIESIĘTNA PARAMETRÓW n

Przykłady:

```
SKALA DZIESIETNA PARAMETRÓW 0
SKALA DZIESIETNA PARAMETRÓW 3
```

Znaczenie:

Deklaracja ta ustala skalę dziesiętną, w jakiej będą wprowadzane /wraz z programem/ wszystkie liczby ułamkowe /tak zwane parametry liczbowe/ w rozkazach o dalszej kolejności wypisania - aż do nowej deklaracji, dotyczącej skali parametrów.

Forma:
n - liczba naturalna, zawarta w przedziale 0, 1oJ.

Reguła:
W każdym programie pojawienie się liczb ułamkowych musi być poprzedzone deklaracją dotyczącą skali parametrów.

Uwaga:
Parametry liczbowe w programie występują jedynie w:

- wyrażeniach arytmetycznych,
- tablicach,
- formułach arytmetycznych i operacyjnych.

SKALA BINARNA PARAMETRÓW n

Przykłady:

```
SKALA BINARNA PARAMETRÓW 0
SKALA BINARNA PARAMETROW 33
```

Znaczenie:
Deklaracja ta ustala skalę binarną liczb ułamkowych, wprowadzanych w dalszej kolejności - aż do nowej deklaracji, dotyczącej skali parametrów.

Forma:
n - liczba naturalna zawarta w przedziale [0,35].

Reguła:
Patrz - Reguła dla deklaracji SKALA DZIESIETNA PARAMETROW.

JEZYK SAS

Znaczenie:

Deklaracja ta stwierdza, że następujące po niej rozkazy są napisane w języku SAS. Powrót do języka SAKO wymaga deklaracji JEZYK SAKO.

Uwaga:

W programie napisanym w języku SAS nie można użyć odpowiedników deklaracji ROZDZIAL, KONIEC oraz rozkazów IDZ DO ROZDZIALU.

JEZYK SAKO

Znaczenie:

Deklaracja ta stwierdza, że następujące po niej rozkazy są napisane w języku SAKO. Deklaracji tej używa się po uprzednim zastosowaniu deklaracji JEZYK SAS.

KONIEC

Znaczenie:

Deklaracja ta określa koniec napisanego programu.

Uwagi:

Po wprowadzeniu deklaracji KONIEC maszyna kończy wprowadzanie programu i w zależności od położenia kluczy na pulpicie sterującym maszyny:

- zatrzymuje się i po naciśnięciu przycisku START przechodzi do wykonania programu,
- wyprowadza program w języku SAS,
- wyprowadza program w języku SAB,
- wprowadza program w systemie oktalnym.

Rozkazy sterujące

SKOCZ DO α

Przykłady:

```
SKOCZ DO 3
SKOCZ DO 5F
```

Znaczenie:

Rozkaz powoduje przejście do rozkazu o numerze α .

Forma:

α - numer rozkazu.

Uwaga:

Zamiast numeru α w rozkazie SKOCZ DO α można napisać słowo NASTEPNY; rozkaz SKOCZ powoduje wówczas przejście do rozkazu wypisanego w najbliższej kolejności /staje się rozkazem nieefektywnym/.

SKOCZ WEDLUG I : α, ß, ...,ω

Przykłady:

```
SKOCZ WEDŁUG ALFA : 1, 2A, NASTĘPNY
SKOCZ WEDŁUG K2 : 3AB, 4, 2, 5C
```

Znaczenie:

Rozkaz określa przejście do rozkazu o numerze, który występuje na liście w kolejności I. Jeśli zmienna I posiada wartość 0, rozkaz SKOCZ WEDŁUG I spowoduje przejście do rozkazu o numerze pierwszym na liście; gdy wartością I jest 2, wówczas jest wykonane przejście do rozkazu o numerze trzecim na liście itd.

Forma:

I - zmienna całkowita

α, ß, ...,ω - numery rozkazów.

Każdy z tych numerów może być zastąpiony słowem NASTĘPNY.

Uwaga:

Przed wykonaniem rozkazu SKOCZ WEDŁUG I wartość zmiennej I powinna
być przez program określona.

POWTORZ OD α : I = J(K)L

Przykłady:

```
POWTORZ OD 3 : ALFA = 0(1)29
POWTORZ OD 1ERF : C3 = 0.01(0.01)1.
POWTORZ OD 2 : AB1 = X0(DELTA)5
```

Znaczenie:

Rozkaz **POWTORZ OD α : I = J(K)L** powoduje wielokrotne wykonanie odcinka programu, zawartego między rozkazem oznaczonym numerem α a samym rozkazem **POWTORZ.** Pierwszy raz dany fragment programu wykonuje się dla I = J, przy każdym następnym wykonaniu wartość I zwiększa się o K. Gdy zmienna I osiągnie wartość L, wówczas następuje przejście do następnego rozkazu po rozkazie **POWTORZ.**

Forma:

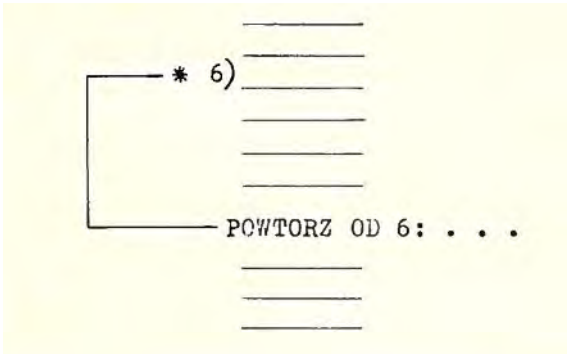
α - numer rozkazu

I, J, K, L - liczby lub zmienne proste.

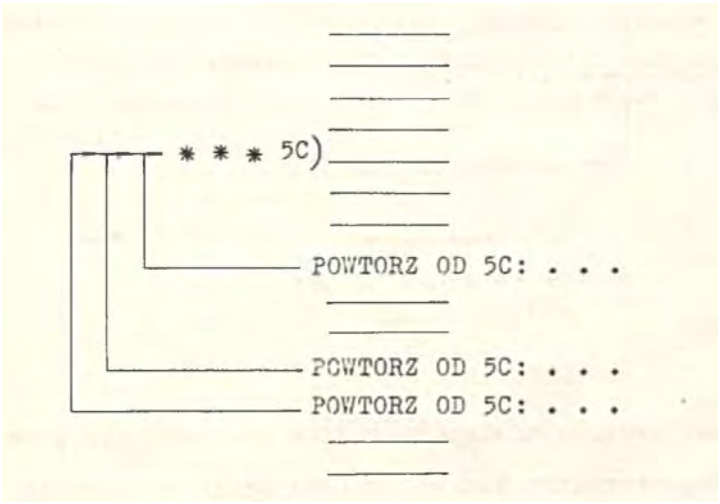
Ich wartości powinny posiadać jednakowy charakter, tzn. winny być jednocześnie całkowitymi lub ułamkowymi.

Reguły:

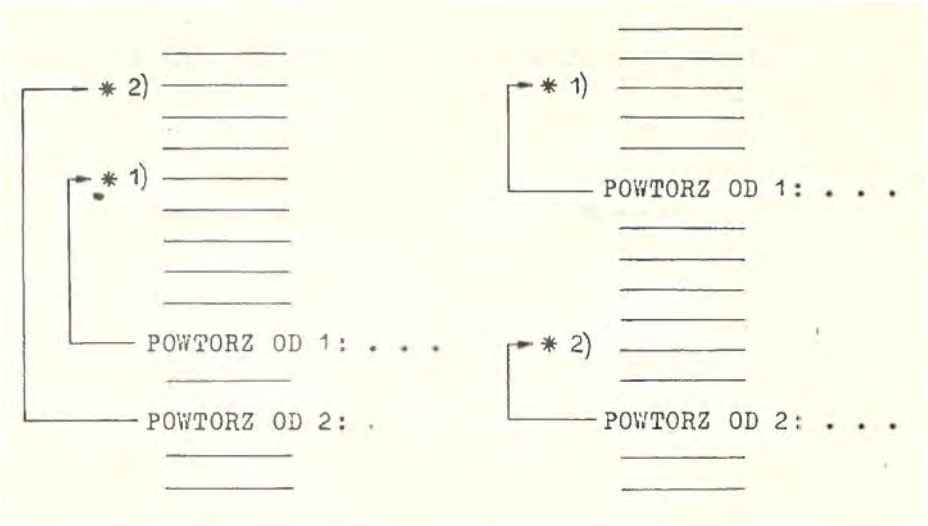
1. Stosując rozkaz **POWTORZ OD** α stawiamy znak * z lewej strony numeru α :



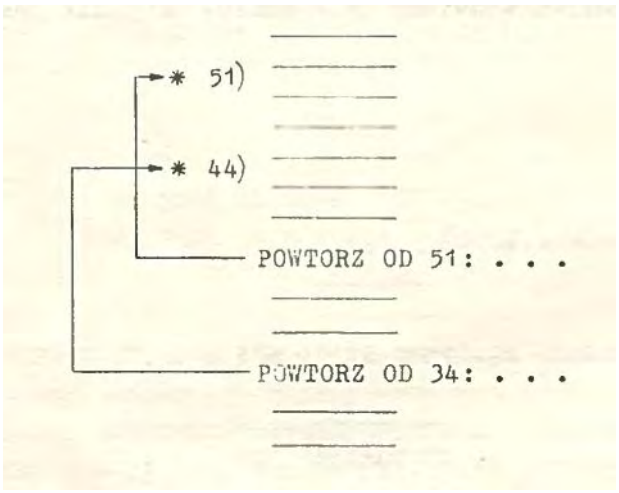
Jeżeli do jednego numeru α odnosi się kilka rozkazów **POWTORZ,** to przed tym numerem stawiamy tyle znaków * , ile jest tych rozkazów:



2. Rozkaz **POWTORZ** może się odnosić tylko do numeru rozkazu, wypisanego przed nim w programie.
3. Odcinek programu, zawarty między rozkazem o numerze α a rozkazem **POWTORZ OD** α nazywamy zasięgiem tego rozkazu **POWTORZ.** Dwa zasięgi mogą być albo całkowicie rozłączne /tzn. nie posiadać żadnego wspólnego rozkazu/, albo jeden z nich może zawierać się wewnątrz drugiego:



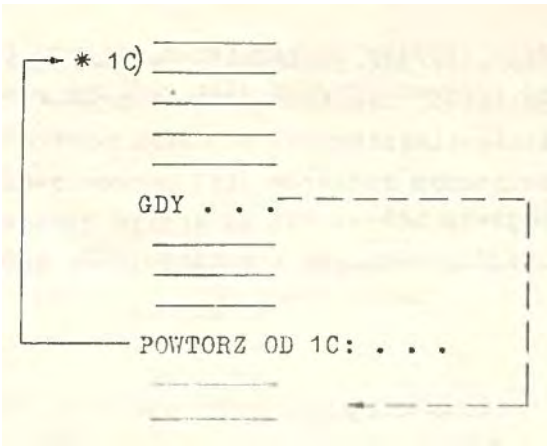
Nie jest prawidłowy następujący układ rozkazów POWTORZ:



Zasadę tę wyrażamy krótko, mówiąc, że "pętle nie mogą się przecinać"

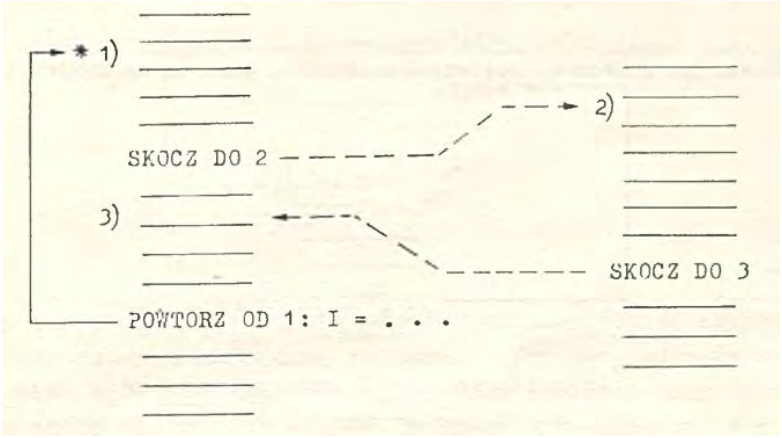
4. Przerwanie powtarzania danego odcinka programu może nastąpić nie tylko dla wartości I równej L; w zasięgu rozkazu **POWTORZ** może znajdować się rozkaz sterujący, który spowoduje przejście do wykonania rozkazu, położonego poza zasięgiem **POWTORZ.**

Mówimy wówczas, że działanie **POWTORZ** zostało przerwane.



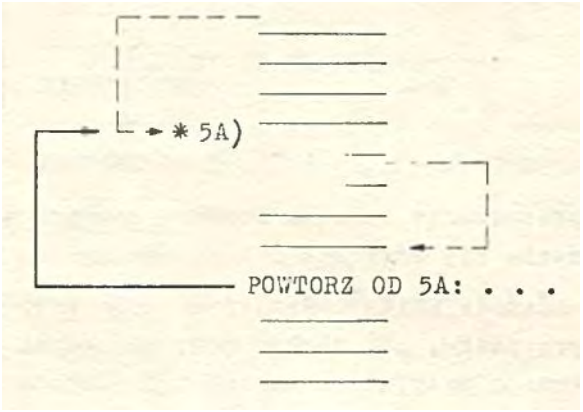
Jeśli natomiast powtarzanie zasięgu rozkazu **POWTORZ** zostało zakończone przez zrównanie się wartości I i L, mówimy, że działanie **POWTORZ** zostało wykonane.

5. Zarówno w przypadku, gdy rozkaz **POWTORZ** został przerwany, jak i wykonany, ostatnia z przyjętych wartości I zostaje zachowana w dalszym ciągu programu. Umożliwia to również chwilowe przerwanie działania **POWTORZ** w celu wykonania pewnego odcinka programu, a następnie powrót do zakresu danego **POWTORZ**; ten odcinek programu może korzystać z aktualnej wartości I, lecz nie może zmieniać jej wartości:

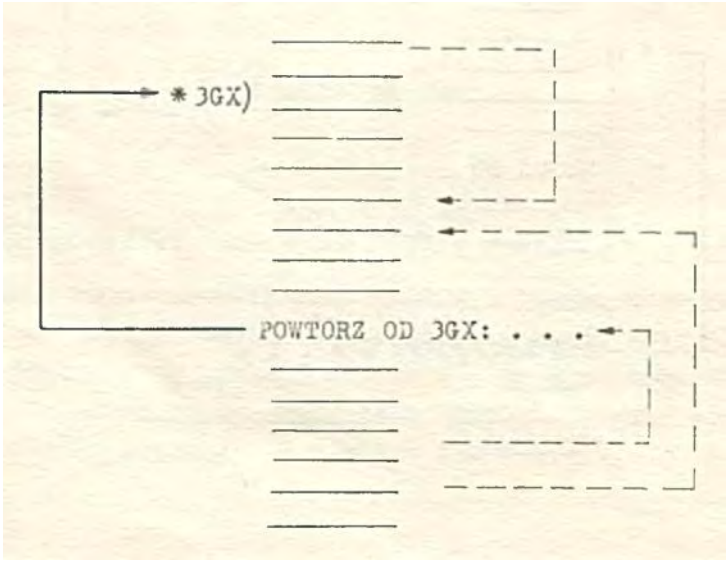


6. Przejście skokiem do wykonania rozkazu znajdującego się w zasięgu pewnego rozkazu POWTORZ jest możliwe ponadto w następujących przypadkach:

- gdy wykonywanym rozkazem jest rozkaz o numerze α ,
- gdy przejście odbywa się na skutek wykonania rozkazu sterującego, znajdującego się w zasięgu tego samego rozkazu **POWTORZ**:



Poza opisanymi wyżej przypadkami, żadne przejście do wykonania rozkazu z zasięgu **POWTORZ** nie jest dopuszczalne. Niedopuszczalne są więc sytuacje, przedstawione na schemacie poniżej:



7. Jeśli J, K, L są ułamkowe i wartości ich są zaokrąglone tak, że wartość I nie może stać się równą wartości L dla żadnego powtórzenia, wówczas zakończenie powtarzania następuje dla wartości I najbliższej wartości L.

IDZ DO ROZDZIAŁU : nazwa

Przykłady:

```
IDZ DO ROZDZIAŁU: GAUSS SEIDEL
IDZ DO ROZDZIAŁU: LICZENIE PIERWIASTKOW
```

Znaczenie:

Rozkaz powoduje wczytanie z pamięci bębnowej do pamięci wewnętrznej maszyny nowego rozdziału o podanej nazwie, a następnie przejście do wykonania pierwszego rozkazu z nowo wczytanego rozdziału.

Forma:

Nazwa rozdziału: ciąg liter, cyfr i spacji, rozpoczynający się od litery. Własność identyfikującą posiada 10 pierwszych znaków /nie licząc spacji/.

Uwaga:

Po wczytaniu nowego rozdziału wszelkie deklaracje zmienne, numery i nazwy podprogramów, używane w starym rozdziale, przestają być aktualne.

WROC

Znaczenie:

Rozkaz ten powoduje przejście od wykonywania rozkazów podprogramu do wykonywania rozkazów tego programu, który ten podprogram wywołał. Rozkaz **WROC** jest ostatnim wykonywanym rozkazem podprogramu. Przejście następuje do tego miejsca programu wywołującego, w którym nastąpiło wywołanie podprogramu. Mówiąc krótko, rozkaz **WROC** realizuje powrót z programu podległego do programu nadrzędnego.

GDY A > B : α, **INACZEJ** β

Przykłady:

```
GDY A x X + B > 0 : 1A, INACZEJ 4B
GDY 0 > X : 81, INACZEJ NASTĘPNY
GDY B(I,J)>C(I) : NASTĘPNY, INACZEJ 2
```

Znaczenie:

Gdy podana nierówność jest spełniona, rozkaz powoduje przejście do rozkazu o numerze α ; w przeciwnym przypadku do rozkazu o numerze β .

Jeżeli zamiast α lub β występuje słowo NASTEPNY, oznacza ono rozkaz wypisany w najbliższej kolejności.

Forma:

A, B - wyrażenia arytmetyczne

α, β – numery rozkazów lub słowo NASTEPNY.

Uwaga:

Wartości A i B mogą mieć ten sam lub różny charakter, tzn, jedno z nich może mieć wartość całkowitą, zaś drugie ułamkową.

GDY A = B ; α, **INACZEJ** β

Przykłady:

```
GDY A = B1 : 5, INACZEJ 2
GDY ALFA (I, K + 4) = O : NASTĘPNY, INACZEJ 4C
```

Znaczenie:

Gdy podana równość jest spełniona, rozkaz powoduje przejście do rozkazu o numerze α; w przeciwnym przypadku do rozkazu o numerze β.

Jeśli zamiast C lub β występuje słowo NASTEPNY, oznacza ono rozkaz wypisany w najbliższej kolejności.

Forma:

A, B - wyrażenia arytmetyczne

α, β - numery rozkazów lub słowa NASTEPNY

Uwaga:

Patrz rozkaz GDY A>B.

GDY BYL NADMIAR: α, **INACZEJ** β

Przykłady:

```
GDY BYL NADMIAR: 1A, INACZEJ 3
GDY BYL NADMIAR: 4, INACZEJ NASTEPNY
```

Znaczenie:

Gdy we wskaźniku nadmiaru znajduje się liczba 1, rozkaz ten powoduje przejście do rozkazu o numerze α; w przeciwnym przypadku /gdy we wskaźniku nadmiaru znajduje się liczba 0/ do rozkazu o numerze β .

Jeśli zamiast α lub β występuje słowo NASTEPNY, oznacza ono rozkaz wypisany w najbliższej kolejności.

Równocześnie rozkaz ten powoduje wpisanie liczby 0 do wskaźnika nadmiaru.

Forma:

α, β – numery rozkazów. Jeden z nich /lub oba/ może być zastąpiony słowem NASTEPNY.

Uwaga:
Pojawienie się liczby 1 we wskaźniku nadmiaru jest spowodowane wystąpieniem w maszynie liczby, przekraczającej zakres przyjętej skali /patrz LICZBA CAŁKOWITA, LICZBA UŁAMKOWA/.

GDY KLUCZ I : α, **INACZEJ** β

Przykłady:

```
GDY KLUCZ 15: 2CD, INACZEJ 3
GDY KLUCZ I : NASTĘPNY, INACZEJ 5X
```

Znaczenie:

Gdy klucz, wskazany przez rozkaz, jest podniesiony w trakcie działania maszyny, rozkaz ten powoduje przejście do rozkazu o numerze α; w przeciwnym przypadku do rozkazu o numerze β .

Jeśli zamiast α lub β występuje słowo NASTEPNY, oznacza ono rozkaz wypisany w najbliższej kolejności.

Forma:

I - zmienna prosta lub liczba całkowita przyjmująca wartości 0, 1, 2, ..., 35.

α, β - numery rozkazów, lub słowa NASTĘPNY.

STOP α

Przykłady:

```
STOP 10
STOP 1 ABC
```

Znaczenie:

Rozkaz powoduje zatrzymanie pracy maszyny. Po ponownym naciśnięciu przycisku START maszyna rozpoczyna pracę od rozkazu o numerze α.

Forma:

α – numer rozkazu lub słowo NASTEPNY

Uwaga:
Gdy zamiast numeru α w rozkazie STOP α występuje słowo NASTEPNY; oznacza ono rozkaz wypisany w najbliższej kolejności.

Rozkazy arytmetyczne i bulowskie

A = B

/formuła arytmetyczna/

Przykłady:

```
A = B + C
X1 = ALFA + LOG(SIN(X))
I = I + 1
S = S + A (I, J) x B(J, K)
D = 45.6789
```

Znaczenie:

Rozkaz ten powoduje obliczenie wartości wyrażenia arytmetycznego B oraz nadanie tej wartości zmiennej A, stojącej po lewej stronie znaku = Formuła arytmetyczna nadaje lub zmienia dotychczasowe wartości zmiennych.

Forma:

A - zmienna ułamkowa lub całkowita, prosta lub indeksowana,
B - wyrażenie arytmetyczne.

Formuła arytmetyczna nie może zajmować więcej niż jeden wiersz formularza.

Reguły:

1. Jeśli zmienna A jest całkowita, zaś wyrażenie B jest ułamkowe, wówczas zmiennej A nadaje się wartość wyrażenia B, zaokrągloną do liczby całkowitej.

2. Jeśli zmienna A jest ułamkowa, zaś wyrażenie B jest całkowite, wówczas zmiennej A nadaje się wartość B sprowadzoną do postaci ułamkowej, zgodnie z aktualnie ustaloną skalą.

Uwaga:

Znak = występujący w formule, różni się znaczeniem od ogólnie przyjętego w matematyce; w formule określa on nie równość, lecz zmianę dotychczasowej wartości zmiennej.

A = B

/formuła bulowska/

Przykłady:

```
A ≡ B + C
SDF ≡ 0000.4567.3241 x A
D ≡ D*(-23)
```

Znaczenie:

Rozkaz ten powoduje obliczenie wartości wyrażenia bulowskiego B i nadanie obliczonej wartości zmiennej A, stojącej po lewej tronie znaku ≡. Formuła bulowska nadaje lub zmienia dotychczasowe wartości zmiennych.

Forma:

A - zmienna ułamkowa lub całkowita /z reguły prosta/

B - wyrażenie bulowskie

Formuła bulowska nie może zajmować więcej niż jeden wiersz formularza.

Uwagi:

1. Znak ≡ , występujący w formule bulowskiej, różni się znaczeniem od ogólnie przyjętego w matematyce; w formule określa on nie równość, lecz zmianę dotychczasowej wartości zmiennej.
2. Ponieważ słowo bulowskie może być traktowane jako liczba /ułamkowa lub całkowita/, zatem za pośrednictwem formuł bulowskich można nadawać wartości zmiennym, używanym w dalszym ciągu jako symbole liczb w wyrażeniach arytmetycznych i na odwrót: zmienne, których wartości były określone formułami arytmetycznymi, mogą w wyrażeniu bulowskim być traktowane jako symbole słów bulowskich.

(lista wyników) = nazwa (lista argumentów)

/formuła operacyjna/

Przykłady:

```
(X, ALFA, C1) = TRY X + Y, 7.89 - X + Y, SIN(X), R MALE
(*Y) = KROK RK(*Y, PRAWE STR( ), 0.00001 , 5)
(*A) = MN MACIERZY (* B, *C, 5, 6, 7)
(DF, T) = GX6(•,•,•,•, 45.6789)
```

Znaczenie:

Rozkaz ten określa wartości argumentów danego podprogramu, powoduje jego wykonanie i otrzymane rezultaty przypisuje zmiennym i blokom, występującym na liście wyników.

Formuła operacyjna stanowi uogólnienie formuły arytmetycznej; ta ostatnia nadaje wartości pojedynczej zmiennej przez wykonanie określonych działań na zmiennych wyrażenia arytmetycznego, podczas gdy formuła operacyjna nadaje wartości układowi zmiennych poprzez wykonanie działań podprogramu na układzie jego argumentów. W miejsce argumentów, będących liczbami, można podstawiać wartości wyrażeń, których wyniki są zgodne co do charakteru z odpowiednimi argumentami podprogramu.

Forma:

Charakter zmiennych i ich kolejność na listach formuły operacyjnej muszą być zgodne z charakterem i kolejnością symboli, występujących na odpowiednich listach w deklaracji PODPROGRAM.

Reguła:

W chwili wykonywania formuły operacyjnej wszystkie argumenty wywoływanego podprogramu muszą być określone. Tym niemniej formuła operacyjna może określać przez podstawienie tylko niektóre z nich - pozostałe muszą być w tym przypadku podstawione uprzednio przez rozkaz PODSTAW. Na liście argumentów formuły wypisuje się wówczas tylko te zmienne, które ulegają podstawieniu. W pozostałych miejscach wpisuje się kropki.

Uwagi:

1. Nie jest konieczne wypisywanie kropek za ostatnim podstawianym argumentem. Tak więc, zamiast pisać:

   (A, B) = TRAP(c, D, • , • , • , • , •)

   można krócej pisać:

   (A, B) = TRAP(C, D)

2. Podstawione argumenty zachowują swoją wartość aż do wykonania kolejnego następnego podstawienia. Tak więc zmiana jednego argumentu podprogramie nie wymaga dokonania wszystkich pozostałych podstawień.

PODSTAW: nazwa (lista)

Przykłady:

```
PODSTAW: TRY(A, B, c)
PODSTAW: TRANS(• , • , S8)
PODSTAW: CALKA(• , F())
PODSTAW: ILMA( , • , *A, 5)
```

Znaczenie:

Rozkaz ten powoduje określenie wartości argumentów podprogramu o podanej nazwie. Inaczej mówiąc, rozkaz ten powoduje podstawienie pewnych wielkości w miejsce odpowiednich argumentów podprogramem.

Forma:

1. Na liście wypisuje się tylko te wielkości, które ulegają podstawieniu. W miejscach przeznaczonych dla pozostałych argumentów /nie podstawianych tym rozkazem/ stawia się kropki, zachowując rozdzielające je przecinki. Kropek tych można nie stawiać po ostatniej wypisanej na liście wielkości.
2. Nazwy funkcji, podstawianych do podprogramu, uzupełnia się parą nawiasów, postawionych za nazwą funkcji; natomiast nazwy podstawianych bloków poprzedza się, zgodnie z ogólną regułą, symbolem * .

Reguły:

1. Charakter podstawianych wielkości musi być zgodny *z* charakterem argumentów, zadeklarowanych deklaracją PODPROGRAM.
2. Podstawione wielkości pozostają tak długo argumentami podprogramu, aż nie nastąpi nowe podstawienie w miejsce tych samych argumentów. Może to być dokonane za pośrednictwem innego rozkazu PODSTAW bądź też formuły operacyjnej.

USTAW SKALE DZIESIETNIE : I

Przykłady:

```
USTAW SKALE DZIESIETNIE : 3
USTAW SKALE DZIESIETNIE : ALFA
```

Znaczenie:

Rozkaz ten ustala skalę dziesiętną wszystkich zmiennych ułamkowych, prostych i indeksowanych w rozkazach o dalszej kolejności wykonania - aż do nowego rozkazu ustalającego skalę.

Forma:
I - zmienna prosta całkowita lub liczba całkowita o wartości:
0, 1 , ..., 10

Reguła:
W każdym programie wykonanie rozkazów zawierających zmienne ułamkowe musi być poprzedzone rozkazem USTAW SKALE DZIESIETNIE lub USTAW SKALE BINARNIE. Ustalona skala zachowuje swe znaczenie przy przejściu do podprogramów oraz przy przejściu z jednego rozdziału do drugiego.

USTAW SKALE BINARNIE: I

Przykłady:
USTAW SKALE BINARNIE: 5
USTAW SKALE BINARNIE: ALFA

Znaczenie:
Rozkaz ten ustala skalę dwójkową wszystkich ułamków binarnych w rozkazach o dalszej kolejności wykonania - aż do nowego rozkazu ustalającego skalę.

Forma:
I — zmienna prosta lub liczba całkowita o wartości 0, 1, ..,33.

Reguła:
Patrz rozkaz USTAW SKALE DZIESIETNIE

ZWIEKSZ SKALE DZIESIETNIE O I: lista

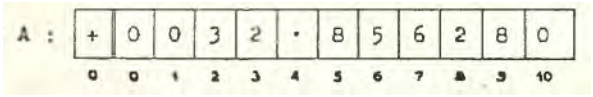
Przykłady:
ZWIEKSZ SKALE DZIESIETNIE O ABC : A, *B, C, *D
ZWIEKSZ SKALE DZIESIETNIE O -7 : EF, *G

Znaczenie:
Rozkaz ten powoduje zwiększenie skali dziesiętnej o wielkości I zmiennych i bloków, wymienionych na liście.

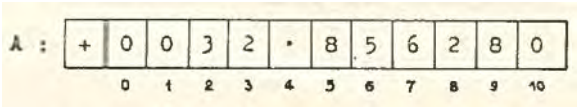
Forma:
I - zmienna prosta lub liczba całkowita o wartości
-10, -9, -8, -7, ..., 0, 1, ..., +10
lista - zbudowana ze zmiennych prostych i nazw bloków.

Reguła:
Rozkaz ten można stosować jedynie wówczas, gdy wielkości, których nazwy występują na liście, są zapisane w maszynie w skali dziesiętnej /przy pomocy-rozkazu USTAW SKALE DZIESIETNIE/.

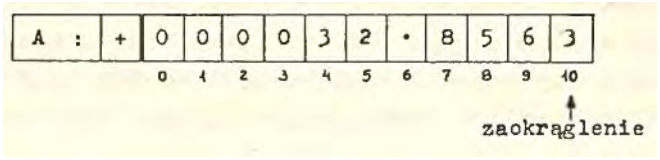
Uwagi:
1. W przypadku zmiany ze skali większej na mniejszą (I<0) może wystąpić nadmiar wraz z wszystkimi jego konsekwencjami.
Przykład:



2. W przypadku zmiany ze skali mniejszej na większą (I>0) może nastąpić strata najmniej znaczących cyfr wartości zmiennych, których
dotyczy ten rozkaz. Wartości tych zmiennych zostają wówczas zaokrąglone .
Przykład:



Rozkaz: ZWIEKSZ SKALE DZIESIETNIE O 2 : A powoduje następującą zmianę zapisu wartości A:



ZWIEKSZ SKALE BINARNIE O I : lista

Przykłady:

```
ZWIEKSZ SKALE BINARNIE 0 KL1 : A, *B, C, *D
ZWIEKSZ SKALE BINARNIE 0 -8 : E, F, *G
```

Znaczenie:
Rozkaz ten powoduje zwiększenie skali Binarnej zmiennych i bloków, wypisanych na liście, o I.

Forma:
I — zmienna prosta całkowita lub liczba całkowita o wartości

-35, -34, ..., +34, +35

Uwaga:
Analogicznie jak w przypadku ZWIEKSZ SKALE DZIESIETNIE.

Rozkazy wejścia i wyjścia

CZYTAJ : lista

Przykłady:

```
CZYTAJ: AB, *ALFA, OMEGA
CZYTAJ: *AXY1
```

Znaczenie:

Rozkaz powoduje wczytanie /w ramach pracy programu/ układu liczb z urządzenia wejściowego do pamięci wewnętrznej maszyny. Ponadto rozkaz ten nadaje wartości zmiennym z listy w ten sposób, że kolejnej zmiennej /lub blokowi/ przypisuje się kolejną liczbę /lub układ liczb/ z urządzenia wejściowego.

Forma:

lista - zbudowana jest ze zmiennych prostych i nazw bloków.

Reguły:

1. Każda liczba, przewidziana jako wartość zmiennej prostej, jak i każdy układ liczb, stanowiący blok, musi zaczynać się od nowego wiersza formularza danych wejściowych.
2. Po ostatniej liczbie zapisanej w bloku następować musi oddzielny wiersz z wypisanym symbolem * .
3. Ilość liczb, stanowiących w programie blok, musi być dokładnie równa ilości przewidzianej przez właściwą deklarację BLOK lub STRUKTURA - w przeciwnym przypadku maszyna przerwie obliczenia, wykazując błąd.
4. Kolejność wypisania zmiennych na liście rozkazu CZYTAJ musi być zgodna z kolejnością wypisania danych na formularzu wejściowym.
5. Jeśli układ liczb na formularzu wejściowym stanowi blok, wówczas w jednym wierszu formularza można pisać kilka z tych liczb, oddzielając je spacjami. W związku z tym, między znakami jednej liczby nie może występować spacja. Kolejność wczytywania liczb, wypisanych w jednym wierszu, jest od lewej do prawej strony formularza.
6. Poszczególne liczby, wiersze lub bloki mogą być opatrzone komentarzami, nie wczytywanymi do pamięci maszyny. Komentarz musi zaczynać się od litery, a kończyć na znaku = lub : . Wśród znaków komentarza nie może występować ani znak =, ani znak : .Komentarz może zajmować część, cały, a nawet kilka wierszy formularza.
7. Dane wejściowe wczytywane są do pamięci maszyny w skali określonej przez ostatnio wykonany rozkaz USTAW SKALE DZIESIETNIE lub USTAW SKALE BINARNIE. Jeżeli pewna wielkość wczytywana nie mieści się w przewidzianej skali, maszyna przerywa obliczenia wykazując błąd.

Uwaga:

Ilość liczb wczytywanych do pamięci maszyny jest równa sumie ilości elementów bloków i ilości zmiennych prostych, występujących na liście rozkazu CZYTAJ. Tak więc, jeśli A i B są zmiennymi prostymi, zaś C nazwą bloku o 25 elementach, rozkaz

```
CZYTAJ: A, B, *C
```

spowoduje wczytanie

1 +1 + 25 = 27

liczb do pamięci maszyny.

CZYTAJ WIERSZ : nazwa bloku

Przykłady:

```
CZYTAJ WIERSZ: ALFA
CZYTAJ WIERSZ: N
```

Znaczenie:

Rozkaz ten powoduje wczytanie do pamięci wewnętrznej maszyny następujących po sobie znaków, tworzących dane wejściowe, począwszy od skrajnie lewego znaku najbliższego wiersza. Każdy znak wiersza, wyrażony za pośrednictwem sześciu zer i jedynek, wpisywany jest do ostatnich sześciu pozycji każdego z kolejnych słów bloku o podanej nazwie. Znaki są wczytywane do znaku P.K. /Powrót Karetki/ włącznie.

Forma:

Na liście może występować tylko jedna nazwa bloku zadeklarowana uprzednio jako całkowita.

Reguła:

Znaki poprzedzone na taśmie symbolem "Litery" aż do pierwszego kolejnego znaku "Cyfry" wyłącznie uzupełniane są w trakcie czytania dodatkową jedynką na dwunastej pozycji słowa..

Uwaga:

Ostatnim znakiem wczytywanego wiersza jest zawsze Powrót Karetki.

CZYTAJ OKTALNIE : lista

Przykłady:

```
CZYTAJ OKTALNIE : AB
CZYTAJ OKTALNIE : *ALFA
CZYTAJ OKTALNIE : AB,*ALFA
```

Znaczenie:

Rozkaz powoduje wczytanie słów bulowskich z urządzenia wejściowego do pamięci wewnętrznej maszyny z jednoczesnym ich przypisaniem kolejnym zmiennym na liście.

Forma:

Na liście występować mogą zmienne proste i nazwy bloków.

Reguła:

Wszystkie reguły rozkazu CZYTAJ odnoszą się do rozkazu CZYTAJ OKTALNIE, z tym że przez liczbę rozumiemy słowo bulowskie.

DRUKUJ (i.j) : lista

Przykłady:

```
DRUKUJ (2.5) : AX, DELTA
DRUKUJ (P.9) : A(K+1), A(K+2), A(K+3), A(K+4)
DRUKUJ (12) : BCD 9
DRUKUJ (3) : EFG I
```

Znaczenie:

Rozkaz powoduje kolejne wypisanie w aktualnie wypisanym wierszu wartości zmiennych podanych na liście. Parametry rozkazu określają ilość miejsc zarezerwowanych dla wypisania wartości każdej zmiennej oraz sposób jej wypisania.

Forma:

lista - składa się ze:

- zmiennych prostych,
- zmiennych indeksowanych o jednym indeksie będącym liczbą lub zmienną,
- zmiennych indeksowanych, których indeksem jest suma zmiennej liczby;

parametry -

1. - I,J - zmienne całkowite lub liczby całkowite,
2. gdy drukowanie dotyczy liczb ułamkowych, I oznacza ilość zarezerwowanych pozycji przed kropką dziesiętną, zaś J ilość pozycji za kropką, łącznie z kropką i znakiem, powyższy rozkaz powoduje więc zarezerwowanie I+J+2 pozycji arkusza wydawniczego na wypisanie każdej liczby,
3. gdy drukowanie dotyczy liczb całkowitych, rozkaz ma postać:

```
DRUKUJ I : lista
```

gdzie I oznacza ilość zarezerwowanych pozycji na wypisanie liczby całkowitej. Łącznie z miejscem na ewentualny znak powyższy rozkaz powoduje zarezerwowanie I+1 pozycji arkusza wydawniczego.

Reguły:

1. W przypadku liczb ułamkowych sposób drukowania jest następujący:
   - kropkę dziesiętną wypisuje się w I+2 zarezerwowanej pozycji,
   - część całkowitą wypisuje się przed kropką. Bezpośrednio przed liczbą dodatnią pisze się znak + , przed ujemną znak - . Początkowe zarezerwowane pozycje, nie zajęte cyframi znaczącymi liczby i znakiem liczby, wypełniane są spacjami. Gdy żądamy zero cyfr przed kropką, to znak wystąpi tuż przed kropką

```
-.7341
```

gdy żądamy większej ilości cyfr przed kropką i część całkowita jest zero, to między znakiem a kropką zostanie napisana cyfra "0"

```
-0.7341
```

2. W przypadku liczb całkowitych liczbę drukuje się bez kropki pozycyjnej, tak że ostatnia cyfra liczby znajdowała się w I+1

zarezerwowanej przez rozkaz pozycji. W liczbach dodatnich opuszcza się znak + , w ujemnych znak stawiany jest bezpośrednio przed pierwszą znaczącą cyfrą liczby. Początkowe zarezerwowane pozycje, nie zajęte cyframi znaczącymi oraz znakiem - , wypełniane są spacjami. W wypadku gdy liczba jest zerem, piszemy 0 lub -0 w zależności od znaku.

3. Skala drukowanych liczb ułamkowych jest zgodna z ostatnio wykonanym rozkazem USTAW SKALE.
4. W przypadku zbyt małej ilości miejsc zarezerwowanych na wypisanie liczby maszyna się zatrzymuje, sygnalizując błąd.

Uwaga:
Należy uważać, aby rozkazy DRUKUJ wraz z rozkazami TEKST i SPACJA nie przekroczyły pojemności jednego wiersza, która wynosi 69 znaków.

DRUKUJ WIERSZ: nazwa bloku

Przykłady:

```
DRUKUJ WIERSZ : ALFA
DRUKUJ WIERSZ : M
```

Znaczenie:
Rozkaz ten powoduje wypisanie wiersza, począwszy od skrajnie lewej pozycji arkusza w ten sposób, że każdy kolejny znak dalekopisowy jest określony przez sześć najmniej znaczących pozycji kolejnego słowa bloku o podanej nazwie. Zakończenie drukowania następuje ze znakiem P.K. /Powrót Karetki/ włącznie.

Forma:
W liście może występować tylko jedna nazwa bloku, zadeklarowana uprzednio jako całkowita.

Reguła:
Ilość słów w bloku, a tym samym ilość wypisanych znaków dalekopisowych nie może przekraczać 69.

DRUKUJ OKTALNIE: lista zmiennych

Przykłady:

```
DRUKUJ OKTALNIE: A, B, C
DRUKUJ OKTALNIE: D(I)
```

Znaczenie:
Rozkaz powoduje wypisanie wartości zmiennych wypisanych na liście, traktując je jako słowa bulowskie /patrz słowo bulowskie/.

Forma:
lista - jest zbudowana ze zmiennych prostych bądź indeksowanych o jednym indeksie, będącym liczbą bądź zmienną.

Reguła:
Słowo jest drukowane w formie oktalowej przy użyciu 6 lub 12 cyfr: 0, 1, 2, 3, 4, 5, 6, 7 oraz kropek, stawianych co 3 znaki.
Łącznie, na wydrukowanie słowa zarezerwowanych jest 15 pozycji.

W przypadku gdy zmienna występująca na liście jest zadeklarowana jako całkowita na wydrukowanie słowa jest zarezerwowane 7 pozycji.

TEKST:

/dane tekstu/

Przykłady:

```
TEKST:
PIERWIASTEK =

TEKST:
TEMPERATURA OTOCZENIA WYNOSI:
```

Znaczenie:
Rozkaz powoduje kolejne wypisanie danych tekstu w aktualnie drukowanym wierszu.

Forma:
Dane tekstu wypisujemy w następnym wierszu po słowie TEKST. Początkiem danych tekstu jest pierwszy znak, nie będący spacją; końcem - ostatni znak nie będący spacją.

Uwaga:
Wewnątrz danych tekstu spacje zachowują swe znaczenie.

TEKST WIERSZY n:

/n wierszy tekstu/

Przykłady:

```
TEKST WIERSZY 2:
ROZWIĄZANIE UKŁADU ROWNAN LINIOWYCH
METODA ITERACJI

TEKST WIERSZY 1 :
TABLICA WARTOŚCI FUNKCJI GAMMA
```

Znaczenie:
Rozkaz, powoduje przepisanie na formularz wyjściowy podanych n wierszy tekstu.

Forma:
n - liczba naturalna.

Reguły:
1. Przy przepisywaniu uwzględnione są wszystkie pozycje danych tekstu, wliczając spacje, od początku aż do końca wierszy.
2. Wiersze niezapisane liczą się tak samo, jak wiersze zapisane. Umożliwia to tworzenie odstępów między wierszami wypisywanego tekstu.

SPACJA n

Przykłady:

```
SPACJA 3
SPACJA
```

Znaczenie:
Rozkaz powoduje pozostawienie n spacji w aktualnie wypisywanym wierszu.

Forma:
n - liczba naturalna

Uwaga:
Zamiast pisać SPACJA 1, można krócej pisać SPACJA

LINIA n

Przykłady:

```
LINIA 5
LINIA 8
LINIA
```

Znaczenie:
Rozkaz powoduje przejście do wypisywania danych wyjściowych począwszy od skrajnie lewej pozycji n-tego kolejnego wiersza formularza danych wyjściowych.

Forma:
n - liczba naturalna.

Reguły:
1. Rozkazy: **DRUKUJ, SPACJA** oraz **TEKST** powodują kolejne wypisywanie wyników w obrębie jednego wiersza. Przejście do nowego wiersza formularza wymaga zastosowania rozkazu **LINIA** i wykonujemy je wtedy, gdy wymaga tego żądana przez nas forma danych wyjściowych lub gdy zachodzi obawa przekroczenia dopuszczalnej /9/ ilości znaków, mieszczących się w jednym wierszu formularza.
2. Rozkaz **LINIA n** można napisać w formie **LINII n.**
3. Rozkaz **LINIA 1** można krócej pisać: **LINIA.**

Uwaga:
Rozkaz **LINIA n** powoduje pozostawienie na formularzu wyjściowym
n-1 wierszy wolnych.

Rozkazy współpracy z bębnem

CZYTAJ Z BEBNA OD I : lista

Przykłady:

```
CZYTAJ Z BEBNA OD 120: A, *BETA
CZYTAJ Z BEBNA OD 2134: X, A, -
      R, B, *DZETA, KSI
CZYTAJ Z BEBNA OD KSI: *C, *D
```

Znaczenie:

Rozkaz powoduje przepisanie liczb z pamięci bębnowej do pamięci wewnętrznej maszyny traktując je jako wartości zmiennych i bloków, wymienionych na liście. Liczby te przepisuje się z bębna kolejno poczynając od miejsca na bębnie, oznaczonego numerem I. Kolejne liczby stanowią wartości kolejnych zmiennych i bloków w. takim porządku, w jakim są one wypisane na liście.

Forma:

I - liczba naturalna lub zmienna prosta, całkowita o wartości dodatniej,

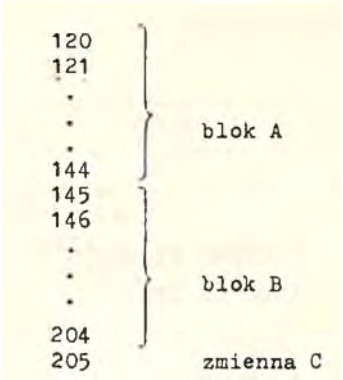
lista - jest utworzona ze zmiennych prostych i nazw bloków.

Uwagi:

1. Ilość liczb, wczytywanych z bębna do pamięci wewnętrznej maszyny,
   jest równa sumie ilości zmiennych prostych i ilości elementów bloków, wymienionych w liście.
   Tak, więc jeśli A jest nazwą bloku o 25 elementach B jest nazwą bloku o 60 elementach, zaś C jest zmienną prostą, wówczas rozkaz:

   ```
   CZYTAJ Z BEBNA OD 120: A, B, C
   ```

   powoduje wczytanie do pamięci wewnętrznej maszyny ogółem 86 liczb, zapisanych na bębnie w miejscach:



2. Kolejność wypisania zmiennych na liście rozkazu CZYTAJ Z BEBNA musi być zgodna z kolejnością zapisu liczb na bębnie.

PISZ NA BEBEN OD I: lista

Przykłady:

```
PISZ NA BEBEN OD 345: AB, CDEF, *D4
PISZ NA BEBEN OD GHI: KLM
```

Znaczenie:

Rozkaz powoduje przepisanie wartości zmiennych, podanych na liście z pamięci wewnętrznej- maszyny do pamięci Bębnowej. Liczby te wpisuje się do pamięci bębnowej kolejno, począwszy od numeru miejsca na bębnie I. Rozkaz ten można nazywać "odwrotnym" w stosunku do rozkazu CZYTAJ Z BEBNA.

Forma:

I - liczba naturalna lub zmienna prosta, całkowita o wartości dodatniej

lista - jest utworzona ze zmiennych prostych i nazw bloków.

Uwaga:

Analogiczna, jak w rozkazie CZYTAJ Z BEBNA.

Komentarz

Przykłady:

```
K) PIERWIASTEK JEST UJEMNY
K) PRZYPADEK, GDY X-GAMMA JEST > O
K) BADANIE, CZY WARUNEK ZBIEZ-
K)     NOSCI JEST SPEŁNIONY
```

Znaczenie:

Komentarz nie wpływa na działanie programu, posiada jedynie znaczenie objaśniające. Używa się go, aby uczynić program bardziej przejrzystym.

Forma:

Komentarz może składać się z kilku wierszy. Jeden wiersz komentarza stanowi ciąg dowolnych znaków dalekopisowych, którego pierwszym znakiem jest litera K, zaopatrzona z prawej strony w nawias zamykający.

PODPROGRAMY W JĘZYKU SAKO

Podprogramem nazywamy wyodrębniony fragment programu, służący do wykonywania pewnych ustalonych działań na przekazanych mu wielkościach. Ściślej mówiąc, podprogram realizuje w maszynie operację typu:

$$(u, v, \ldots, w) = f(x, y, \ldots, z)$$

przekształcającą układ argumentów x, y, ..., z na układ wyników u, v, ..., w.

Podprogram stanowi zazwyczaj część programu wielokrotnie wykonywaną; przez wyodrębnienie jej w podprogram całość programu staje się krótsza i bardziej przejrzysta. Dla często spotykanych operacji tworzy się raz na zawsze ułożone podprogramy. Korzystanie z nich jest proste i nie wymaga każdorazowego programowania, lecz tylko dołączania uprzednio napisanych podprogramów do tzw. programu głównego.

Przygotowanie programów dla maszyny cyfrowej jest tym łatwiejsze, im więcej istnieje przygotowanych zawczasu podprogramów i im łatwiejszy jest sposób ich wykorzystania w każdym z konkretnych przypadków. Z tego względu przy budowie języka SAKO szczególną uwagę zwrócono na system współpracy programu głównego z podprogramami.

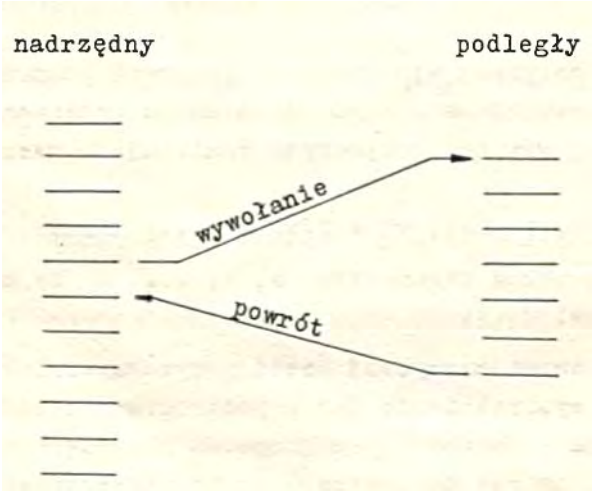
Terminologia i oznaczenia

W punkcie tym opisane są pewne pojęcia i terminy, używane w alszym ciągu.

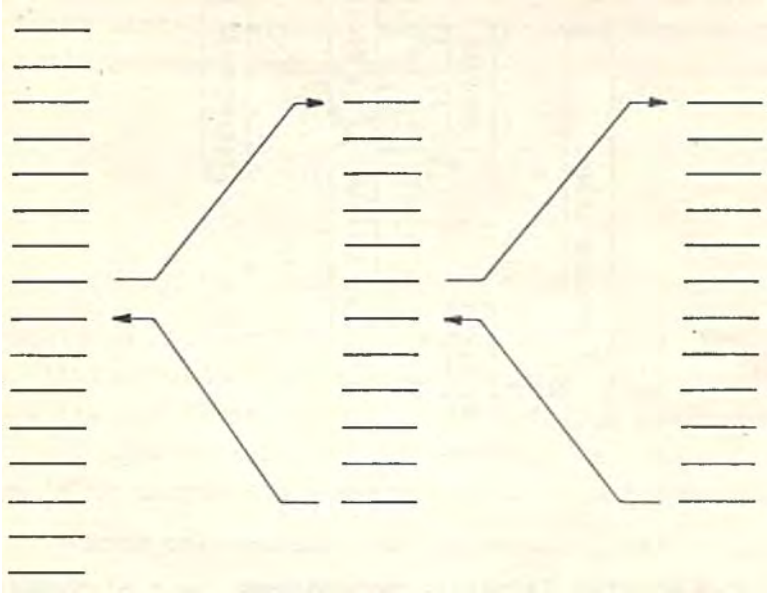
Przejście do wykonania podprogramu nazywamy wywołaniem podprogramu. W trakcie obliczeń jeden podprogram bywa zazwyczaj wywoływany kilkakrotnie.

Program, wywołujący podprogram, nazywa się nadrzędnym w stosunku do danego podprogramu; podprogram wywoływany nosi nazwę podległego w stosunku do programu nadrzędnego.

Przejście do wykonywania programu nadrzędnego nazywa się powrotem.



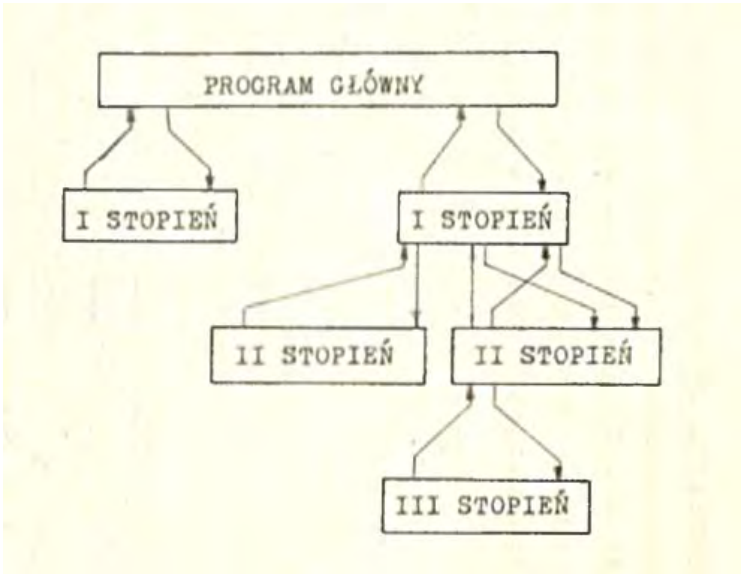
Program, nadrzędny w stosunku do pewnego podprogramu może być sam podprogramem.



Zachodzi wówczas przypadek tzw. wielostopniowego korzystania z podprogramów.

Część programu, która nie jest podległa żadnej innej części, nosi nazwę programu głównego. Każdy rozdział programu pisanego w SAKO posiada dokładnie jeden program główny raz ewentualnie pewną liczbę podprogramów.

Podprogramy, podległe programowi głównemu, nazywają się podprogramami I stopnia; podprogramy, podległe podprogramom I stopnia, noszą nazwę podprogramów II stopnia itd. O tym, czy podprogram jest podprogramem tego lub innego stopnia, decyduje sposób jego wykorzystania w konkretnym rozdziale programu. Ten sam podprogram może być w jednym rozdziale programu podprogramem stopnia I, zaś w drugim - podprogramem stopnia II. Poniżej podajemy schemat wielostopniowego korzystania z podprogramów:



Zasadniczym efektem działania podprogramu jest otrzymanie układu wyników, określonych przez wykonanie pewnych operacji arytmetycznych lub bulowskich na układzie argumentów podprogramu:

$$(u, v, \ldots, w) = f(x, y, \ldots, z)$$

wyniki — argumenty

Operacją takiego typu jest na przykład przekształcenie współrzędnych przy przejściu z jednego układu odniesienia do drugiego; argumentami są stare współrzędne, zaś wynikami nowe, przekształcone.

Gdy wynikiem operacji jest jedna liczba, operacja ta nosi nazwę funkcji:

$$u = F(x, y, \ldots, z)$$

Każdy podprogram posiada ściśle określoną i ustaloną ilość, kolejność i charakter argumentów i wyników, wyrażonych przy pomocy symboli ogólnych. Aby wywołać podprogram,należy te symbole ogólne zastąpić symbolami o konkretnych znaczeniach, takich jak liczby, bloki itp. Proces zastępowania symboli ogólnych konkretnymi nosi nazwę podstawienia.

Argumentami operacji mogą być w zasadzie dowolne obiekty matematyczne tak jak liczby, wektory, liczby zespolone itp. W szczególności argumentami mogą być inne operacje lub funkcje, przykładem może tutaj służyć operacja, zwana "krokiem całkowania" układu równań różniczkowych zwyczajnych:

$$\begin{aligned} y_0' &= f_0(y_0, y_1, \ldots, y_n) \\ y_1' &= f_1(y_0, y_1, \ldots, y_n) \\ &\cdots\cdots\cdots\cdots \\ y_n' &= f_n(y_0, y_1, \ldots, y_n) \end{aligned}$$

Operacja ta na podstawie układu wartości $y_i(t)$, kroku całkowania h, liczby równań n i układu funkcji $f_i()$ określa układ wartości $y_i(t+h)$. Układ funkcji $f_i( )$ może być traktowany jako operacja f(), przekształcająca układ liczb y_i na układ liczb y_i. Operację tę nazywa się zazwyczaj "obliczaniem prawych stron".

Tak więc "krok całkowania" jest operacją typu:

$$(*z) = g(*y, h, n, f())$$

gdzie przez *y, *z oznaczamy wektory rozwiązań w punkcie t i t+h.

Aby wyróżnić spośród innych operacje tego typu, wprowadzamy na stępującą definicję:

1 . Operacja, której żaden argument nie jest operacją, nosi nazwę operacji pierwszego rzędu.

2. Operacja, której jeden argument jest operacją rzędu N, zaś wśród pozostałych argumentów nie ma operacji o rzędzie większym od N, nosi nazwę operacji rzędu N+1.

W myśl tej definicji, przekształcenie układu współrzędnych jest operacją rzędu I, natomiast "krok całkowania" jest operacją rzędu II.

Podprogram, który definiuje operację rzędu N, nosi nazwę podprogramu rzędu N.

Operacja drugiego rzędu, której wynikiem jest jedna liczba, nazywa się zgodnie z terminologią matematyczną funkcjonałem. Przykładem funkcjonału jest operacja obliczania całki z zadanej funkcji. Operacja ta na podstawie liczb a i b /granic całkowania/ liczby ε /żądanej dokładności obliczeń/ oraz funkcji podcałkowej g() znajduje liczbę c:

$$c = \int_a^b g(x)dx + R \qquad |R| < \varepsilon$$

Operacja ta jest więc typu:

$$c = f(a, b, \varepsilon, g())$$

Operacje rzędu II i wyższego często występują w problemach obliczeniowych. Narzuca to konieczność zbudowania takiego systemu korzystania z podprogramów, który by w łatwy sposób pozwalał stosować operacje wyższych rzędów.

Ogólne własności systemu korzystania z podprogramów w języku SAKO

Do cech charakterystycznych systemu SAKO, związanych z wykorzystaniem podprogramów należy:

- możliwość tworzenia i korzystania z podprogramów dowolnie wysokiego rzędu,
- możliwość wielostopniowego korzystania z podprogramów, ograniczona jedynie pojemnością pamięci maszyny,
- niezależna numeracja rozkazów i niezależne stosowanie zmiennych w każdym podprogramie i programie głównym,
- możliwość wielokrotnego wywoływania podprogramu przez dowolny podprogram niższego stopnia,
- możliwość dokonywania podstawień w podprogramach przez dowolne podprogramy niższego stopnia,
- możliwość traktowania jako funkcji języka tych podprogramów, których językiem jest jedna liczba,
- możliwość używania jako argumentów podprogramu:
  - liczb,
  - zmiennych,
  - bloków liczbowych,
    operacji.
- możliwość otrzymywania w wyniku:
  - liczb,
  - bloków liczbowych.

Budowa podprogramów SAKO

Podprogramy pisze się w standardowym języku SAKO. Program główny wraz z odpowiadającymi mu podprogramami musi mieścić się w jednym rozdziale programu. Podprogram rozpoczyna się zawsze od deklaracji

```
PODPROGRAM: (lista wyników) = nazwa (lista argumentów)
```

Deklaracja ta określa nazwę danego podprogramu oraz ilość, kolejność i charakter jego argumentów i wyników.

Dla większej przejrzystości oraz dla ułatwienia pracy translatora wyróżnia się spośród argumentów i wyników podprogramów

- bloki liczbowe, poprzedzając ich nazwy symbolem * ,
- operacje, przez dopisywanie po ich nazwach pary nawiasów ().

```
PODPROGRAM: (U, V) = TRANS(X, Y, ALFA)
PODPROGRAM: (*C) = MN MACIERZY (*A, *B, N, M, L)
PODPROGRAM: (*Z) = KROK RK (*Y, H, N, T, F())
```

Po deklaracji PODPROGRAM następują wszelkie inne deklaracje i rozkazy podprogramu. Występować w nich mogą tylko zmienne, wprowadzone przez rozkazy podprogramu, bądź też zmienne, wymienione w deklaracji PODPROGRAM. Tym ostatnim wartości nadaje program nadrzędny przez

podstawienie. Ostatnim wykonywanym rozkazem podprogramu jest zawsze rozkaz **WROC**, który realizuje powrót do programu nadrzędnego. Tak więc jedyną komunikacją między programem nadrzędnym a podległym są z jednej strony rozkazy **PODSTAW** oraz formuły arytmetyczne i operacyjne, które powodują podstawienia argumentów i wywołują program podległy /będzie o nich mowa w punkcie 4.4 z drugiej zaś strony rozkaz **WROC**.

W podprogramach stosuje się symbolikę niezależną od innych podprogramów i programu głównego. Oznacza to, że w podprogramie można używać nazw zmiennych i numerów rozkazów takich samych, jak w pozostałych częściach programu, mimo różnych znaczeń, jakie mogą one posiadać. Wynika stąd możliwość wielokrotnego wykorzystywania raz przygotowanego podprogramu; podprogram taki, posiadający niezależną symbolikę, można dołączać do każdego programu głównego. Również wszelkie deklaracje programu nadrzędnego /z wyjątkiem deklaracji, dotyczących skali/ tracą moc w stosunku do programu podległego.

Rozkazy i deklaracje podprogramu mają identyczną formę i znaczenie jak rozkazy programu głównego. Poniżej przytaczamy przykład budowy prostego podprogramu:

```
PODPROGRAM: (U, V) = TRANS(X, Y, ALFA)
C = COS (ALFA)
S = SIN (ALFA)
U = X x C + Y x S
V = –X x S + Y x C
WROC
```

Jeśli operacja jest funkcją, tzn. posiada tylko jeden wynik liczbowy, wówczas forma budowy podprogramu, definiującego tę funkcję ma nieco odmienną postać od formy innych podprogramów.

Deklaracja **PODPROGRAM**, odpowiadająca takiemu podprogramowi ma wówczas postać

```
PODPROGRAM: nazwa (lista argumentów)
```

zaś ostatnim wykonywanym rozkazem arytmetycznym takiego podprogramu jest rozkaz

```
nazwa funkcji = odp. wyrażenie arytmetyczne
```

lub:

```
nazwa funkcji = odp. wyrażenie bulowskie
```

Przykład:

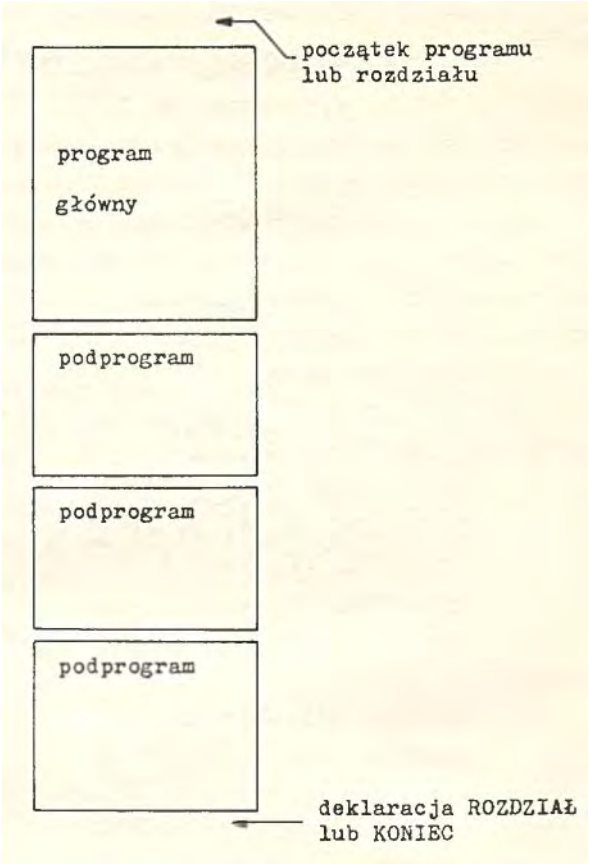
```
    PODPROGRAM: WIELOMIAN (X, *A, N)
    CALKOWITE: N, I
    STRUKTURA (N) : A
    S = 0
*1) S = S x X + A(I)
    POWTORZ OD 1 : I = N(-1)0
    WIELOMIAN() = S
    WROC
```

Poniżej podajemy jeszcze jeden przykład budowy podprogramu, którego wyniki tworzą blok liczbowy:

```
          PODPROGRAM: (*C) = MN MACIERZY (*A, *B, N, M, L)
          CALKOWITE: N,M, L, I, J, K
          STRUKTURA (N,M) : A
          STRUKTURA (M,L) : B
          STRUKTURA (N,L) : C
    **2) S = 0
     *1) S = S + A(I,J) x B(J, K)
          POWTORZ OD 1 : J = 0(1)M
          C(I, K) = S
          POWTORZ OD 2 : I = 0(1)N
          POWTORZ OD 2 : K = 0(1)L
          WROC
```

Korzystanie z podprogramów, napisanych w języku SAKO

Podprogram, używany w określonym rozdziale programu, z reguły wchodzi w skład tego rozdziału. Rozdział składa się z programu oraz pewnej liczby podprogramów, wypisanych po programie głównym:



Korzystanie z podprogramu wymaga wykonania podstawień oraz wywołania podprogramu. Dokonuje się tego za pośrednictwem formuł arytmetycznych, jeśli podprogram definiuje funkcję, lub za pośrednictwem rozkazu

$$(u, w, \bullet\bullet\bullet, w) = f(x, y, \bullet\bullet\bullet, z)$$

zwanego w dalszym ciągu formułą operacyjną. Formuła operacyjna określa argumenty wywoływanego podprogramu, wyniki zaś wykonywanej operacji przyporządkowuje zmiennym, stojącym po lewej stronie znaku = .

Formuła operacyjna stanowi pewną analogię formuły arytmetycznej. Formuła arytmetyczna nadaje wartość pojedynczej zmiennej, podczas gdy formuła operacyjna nadaje wartości układowi zmiennych lub bloków. Formułę operacyjną można traktować jako pewną ilość formuł arytmetycznych, określających wartości zmiennych, stojących na liście wyników.

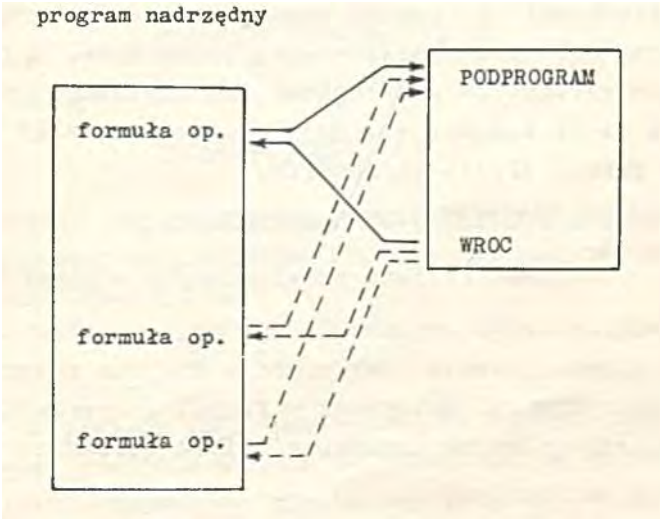
Poniżej dajemy przykład wykorzystania operacji TRANS za pośrednictwem formuły operacyjnej

```
(X, Y) = TRANS(X, 5.6789, BETA)

PODPROGRAM: (U, V) = TRANS(X, Y, ALFA)
```

Kolejność i charakter symboli, stojących na listach formuły operacyjnej, musi być zgodna z kolejnością i charakterem symboli, wymienionych w deklaracji **PODPROGRAM.**

Powrót do programu nadrzędnego zabezpiecza rozkaz **WROC,** który powoduje przejście do wykonywania działań programu nadrzędnego od tego miejsca począwszy, w którym nastąpiło wywołanie podprogramu



Jeśli podprogram definiuje funkcję, wówczas korzystamy z niego za pośrednictwem formuły arytmetycznej według takich samych zasad, jak z funkcji języka. Nazwa podprogramu jest w tym przypadku traktowana jako nazwa funkcji.

Przykład:

Stosowanie formuły arytmetycznej lub operacyjnej nie jest jedynym sposobem podstawienia konkretnych wartości do podprogramu. Część lub nawet wszystkie argumenty mogą być podstawione do pod programu za pośrednictwem rozkazu **PODSTAW.**

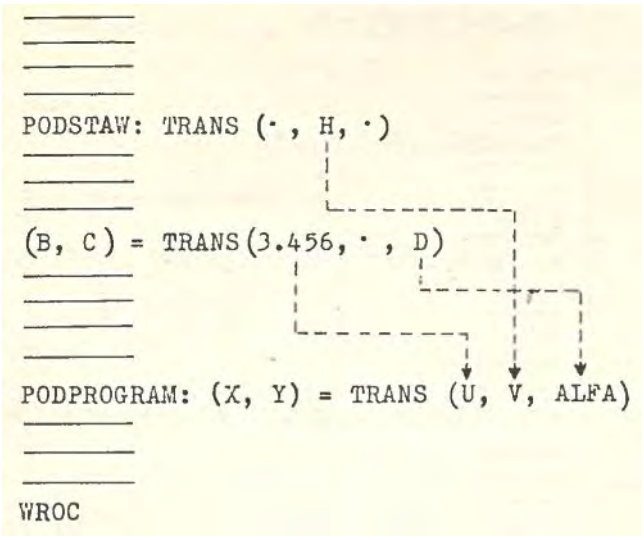
Przyczyną wprowadzenia rozkazu **PODSTAW** jest uniezależnienie procesu podstawiania od procesu wywoływania podprogramu. Jest bowiem niezbędna możliwość podstawiania parametrów, obliczanych przez program główny, do

podprogramu, wywoływanego przez inny pod program, jak ma to miejsce np. przy obliczaniu całki z funkcji, zależnej od pewnej ilości parametrów.

Postać rozkazu **PODSTAW** jest następująca:

PODSTAW: f(lista podstawianych argumentów)

Na liście podstawianych argumentów wypisujemy tylko te argumenty, które ulegają podstawieniu, natomiast w miejsce pozostałych argumentów stawiamy kropki. Natomiast w formule, wywołującej podprogram, wypisujemy tylko te argumenty, które uprzednio nie były podstawiane :



Na powyższym rysunku kreską przerywaną zaznaczono drogi podstawień.

Kropki służą jedynie do określania właściwej kolejności podstawianych argumentów, tzn. ilość kropek na liście, poprzedzających dany argument wyznacza jego kolejność wśród argumentów. Nie jest konieczne wypisywanie kropek za ostatnim podstawianym argumentem.

Zamiast więc pisać:

PODSTAW TRANS (A, •,•)

można pisać krócej:

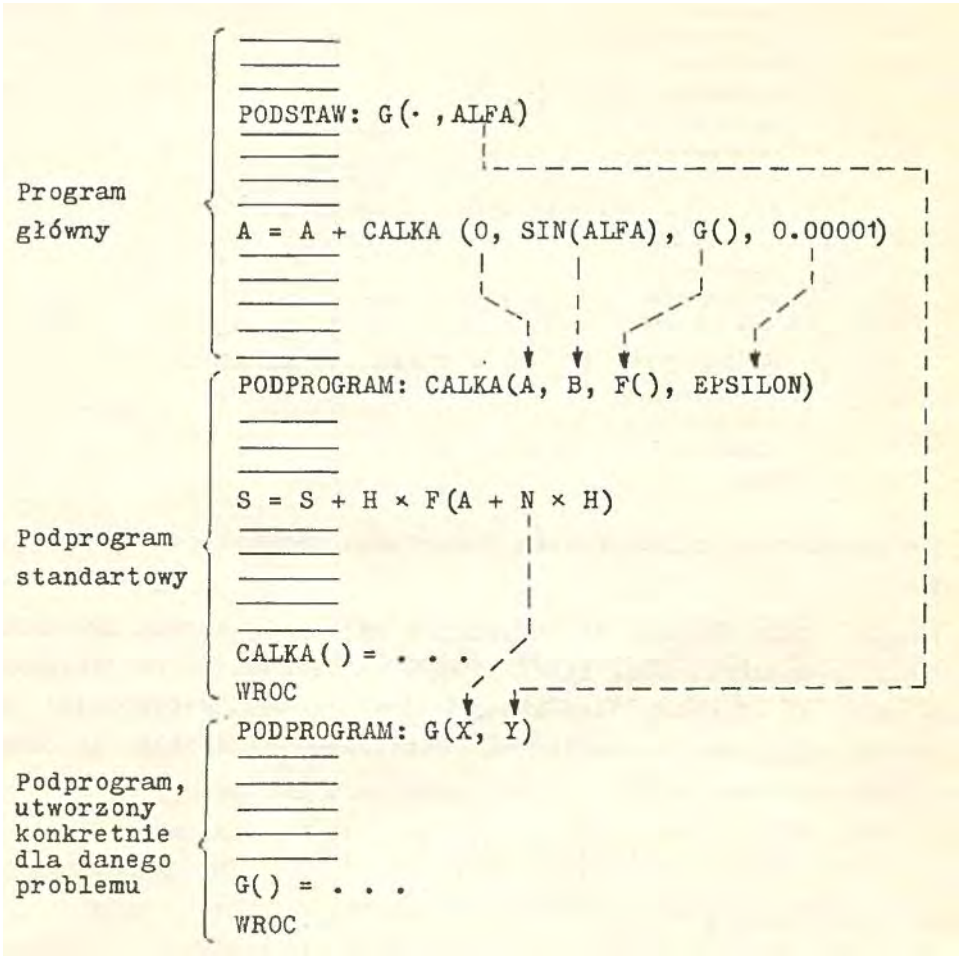
PODSTAW: TRANS(A)

Jeśli wszystkie argumenty podprogramu zostały podstawione przy pomocy rozkazów **PODSTAW**, wówczas formuła operacyjna, wywołująca podprogram, może mieć postać:

(U,V) = TRANS

Możliwość ta posiada duże znaczenie dla korzystania z podprogramów II rzędu, przystosowanych do działania na podprogramach o ustalonej ilości argumentów, gdy chcemy uwzględniać jeszcze pewną dodatkową ilość parametrów.

Pokazuje to następujący przykład:



W powyższym przykładzie CALKA jest funkcjonałem, G()jest funkcją. Podprogram CALKA jest podprogramem II rzędu, podprogram G() jest podprogramem I rzędu. Argumentami podprogramu CALKA są:

- granice całkowania A i B,
- funkcja podcałkowa F(),
- dokładność obliczania całki EPSILON.

Podprogram G() zależy od dwóch argumentów: zmiennej X i parametru Y.

Program główny przy pomocy rozkazu PODSTAW określa parametr Y funkcji G jako równy ALFA. Następnie poprzez formułę arytmetyczną wywołuje podprogram CALKA, określając w nim granice całkowania, funkcję podcałkową oraz żądaną dokładność. /Funkcją podcałkową jest tutaj funkcja G(), nie zaś SIN(ALFA). Dla uniknięcia nieporozumień piszemy zawsze:

F()

gdy chodzi o funkcję, zaś:

F(lista argumentów)

gdy chodzi o wartość funkcji.

Należy zauważyć, że jeden argument funkcji G pozostał nieokreślony - jest nim zmienna całkowania.

W podprogramie CALKA wykorzystujemy podprogram G wywołując go formułą arytmetyczną i przekazując mu pierwszy /dotychczas nie podstawiony/ argument jako równy wartości wyrażenia arytmetycznego A + N x H. W momencie wywołania podprogramu G wszystkie jego argumenty są określone.

Przykład powyższy objaśnia możliwość zmiany ilości parametrów funkcji podcałkowej przez program główny przy wykorzystaniu tylko jednego podprogramu standardowego CALKA. Z przykładu tego widać również, że funkcją podcałkową może być funkcja o dowolnej liczbie argumentów, mimo wykorzystywania tej funkcji przez podprogram również, że funkcją podcałkową może być funkcja o dowolnej liczbie argumentów, mimo wykorzystywania tej funkcji przez podprogram CALKA jako funkcji jednej zmiennej. Jest to możliwe z jednej strony dzięki rozkazowi PODSTAW, z drugiej - dzięki możliwości niewypisania kropek po ostatnim podstawianym argumencie.

System SAKO dopuszcza uogólnienie zasady, zilustrowanej powyższym przykładem, na przypadek podprogramów dowolnie wysokiego rzędu, o dowolnej ilości argumentów, które są również podprogramami dowolnie wysokiego rzędu o dowolnej ilości parametrów.

Parę przykładów napisanych w SAKO dla ZAM-41

```
WYJŚCIE:0=PDW(DW,DDW1);
PROBLEM:SAKO,PROGRAM.
NIWELUJ NADMIARY:TAK
TRANSLACJA
BLOK (2) : A,B
CZYTAJ: * A, *B
P = ILOCZYN (* A, *B)
Q = ILOCZYN(*A, *A)
R = ILOCZYN(*B, *B)
C = P/PWK (QxR)
DRUKUJ (4.6) : C
STOP NASTĘPNY
PODPROGRAM : ILOCZYN(*U, *V)
CAŁKOWITE : I
STRUKTURA (2) : U,V
S=0
*) S=S+U(I)OV(I)
POWTORZ : I=0(1 )2
ILOCZYN ( ) = S
KONIEC

DANE:
-1.2.5 -0-76
#
10. 8.1 -3.01
*
```

```
WYJŚCIE:0=PDW(DW,DDW1);
PROBLEM:SAKO,PROGRAM.
NIWELUJ NADMIARY:TAK
TRANSLACJA
CZYTAJ:K
I1 =CAŁKA(0.0,1 .0,G() )
PODSTAW:H(• ,K)
I2=CALKA(0.5,1.0,H())
LINIA 1
DRUKUJ(3.5)I1,12
STOP NASTĘPNY
PODPROGRAM: CALKA(A,B, F( ) )
DX=(B-A)/8
S=0
*)S=S+F(X)
POWTORZ:X=A(DX)B
CAŁKA ()=(B-A)xS/9
WROC
PODPROGRAM:G(X)
G()=PWK(1-X*3)
WROC
PODPROGRAM:H(X, K)
H=PWK(1-KxSIN(X)*2)
WROC
KONIEC:O
```

Jan Ściegienny - Maszyny ZAM-2 w NRD

Omówiono zastosowania polskich elektronicznych maszyn cyfrowych ZAM-2 w dwóch ośrodkach niemieckich: w energetyce (dyspozycja mocy) oraz w walcowni stali (prace administracyjne i przetwarzanie danych).

W Niemieckiej Republice Demokratycznej już od dłuższego czasu pracują dwie polskie maszyny ZAM-2 Gamma wyprodukowane przez Zakład Doświadczalny Instytutu Maszyn Matematycznych. Pierwsza z nich zainstalowana została w końcu 1963 roku w Niemieckiej Dyspozycji Mocy w Berlinie *(Dispetcherorganisation fur die Elektroenergieversorgung — Berlin).* Utworzono Ośrodek Obliczeniowy, którego personel liczy 20 osób (praca jednozmianowa). Maszyna ZAM-2 znalazła tam zastosowanie do sporządzania codziennego planu pracy poszczególnych bloków energetycznych (zestaw kocioł-turbina) we wszystkich elektrowniach NRD. Zarówno dane, jak i wyniki przesyłane są telegraficzne siecią dalekopisową. Oprócz tej zasadniczej stałej pracy Ośrodek wykonuje też obliczanie rozkładu napięć i rozpływu mocy dla nowo projektowanych systemów energetycznych.

Ze względu na wdrażanie się w prace, jak i rozwagę, z jaką koledzy niemieccy przystępują do stosowania nowego rodzaju techniki, maszyna, pomimo niepełnego obciążenia, nie była dotychczas udostępniana, poza sporadycznymi wypadkami, użytkownikom z zewnątrz. Dopiero obecnie planuje się takie usługi dla resortu budownictwa, w związku z wizytą specjalistów niemieckich w ośrodkach polskich, w której czasie szczególne zainteresowanie wzbudził Ośrodek Obliczeniowy Budownictwa ETOPROJEKT w Warszawie, który posiada maszynę ZAM-2 i opracowuje pokrewną problematykę. Po zainicjowaniu współpracy niektóre programy opracowane w ETOPROJEKCIE zostały przejęte przez ośrodek niemiecki i po przystosowaniu do potrzeb własnych zostaną uruchomione na ZAM-2 znajdującej się w Berlinie. Pomimo zawężonego wykorzystania maszyny w Ośrodku Berlińskim — oszczędności uzyskane dzięki jej zastosowaniu, a skrupulatnie liczone każdego dnia, wynoszą średnio kolo 20 000 marek dziennie. Są to oszczędności powstałe przez zmniejszone zużycie węgla. Biorąc pod uwagę cenę maszyny (około 1 miliona marek) stwierdzić można, że jej koszt dawno już zwrócił się.

Druga z maszyn ZAM-2 zainstalowana została w listopadzie 1964 r. w Walcowni Stali im. Wilhelma Florina w Hennigsdorfie pod Berlinem *(VEB Stahl u. Walzwerk — Hennigsdorf).* Maszyna przeznaczona jest tutaj raczej do prac z dziedziny administracji i przetwarzania danych, przy czym istnieje tendencja włączenia jej do współpracy z istniejącą w walcowni od wielu lat stacją maszyn analitycznych ARITMA. Organizacyjnie maszyna znajduje się w tzw. Centrum Obliczeniowym, podlegającym bezpośrednio dyrektorowi ekonomicznemu zakładu. Dla potrzeb Centrum wybudowano osobny budynek na terenie walcowni. Należy jeszcze dodać, że Centrum jest ośrodkiem wiodącym dla hutnictwa w NRD.

Po okresie przygotowawczym w Hennigsdorfie rozpoczyna się obecnie przejmowanie pewnych konkretnych prac obliczeniowych. Jako pierwsze będą opracowywane przy pomocy maszyny ZAM-2 listy plac. Tytułem próby dubluje się obliczenia wykonywane przez księgowość dla tysięcy robotników. Od kwietnia 1966 r. planuje się objecie automatycznie liczoną listę płac całego personelu walcowni. Należy zaznaczyć, że sporządzanie listy plac netto w takim przedsiębiorstwie jak walcownia jest bardzo pracochłonne, ponieważ place składają się z wielu ruchomych składników. Przewiduje się,

że dzięki zastosowaniu maszyny cyfrowej około 15 pracowników księgowości będzie mogło przejść do innych prac, a czas sporządzania listy płac skróci się do około 10% dawnego czasu opracowania.

W związku ze współpraca maszyny ZAM-2 z maszynami ARITMA, dołączony został na wejściu maszyny czytnik kart ELLIOTT-B 42. Obecnie zdecydowano się przyłączyć reproducer kart na wyjściu maszyny, aby przyspieszyć współpracę z tabulatorami ARTIMA, które w tej organizacji służą jako drukarki formularzy placowych. Dotychczas z taśmy wynikowej przechodziło się na karty za pomocą specjalnych przystawek do dziurkarek kart, co bardzo opóźniło cykl pracy. W opracowaniu znajdują sic jeszcze takie kwestie jak: obliczanie zadań asortymentowych dla wydziałów produkcji podstawowej, przy maksymalnym wykorzystaniu obliczeń ze statystyki dotyczącej wytopów stali w celu ustalenia korelacji między gatunkami stali a kolejnością sadzenia składników, czasu sadzenia i całokształtu zagadnień związanych z prowadzeniem wytopów.

Z innych ciekawszych prac przeprowadzanych przez Centrum Obliczeniowe w Hennigsdorfie można by jeszcze wymienić pracę wykonaną na zlecenie przemysłu chemicznego, w której przeliczono kilkadziesiąt tysięcy wariantów inwestycyjnych. Dla opracowania tego problemu maszyna musiała pracować około 250 godzin, przy czym praca została wykonana w ciągu 12 dób.

Obsługa techniczna i programiści maszyn ZAM-2 znajdujących się w NRD zostali przeszkoleni w Polsce w Zakładzie Doświadczalnym IMM. Wyszkolonych zostało 4 techników-konserwatorów i 6 programistów. Do końca ubiegłego roku przebywali w NRD jeden konserwator i jeden programista ze strony polskiej, którzy nadzorowali i doszkalali obsługę niemiecką.

Użytkownicy z NRD bardzo chwalą sobie ten rodzaj współpracy i podkreślają z zadowoleniem rzetelny serwis zapewniony przez producenta. Jeżeli dodać jeszcze do tego dużą niezawodność pracy naszych maszyn (awaryjność poniżej 5% efektywnego czasu liczenia), to wydaje się zrozumiała dobra atmosfera, jaka wytworzyła się wokół maszyn ZAM-2. Prawdopodobnie w związku z dotychczasowymi doświadczeniami obserwuje się duże zainteresowanie polskimi maszynami cyfrowymi i chęć zakupu w Polsce dalszych maszyn przez rożne resorty kierownictwa gospodarczego NRD.

Władysław Klepacz, Jan Wierzbowski – Zastosowanie ZAM-2 w przedsiębiorstwie ubezpieczeniowym

Artykuł zawiera charakterystykę oraz przebieg projektowania i realizacji zastosowania maszyny typu ZAM-2 do przetwarzania danych w przedsiębiorstwie ubezpieczeniowym. Opisany system znajduje się od kilku lat w eksploatacji użytkowej, w której czasie przetworzono łącznie dane z ok. 400 000 dokumentów. W oparciu o uzyskane doświadczenia autorzy wyprowadzają szereg wniosków o charakterze ogólnym.

Wstęp

W drugiej połowie 1962 roku kilku pracowników Instytutu Maszyn Matematycznych otrzymało wiadomość, że Towarzystwo Ubezpieczeń i Reasekuracji „WARTA" posiada kilkadziesiąt tysięcy dokumentów zawierających liczby, które trzeba szybko zsumować. Oczywiste było, że dla samego zsumowanianawet bardzo dużej ilości liczb — elektroniczna maszyna cyfrowa jest urządzeniem zbyt kosztownym i tym samym nieopłacalnym. Tym niemniej nawiązano kontakt z Towarzystwem „WARTA" celem stwierdzenia rzeczywistych potrzeb obliczeniowych tej instytucji. **I** tak rozpoczęła się współpraca, której wyniki można chyba uznać jako bardzo pozytywne i to nie tylko dla obu zainteresowanych stron, ale również z ogólnego punktu widzenia zastosowania elektronicznej techniki obliczeniowej w Polsce. Opisane fakty przyczynia się mogą również do skorygowania szeregu poglądów na temat krajowych maszyn matematycznych.

Potrzeby obliczeniowe „WARTY"

Bezpośrednie spotkanie z kierownictwem „WARTY" wyjaśniło charakter oraz rozmiary potrzeb obliczeniowych. Polegały one na wykonaniu szczegółowej analizy danych finansowych z paruset tysięcy dokumentów operacyjnych ubiegłych okresów obrachunkowych. Głównym celem tej analizy miało być ustalenie rzeczywistego poziomu tzw. wskaźników szkodowości, tj. stosunków kwot wypłaconych odszkodowań do pobranych składek ubezpieczeniowych wg rożnych kryteriów klasyfikacyjnych. Analiza tego rodzaju jest podstawowym instrumentem zarządzania w każdym przedsiębiorstwie ubezpieczeniowym. Umożliwia ona sprawdzenie prawidłowości pobieranych stawek, zabezpieczających z jednej strony niezbędną rentowność transakcji ubezpieczeniowych, **z** drugiej pełną konkurencyjność „WARTY" w stosunku do zagranicznych towarzystw ubezpieczeniowych. Ten ostatni problem jest szczególnie istotny, ponieważ „WARTA" posiada **w** kra**ju** wyłączność ubezpieczeń w obrocie zagranicznym, a więc całość jej transakcji posiada charakter dewizowy. W związku z dużą dynamiką wzrostu obrotów naszego handlu zagranicznego rośnie również szybko ilość oraz wartość transakcji „WARTY".

Z uwagi na wielką ilość operacji ubezpieczeniowych oraz duże zróżnicowanie pobieranych składek w zależności od rodzaju towaru, zakresu ubezpieczenia, strefy geograficznej, rodzaju środka transportowego oraz rodzaju transakcji handlowej, wspomniana analiza byłaby w ogóle niemożliwa do realizacji metodą obliczeń ręcznych, a bardzo trudna i pracochłonna przy użyciu maszyn analityczno-liczących. Wielkie zróżnicowanie kryteriów analizy (kilka tysięcy) powoduje, że dla uzyskania dostatecznie wiarygodnych §średnich wartości wskaźników

szacowności niezbędne jest oparcie się na dostatecznie dużej liczbie danych, których zebranie wymaga kilku kolejnych lat. Ponieważ największy ciężar gatunkowy w całokształcie działalności „WARTY" posiadają ubezpieczenia transportu towarów czyli tzw. ubezpieczenia CARGO, kierownictwo przedsiębiorstwa wytypowało je do opracowania w pierwszej kolejności. Roczna liczba wystawianych dokumentów wynosi tu ok. 100 000, a więc zakładając do analizy okres co najmniej pięcioletni, problem pod względem ilościowym zakwalifikować do dużych.

Decyzja realizacji problemu

Opisane wyżej motywy w dostatecznie jasny sposób tłumaczą dążenia kierownictwa „WARTY" do przeprowadzenia analizy. Podkreślić należy jednak śmiałość decyzji, a następnie konsekwentną postawę w najtrudniejszych momentach realizacji obliczeń, posiadających niewątpliwie całkowicie eksperymentalny charakter.

Decyzja oraz stanowisko kierownictwa „WARTY" stanowiły bardzo istotny element całej sprawy, nie mogły jednak automatycznie przesądzić o decyzji przyszłych realizatorów systemu. Skierowanie problemu do pracowników Instytutu Maszyn Matematycznych z góry przesądzało o wyborze maszyny matematycznej, jaka może być użyta do obliczeń. W tym czasie tj. w r. 1962, w 1MM eksploatowana była użytkowo jedynie maszyna typu ZAM-2, która, jak wiadomo, nie posiada elementów niezbędnych dla masowego przetwarzania danych, takich jak: duża pamięć zewnętrzna oraz drukarka wierszowa. Tymczasem omawiany problem — jak jut wspomniano wymagał przetworzenia danych z ok. 500 000 dokumentów po ok. 20 cyfr każdy, a więc łącznie ok. 10 000 000 cyfr. Takie rozmiary danych wejściowych, jak również żądanie bardzo rozbudowanych wydawnictw wynikowych kwalifikowały problem w sposób jednoznaczny do opracowania na maszynie posiadającej wspomniane urządzenia zewnętrzne. Maszyna ZAM-2, podobnie jak inne istniejące wówczas w kraju EMC, do badan takich nie była przystosowana.

Tym niemniej w braku odpowiednio wyposażonej maszyny przeprowadzano w IMM już od z góry 1,5 roku liczne eksperymenty na maszynie ZAM-2, które doprowadziły z jednej strony do pewnej rozbudowy urządzeń wejścia-wyjścia (podwójne czytniki oraz dziurkarki taśmy papierowej), z drugiej zaś do opracowania dość skutecznych metod realizacji problemów EPD w warunkach ograniczonych możliwości maszyny cyfrowej. 0 ostatecznej decyzji wykonawców przesadził fakt ukończenia właśnie w IMM z pozytywnym rezultatem problemu przetwarzania materiałów ankiety Biura Studiów i Projektów Komunikacji Miejskiej w Warszawie. Ankieta ta zawierająca ok. 110 000 dokumentów o objętości informacji zbliżonej do dokumentów „WARTY" dowiodła, że rozwiązywanie problemów typu statystycznego nawet przy dużej ilości informacji wejściowych i wynikowych jest w pełni możliwe na maszynie typu ZAM-2, i to nawet przy użyciu tak mało jeszcze w kraju wypróbowanego nośnika informacji masowych, jakim jest pieciokanałowa taśma ,papierowa. Problem analizy ubezpieczeń CARGO był jednak od niego nie tylko większy pod względem ilościowym, ale i bardziej złożony pod względem rachunkowym. Dlatego też 5 lat temu wymagał zarówno od zleceniodawcy, jak i wykonawców dużej wyobraźni, optymizmu, a nawet odwagi dla podjęcia ostatecznej decyzji. Trzeba zaznaczyć, że liczni specjaliści z dziedziny maszyn analityczno-liczących stanowczo odradzali kierownictwu „WARTY" decydowanie się na eksperyment tak niepewny, zarówno z uwagi na rodzaj maszyny, jak i nośnika informacji (taśma dziurkowana).

Pomimo tak niekorzystnej atmosfery, kierownictwo „WARTY” nie wycofało swej decyzji. Po przełamaniu ostatecznych oporów wykonawców w listopadzie 1962 roku przystąpiono do opracowania założeń systemu.

Przygotowanie dokumentów źródłowych

Istniejące w „WARCIE” dokumenty ubezpieczeniowe CARGO z lat ubiegłych świadczyły, że istniały tam już uprzednio próby przeprowadzenia ich analizy. Niektóre z nich zawierały symbolizację cyfrową charakteru transakcji, warunkującą maszynowe opracowanie danych. Zamierzenie to nie zostało jednak doprowadzone do realizacji i tylko niewielka część dokumentów została zasymbolizowana. Już wstępna ocena tej symbolizacji wykazała jednak, że nie będzie ona mogla spełnić ustalonych zadań z uwagi na zbyt szczegółową rozbudowy kryteriów klasyfikacji transakcji, tzn. nadmierną ich ilość w stosunku do istniejącego zbioru dokumentów. Podważało to całkowicie sensowność opracowania analizy, ponieważ wyliczone wskaźniki nie mogły być wartościami średnimi, lecz w przytłaczającej większości całkowicie przypadkowymi (przeciętnie 1-2 przypadków na jedno kryterium klasyfikacji). W takiej sytuacji pierwszą czynnością warunkująca opracowanie systemu musiało być stworzenie nowej koncepcji klucza symbolizacji transakcji. Propozycja projektantów systemu dokonania radykalnego zredukowania ilości kryteriów klasyfikacji natrafiła wewnątrz przedsiębiorstwa na silne opory ze strony zwolenników pierwotnego klucza symbolizacji, którzy postulowali w dalszym ciągu potrzebę uzyskania wskaźników bardzo szczegółowych. Przykładem tego rodzaju podejścia było np. żądanie ustalenia średniej szkodowości występującej przy eksporcie jagód polskimi statkami do Anglii.

Wskutek przedłużającej się dyskusji, wstrzymującej przystąpienie do zaprogramowania systemu, ostateczne ustalenie klucza symbolizacji musiało się zakończyć kompromisem obu stanowisk. Po uzyskaniu pierwszych wyników analizy autorzy systemu uzyskali jednak pełne potwierdzenie dla swego stanowiska ponieważ duża część wyliczonych wskaźników miała wartości zerowe„ Świadczące o tym, że transakcje o bardzo szczegółowych kryteriach charakterystyki w ciągu jednego roku w większości przypadków w ogóle nie występują. Z uwagi na trudności skorygowania lub uzupełnienia symbolizacji dokumentów lat poprzednich, postanowiono zrezygnować z opracowywania tych dokumentów, decydując rozpoczęcie opracowań maszynowych od roku bieżącego,tzn. 1963.

Koncepcja rozwiązania systemu analizy

W otrzymanych dla zaprojektowania systemu wytycznych oprócz głównego celu, jakim miało być uzyskanie omówionych na wstępie wskaźników szkodowości, stanowiących podstawę do weryfikacji stosowanych stawek ubezpieczeniowych i w konsekwencji do skorygowania używanej taryfy, kierownictwo „WARTY” postawiło realizatorom systemu również inne cele, a mianowicie:

- kontrolę przestrzegania przez centrale handlu zagranicznego obowiązku ubezpieczania towarów w „WARCIE” poprzez stworzenie możliwości porównania wartości przewozów ubezpieczonych z odpowiednimi danymi MHZ. Istniejący w przedsiębiorstwie aparat ewidencji nie rejestrował danych w takim układzie;
- analizy szkodowośc, pod kątem rodzaju oraz warunków powstawania szkód do celów akcji prewencyjnych (np. lokalizowanie głównych kierunków kradzieży, pożarów itp.).

Kierownictwo „WARTY” postawiło jeszcze jeden cel, którego realizacji nie mogli jednak zagwarantować wykonawcy systemu. Celem tym była kontrola prawidłowego działania księgowości przedsiębiorstwa. Powstała niemal paradoksalna sytuacja, w której kierownictwo „WARTY” wierząc bezkrytycznie w możliwości automatyzacji było przekonane, że doprowadzenie do zgodności sum końcowych analizy z danymi księgowości jest całkowicie realne do spełnienia, podczas gdy wykonawcy twierdzili, że w problemie o charakterze statystycznym uzyskanie tego rodzaju zgodności jest nie tylko niecelowe; ale wręcz niemożliwe. Oceniając realnie istniejącą sytuację oraz warunki organizacyjne, wykonawcy nie wierzyli w możliwość stworzenia dostatecznego systemu kontroli w fazie przygotowania danych z dokumentów źródłowych. Jak się później okazało, prawda leżała jak zwykle pośrodku.

Po skonkretyzowaniu celów analizy opracowano treść i formę dokumentów wynikowych, które ujęte zostały w 3 rodzaje tabulogramów,, a mianowicie:

1. analityczne zestawienie dochodów i wydatków wg instytucji ubezpieczających się (w tys. $ oraz tys. zł), zawierające sumy wartości ubezpieczenia, składek, wypłaconych odszkodowań i kosztów likwidacji szkód w rozbiciu na rodzaj transakcji (import, eksport, inne) oraz rodzaj środka przewozowego;
2. analityczne zestawienie procentowego stosunku wypłaconych odszkodowań do pobranych składek ubezpieczeniowych w rozbiciu na kraj, rodzaj transakcji, rodzaj środka przewozowego, grupę towarową i zakres ubezpieczenia;
3. analityczne zestawienie odszkodowań (w tys. zł), zawierające wysokość wypłaconego odszkodowania w rozbiciu na kraj, rodzaj transakcji., rodzaj środka przewozowego, grupę towarową oraz rodzaj szkody.

Realizacja systemu

Jednocześnie z decyzją, Ustalającą zasady symbolizacji transakcji na dokumentach, przystąpiono do zaprogramowania systemu oraz określenia sposobu przygotowania dokumentów źródłowych i ich perforowania. Wzory ideowych schematów tych dokumentów podane są na rys. 1 i 2. Dokumenty źródłowe „WARTY” posiadają niejednolitą, bardzo zróżnicowaną treść i formę i tym samym nie *są* przystosowane do systemu zautomatyzowanego. Jedynym wyjątkiem w tym względzie jest klucz statystyczny, który umieszczony jest w specjalnej drukowanej klatce. Tym niemniej i on w dużej ilości przypadków znajduje się w różnych miejscach dokumentów, utrudniając perforację. Natomiast znacznie gorzej przygotowane są do perforacji kwoty finansowe, których potrzeba wydziurkowania zaznaczona jest na dokumentach wyłącznie przez ich podkreślenie czerwonym ołówkiem. Przygotowanie dokumentów źródłowych do obliczeń obejmowało wspomnianą już symbolizację oraz podzielenie ich na tzw. pliki, zawierające średnio ok. 100 dokumentów i zaopatrzone w tzw. karty przewodnie dla celów automatycznej kontroli poprawności perforowania danych. Karty przewodnie zawierały następujące' informacje kontrolne: ilość dokumentów oraz łączne sumy poszczególnych rodzajów kwot finansowych ze wszystkich dokumentów pliku. Specjalny program kontroli, oprócz formalnego badania prawidłowości klucza symbolizacji, sumował ilość dokumentów oraz kwoty finansowe z kolejno wczytywanych przez maszynę dokumentów pliku, a następnie przez porównanie z danymi karty przewodniej umożliwiał skuteczne wykrywanie błędów perforowania.

Programowanie systemu zostało zakończone w r. 1963 i natychmiast przystąpiono do wykonywania obliczeń na maszynie. Pierwsze wyniki, otrzymane w początkach 1964 r. obejmowały zestawienie dochodów i wydatków pogrupowanych wg instytucji ubezpieczających się (central handlu zagranicznego) w poszczególnych oddziałach „WARTY" za rok 1963. Przekazane wyniki zostały poddane szczegółowej analizie przez porównanie z odpowiednimi zapisami w. księgowości „WARTY". Wyniki tego porównania wypadły bardzo korzystnie, zarówno dla samego systemu, jak i maszyny ZAM-2. Okazało się, że dzięki zastosowaniu wspomnianej kontroli danych źródłowych oraz dużej niezawodności pracy maszyny ZAM-2 wszystkie rozbieżności pomiędzy sumarycznymi wynikami z maszyny a zapisami księgowymi dały się wyjaśnić. Wynikały' one z założonego nieuwzględniania przy opracowaniu Maszynowym pewnej grupy nietypowych (niemożliwych do pełnego zasymbolizowania) dokumentów źródłowych. Należy dodać, że przy okazji .uzgodnień udało się stwierdzić i wyeliminować pewne usterki w rachunkowości przedsiębiorstwa, wynikające z omyłkowego zaksięgowania niektórych dokumentów. Opracowanie pozostałych wydawnictw wynikowych za rok 1963 wraz z jednym dodatkowym, którego potrzeba wynikła w międzyczasie, zakończono w listopadzie 1964 roku: Zostały one porównane z treścią wyników pierwszego opracowania i wykazały całkowitą zgodność. Należy dodać, że spływ dokumentów do obliczeń następował z dużymi 'opóźnieniami, a pod koniec obliczeń wynikła konieczność dokonania poważnej korekty (stornowania) danych wskutek omyłkowego dostarczania dokumentów dotyczących następnego okresu obrachunkowego. Biorąc pod uwagę ilość przetworzonych dokumentów (łącznie ze wspomnianym stornowaniem ok. 90 009) oraz nieprzystosowaną do tego rodzaju prac maszynę, otrzymane wyniki stanowiły duży sukces koncepcji organizacyjnej systemu oraz taśmy papierowej jako nośnika informacji masowych.

Modyfikacja systemu

Po całkowitym opracowaniu danych roku 1963 i przekazaniu wyników do wykorzystania, kierownictwo „WARTY" wyraziło zadowolenie z uzyskanych opracowań. Natomiast autorzy systemu, oceniając krytycznie użytkową wartość uzyskanych wyników mieli szereg zarzutów i zastrzeżeń. ,Powstała więc druga paradoksalna sytuacja, w której zleceniodawca jest zadowolony z wykonanej pracy, a wykonawca — nie. Zadowolenie kierownictwa „WARTY" wynikało ze wspomnianej przewagi maszyny matematycznej nad księgowością w zakresie precyzji ewidencjonowania danych, jak również z dostarczenia cennych informacji o przestrzeganiu obowiązku ubezpieczania przez instytucje krajowe, których dotąd nie rejestrowano w przedsiębiorstwie. Natomiast autorzy systemu widzieli w otrzymanych wynikach następujące usterki, dyskwalifikujące zamierzoną wartość użytkową zrealizowanego systemu:

- uzyskane wyniki, a zwłaszcza wskaźniki szkodowości wskutek zbyt dużej ilości kryteriów analizy w większości przypadków miały charakter całkowicie przypadkowych wielkości, potwierdzając w pełni opinię wyrażoną już przed przystąpieniem do zaprojektowania systemu;
- zbyt szczegółowa statystyka była nieczytelna i tym samym niedająca obrazu całości. Powodowała to, że użytkownik gubi/ się w nieistotnych dla niego szczegółach i nie mógł uzyskać spodziewanych korzyści;
- opracowany system programów wskutek założonej i niewykorzystywanej możliwości otrzymania bardzo szczegółowych wyników był zbyt ogólny i przez to nieoptymalny. W rezultacie działał on zbyt wolno i

opracowanie całości materiałów na maszynie było zbyt pracochłonne i tym samym kosztowne.

Sformułowane wyżej zarzuty zostały przekazane w połowie roku 1965 kierownictwu „WARTY” które po skonsultowaniu z rzeczoznawcami z dziedziny ubezpieczeń uznało ich słuszność i zaakceptowało tezę ulepszenia opracowanego systemu. Było to równoznaczne z decyzją dokonania gruntownej modyfikacji wszystkich programów.

Druga koncepcja oraz eksploatacja systemu

W oparciu o uzyskane doświadczenia opracowano w drugiej połowie 1965 r. całkowicie nową koncepcję systemu, skierowaną na zwiększenie walorów użytkowych dokumentów wynikowych. Istota zmian polegała głównie na zwiększeniu ilości opracowań analitycznych (sprawozdań) przy jednoczesnym zmniejszeniu ich szczegółowości. Sprowadziło się to do 10 rodzajów syntetycznych sprawozdań (znacznie czytelniejszych od poprzednich), które podzielić można na trzy grupy:

- sprawozdanie nr 1, zawierające informacje o sumach ubezpieczenia, składkach, odszkodowaniach, kosztach likwidacji szkód, ilościach poszczególnych dokumentów oraz wskaźnikach szkodowości, w podziale na centrale handlu zagranicznego i rodzaj transakcji;
- sprawozdania nr 2-5, zawierające informacje o składkach, odszkodowaniach, ilościach dokumentów i wskaźnikach szkodowości w podziale na rodzaj transakcji oraz w zależności od numeru sprawozdania na: grupę towarową i zakres ubezpieczenia (nr 2), grupę towarową i kraj przeznaczenia (nr 3), kraj ł środek transportowy (nr 4) oraz kraj i zakres ubezpieczenia (nr 5);
- sprawozdanie nr 6-10, zawierające informacje o odszkodowaniach i ilościach dokumentów w podziale na rodzaj transakcji, rodzaj szkody oraz w zależności od numeru sprawozdania na: centralę handlu zagranicznego (nr 6), kraj (nr 7), grupę towarową (nr 8), środek przewozowy (nr 9) i zakres ubezpieczenia (nr 10).

Po zatwierdzeniu powyższej koncepcji przystąpiono do zaprogramowania nowego systemu. Wszystkie programy zostały ukończone w pierwszym kwartale 1966 roku. Na całość systemu składa się łącznie 9 odrębnych programów o sumarycznej objętości ok. 10 000 rozkazów maszyny. Do połowy tego roku przeliczono na maszynie przy użyciu nowych programów powtórnie dokumenty roku 1963 oraz wyperforowano w międzyczasie dokumenty roku .1964 (ponad 92 000 szt). Przy okazji stwierdzona, że wyniki sumaryczne za rok 1963 były całkowicie zgodne ze sprawozdaniami opracowanymi wg pierwszej koncepcji, co było dodatkowym sprawdzianem poprawności działania programów i obliczeń, jak również egzaminem przechowywania taśmy papierowej **z** dużą ilością informacji (ok. 2.5 roku niezbyt starannego przechowywania). Omawiana zmiana 'koncepcji systemu zwiększyła przeszło czterokrotnie wydajność pracy maszyny (z ok. 1000 dok/godz. do ok. 4500 dok/godz.) wyeliminowała pewne dodatkowe czynności i, co najważniejsze, znacznie zwiększyła czytelność, a tym samym użyteczność wyników. Należy podkreślić, że obok wspomnianych bardzo skutecznych metod kontroli danych wejściowych, sprawność działania systemu gwarantuje odpowiednia organizacja obliczeń, która eliminuje w dużym stopniu ryzyko związane z ewentualną awarią maszyny.

Zabezpieczenie polega na przyjęciu zasady jednorazowego opracowania nie większej ilości niż ok. 15 000 dokumentów, mimo że system umożliwia

opracowanie wszystkich dokumentów jednorazowo. Opracowane części zbioru dokumentów są sukcesywnie sumowane w sposób narastający przy pomocy specjalnego programu. System taki umożliwia równocześnie kumulowanie danych z wielu lat. Zastosowana ostrożność postępowania nastawiona jest w zasadzie na ewentualność wystąpienia awarii o charakterze zewnętrznym (np. wyłączenie energii), ponieważ bezpośrednia awaryjność maszyny ZAM-2 jest bardzo niska (1-2% strat na ok. 400 godzin pracy maszyny w ciągu 4 lat eksploatacji systemu).

Podsumowanie

Dotychczasowe opracowanie na maszynie ZAM-2 w ramach systemu „WARTA" łącznej ilości ok. 375 000 dokumentów (dwukrotnie rok 1963 oraz jednokrotnie lata 1964 i 1965) jest — wg posiadanych przez autorów informacji — pierwszą w takim zakresie wykonaną pracą w Polsce na maszynie klasy ZAM-2. System ten jest obecnie konsekwentnie kontynuowany zgodnie z przyjętym założeniem opracowania danych z okresu co najmniej 5 lat. Obecnie zakończono opracowanie roku 1965 (ok. 100 000 dokumentów) i rozpoczyna się opracowanie roku 1966. Wyniki z lat 1963 oraz 1964 wydane zostały w formie broszur o objętości po ok. 150 stron przy użyciu techniki kserograficznej. Przewiduje się kontynuowanie wydawnictwa lat następnych z przeznaczeniem dla zainteresowanych komórek organizacyjnych przedsiębiorstwa. Przewiduje się również opracowanie sumaryczne danych z okresu 5 lat (1963-4967), które powinno spełnić podstawowy cel pracy, a mianowicie stworzenie podstaw do generalnej weryfikacji obecnej taryfy ubezpieczeniowej. Jeśli chodzi o koszty opracowania, to przedstawiają się one w przeliczeniu na 1 dokument źródłowy następująco:

1.	przygotowanie danych łącznie z poprawieniem błędów ok.	0,65 zł
2.	programowa kontrola danych wejściowych na maszynie ok.	0,65 zł
3.	opracowanie analiz na maszynie ZAM-2 ok.	0,30 zł
Razem	ok.	1,60 zł

Z powyższego wynika, że ok. 75% kosztów przetwarzania stanowią wspomniane bardzo pracochłonne czynności przygotowania i kontroli danych oraz usuwania wykrytych błędów. Można przyjąć, że w przypadku, gdyby kontrolę ograniczyć do metod stosowanych przy typowych opracowaniach statystycznych, łączny koszt opracowania można by zredukować do ok. 1 zł za 1 dokument.

Na tle powyższych rezultatów nasuwa się szereg wniosków o charakterze ogólnym. W pierwszym rzędzie należy jeszcze raz podkreślić, że do obliczeń została wykorzystana maszyna ZAM-2, nieprzystosowana do przetwarzania danych. Pozytywne rezultaty, uzyskane przede wszystkim dzięki bardzo dobremu systemowi programowania oraz wysokiej pewności pracy maszyny, świadczą o tym, że maszyna ZAM-2 jest w dalszym ciągu maszyną wysoce użyteczną i może być efektywnie wykorzystywana również do problemów o charakterze opracowań statystycznych. Wbrew niektórym opiniom, maszyna ta przy zachowaniu prawidłowej konserwacji wyróżnia się wspomnianą już wysoką niezawodnością pracy. Autorzy są zdania, że ewentualne zastąpienie w ZAM-2 obecnej pamięci operacyjnej pamięcią ferrytową, bez naruszenia istniejącej organizacji wewnętrznej maszyny, jest jeszcze w chwili obecnej przedsięwzięciem bardzo opłacalnym, ponieważ pozwoli kilkakrotnie zwiększyć jej moc obliczeniową. Przeprowadzona w ten sposób modernizacja nie naruszy istniejących systemów programowania (SAKO, SAS) oraz wieloletniego dorobku programowego, który jak wiadomo decyduje o wartości użytkowej każdej maszyny cyfrowej. Dorobek ten oraz istniejące kadry uzasadniają w pełni

celowość dalszego modernizowania, oraz wykorzystywania maszyn typu ZAM-2. Potwierdzają to również dobitnie fakty wysokiej efektywności wykorzystania tych maszyn w NRD.

Druga sprawa, którą warto podkreślić, to użyty rodzaj nośnika informacji wejściowych. Omówiony przykład obala popularne jeszcze w Polsce mniemanie, że tylko karty perforowane mogą być dobrym nośnikiem informacji wejściowych przy przetwarzaniu danych. Oba rodzaje nośników informacji mogą być w równej mierze stosowane, a wybór jednego z nich zależy od konkretnego zastosowania.

Trzeci wreszcie wniosek dotyczy problemu wykorzystywania wyników maszyny. Występujące powszechnie dążenie użytkownika do uzyskania bardzo szczegółowych dokumentów wynikowych, zwłaszcza przy łatwości ich otrzymywania. (szybkie drukarki), często nie posiada głębszego uzasadnienia i dlatego powinno być przez projektantów SEPD konsekwentnie ograniczane. Praktyka wykazuje, że rozbudowa treści tych dokumentów wskutek nadmiernego ładunku informacji zamiast wyjaśnienia, skutecznie zaciemnia obraz wyników i tym samym zmusza do ręcznego opracowywania syntetycznych wyciągów, co jest oczywistym paradoksem na tle stosowanej techniki. Powyższe wnioski mogą być użyteczne w dużej części również w odniesieniu do innych maszyn i problemów. Potwierdzają to liczne wypowiedzi w czasopismach zagranicznych, ,gdzie obok problemów wielkich opisuje się nadal realizację dużej ilości zagadnień, mniej może efektywnych, ale dających bardziej zbliżony do rzeczywistości obraz praktyki zastosowań elektronicznej techniki obliczeniowej.

Mgr inż. Krzysztof Bytnerowicz

Ukończył Wydział Mechaniczny Technologiczny Politechniki Warszawskiej, specjalizując się w zastosowaniach komputerów, a dokładniej – w organizacji, ekonomice i planowaniu w przemyśle budowy maszyn.

Od 1969 r. Podejmuje pierwszą pracę jako projektant i programista systemów EPD podejmuje w Stołecznym Ośrodku Elektronicznej Techniki Obliczeniowej, programując w językach SAS i SAKO na EMC ZAM-2 Gamma. Następnie jeden z programistów odpowiedzialnych za wdrożenie EMC ZAM-41 w ścisłej współpracy z IMM. Autor podprogramów w SAS dla ZAM-2 i ZAM-41 oraz rozszerzeń systemu SO-141.

Od 1972 r. zatrudniony w Pracowni Systemów Operacyjnych IMM, jeden z programistów odpowiedzialnych za opracowanie, obsługę i wdrożenie ostatnich wersji systemu SO-41 dla EMC ZAM-41. Autor szeregu programów pomocniczych, podprogramów, dekoderów WE/WY i fragmentów SO-141 pisanych w języku PJEG.

Po zakończeniu prac nad ZAM-41 członek zespołu odpowiedzialnego za adaptację i wdrożenia systemu IBM OS/360 do pracy na EMC Jednolitego Systemu. Współautor Technologii Woluminów Dokumentacyjnych (TWD), rozszerzonej adaptacji systemu SMAD dla ZAM-41. Odpowiedzialny za wdrożenie i obsługę, we współpracy z IBM, systemów operacyjnych IBM OS/360 MFT, MVT, OS/VS1 i VM/370 na EMC IBM/370 model 148.

Od 1977 r. po zaprzestaniu prac nad OS/JS w IMM, po porozumieniu pomiędzy ZDO IMM i OBRI przenosi się z częścią zespołu OS/JS do OBRI, aby wdra-

żać OS/JS, zwany Technologiczną Wersją OS, w ośrodkach obliczeniowych sieci ZETO. Prowadzi szkolenia i prezentacje OS/JS i TWD, prezentuje TWD w Akademii Nauk ZSRR i w NRD (Robotron). Wdraża HASP z RTAM na IBM/360 model 50, następnie bierze udział w adaptacji HASP i RTAM do pracy w OS/JS. Współpracuje przy pracach nad RODAN z wykorzystaniem TWD, pomaga w instalowaniu u klientów.

Od 1979 r. zatrudniony w Ośrodku Obliczeniowym Uniwersytetu Warszawskiego, odpowiedzialny za obsługę systemów operacyjnych IBM i JS. Wdraża do użytku, z wykorzystaniem TWD, bibliotekę podprogramów CERNLIB (ponad 1000 podprogramów). Pracuje przy przygotowaniach do instalacji EMC JS/65 produkcji ZSRR we współpracy z producentem.

Od 1980 w MERA-SYSTEM, odpowiedzialny za wdrożenie i obsługę OS/JS, HASP i TWD w ośrodku obliczeniowym. Generuje wersje OS/JS do użytku przez ośrodki obliczeniowe Zjednoczenia MERA. Przygotowuje się do pracy konsultanta w Kuwejcie.

Od 1981 r. programista systemowy w Government Computer Centre w Kuwejcie. Obsługuje systemy operacyjne OS/370 SP, OS/VS2, i VM/370, odpowiada za obsługę JES2. Opracowuje szereg rozszerzeń (exit) JES2, prezentowanych później na SHARE i opublikowanych na taśmie modyfikacji JES2 SHARE. Kontynuuje prace nad TWD. Wykorzystując TWD przeprowadza konwersję z Fortranu CDC na IBM Fortran IV systemu OSSM (Ocean Spill Simulation System system przewidywania rozlewu ropy naftowej na morzu) autorstwa NOAA dla Kuwejckiej Agencji Ochrony Środowiska. Szkoli operatorów systemu.

Jednocześnie od 1982 programista systemowy (pół etatu) w KISR (Kuwejt Institute of Scientific Research). Obsługuje IBM OS/VS1 i szereg podsystemów pracujących pod kontrolą VM/370. Instaluje szereg pakietów graficznych dla obsługi terminali, drukarek i plotterów Tektronix. Autor rozszerzenia (exit) JES/VS dla OS/VS1 obsługującego wydruki OS/VS1 pod kontrolą VM/370. Rozszerzenie to było opublikowane w XEPHON. Autor systemu przygotowania danych i prezentacji wyników obliczeń finansowych wsadu dla OS/VS1 pod kontrolą VM/CMS.

Od 1986 r. Specjalista obsługi oprogramowania EMC produkcji Hitachi (kompatybilnych z IBM/370) w Australii, Nowej Zelandii, Azji i Ameryce Południowej (APLA) w National Advanced Systems, później Hitachi Data Systems. Udział w instalacji tych EMC u klientów, między innymi pierwsza instalacja w Tajlandii. Rozwiązywanie problemów kompatybilności oprogramowania systemowego we współpracy z producentem. Odpowiedzialny za całokształt obsługi ośrodka obliczeniowego pracującego pod kontrolą VM/SP, MVS/SP, MVS/XA wykorzystującego własne EMC AS/6600, później AS/80X3. Odpowiedzialny za telekomunikację z USA, Europą i wewnątrz Regionu Azji/Pacyfiku.

Od 1992 r. przeniesiony do EDS (Electronic Data Systems), nadal obsługując HDS. Odpowiedzialny za konwersje telekomunikacji HDS z własnej sieci do EDSNET. Zapewnienie zdalnego dostępu do sieci z komputerów personalnych z zastosowaniem PCKET/3270 za pomocą opracowanej nakładki napisanej w języku C.

Obsługa wewnętrznych sieci LAN i WAN, instalacja i obsługa oprogramowania LAN firmy Novell. Instalacja i obsługa serwerów LAN. Obsługa telekomunikacji biur HDS w Regionie. Nagradzany kilkakrotnie.

W latach 1999 i 2000 tłumacz ochotnik Olimpiady Sydney'2000. Obsługa wioślarskich zawodów przedolimpijskich (Mistrzostwa Świata) w 1999 r. w języku rosyjskim. W 2000 r. obsługa konkurencji wioślarskich w języku polskim. Autor olimpijskich słowniczków angielsko-polskich dla konkurencji jachtowych, konnych i strzeleckich.

Dwa ciekawe kontrakty między pracami: konwersja systemu rozliczania połączeń telekomunikacji morskiej i satelitarnej z PDP-11 na server OS2/Warp, obsługa routerów Cisco obsługujących australijską sieć Zurich Insurance.

Od 2001 r. po ponad rocznej przerwie, w Computer Associates jako projektant-programista systemów Netmaster (jeden z dwóch, oprócz Netview produkcji IBM) zarządzającego telekomunikacją SNA, X.25 i TCPIP w środowisku EMC IBM i Operations, automatycznego operatora dla tego samego środowiska. Oba te australijskie systemy zostały zakupione przez Computer Associates.

Jeden z trzech programistów odpowiedzialnych za interfejs pomiędzy Systemem Operacyjnym (z/OS) a Netmaster i Operations. Autor podsystemów, modyfikacji, rozszerzeń, emulacji (Netwiew pod Netmaster) programowanych w Assembler, Rexx, wyspecjalizowanych językach (OML, NCL, itp.). Autor aplikacji (Rexx, OML) dekodujących zapamiętane (packet trace) pakiety TCP (IP i UDP). Współpracuje z IBM przy uruchomieniu podsystemu BCPI do użytku. Autor interfejsu Assembler-C-Java umożliwiającego przekazywanie informacji systemu z/OS to stron internetowych programowanych w Java. Kilkakrotnie nagradzany.

Od 2011 r. emeryt na skutek reorganizacji Computer Associates.

Translacja tekstów na język Polski dla paru aplikacji Androida.

POLSKIE TOWARZYSTWO INFORMATYCZNE

Polskie Towarzystwo Informatyczne (PTI) od 1981 r. skupia specjalistów branży teleinformatycznej. Wśród członków Towarzystwa są zarówno informatycy pracujący na uczelniach, w jednostkach administracji publicznej, jak i w biznesie. PTI posiada 14 regionalnych oddziałów na terenie całego kraju. Pasjonaci poszczególnych zagadnień są dodatkowo skupieni w 12 sekcjach tematycznych PTI.

NAJWAŻNIEJSZE OBSZARY DZIAŁAŃ

- przygotowywanie branżowych KONFERENCJI I SEMINARIÓW,
- wydawanie specjalistycznych PUBLIKACJI,
- wspieranie ROZWOJU KOMPETENCJI specjalistów IT,
- promocja WIZERUNKU polskich informatyków,
- przygotowywanie OPINII I EKSPERTYZ przez Izbę Rzeczoznawców PTI,
- CERTYFIKOWANIE umiejętności komputerowych przez Polskie Biuro ECDL działające przy PTI,
- wspieranie EDUKACJI INFORMATYCZNEJ, w tym organizowanie konkursów dla uczniów i studentów,
- przygotowywanie obchodów ŚWIATOWEGO DNIA SPOŁECZEŃSTWA INFORMACYJNEGO.

www.pti.org.pl