informa

OŚRODEK BADAWCZO-ROZWOJOWY INFORMATYKI

Zygmunt Ryznar

ZARYS HISTORII PROGRAMOWANIA ELEKTRONICZNYCH MASZYN CYFROWYCH

PROBLEMY INFORMATYKI

Warszawa 1972

ZYGMUNT RYZNAR

ZARYS HISTORII PROGRAMOWANIA ELEKTRONICZNYCH MASZYN CYFROWYCH

(Na tle rozwoju ich konstrukcji i zastosowań)

Warszawa 1972

Recenzent: Andrzej Idźkiewicz

Redakcja: Franciszek Haratym

Wydawca: Ośrodek Badawczo-Rozwojowy
Informatyki
Działowy Ośrodek Informacji
Warszawa, ul.Marszałkowska 104/122

Druk : Zakład Informacji i Wydawnictw CINTE zam. 340/72 nakł. 850 egz.

SPIS TREŚCI

W S T Ę P

Praca niniejsza jest próbą zwięzłej syntezy dorobku w dziedzinie programowania elektronicznych maszyn cyfrowych na tle postępu technicznego i rozwoju zastosowań. Nie jest ona podręcznikiem programowania ani też rozprawą teoretyczną. Przeznaczona jest w zasadzie dla praktyków (projektantów i programistów) jako lektura pogłębiająca ich zawodową wiedzę ogólną w zakresie elektronicznej techniki obliczeniowej (uwalniająca poniekąd od studiowania obszernej obcojęzycznej literatury specjalistycznej). Będzie przeciwdziałać zbyt wąskiej specjalizacji (polegającej na "przywiązaniu" do jednego typu maszyny, tej zainstalowanej w ośrodku obliczeniowym danego projektanta czy programisty) oraz stworzy pomost do dialogu pomiędzy teoretykami programowania i programistami-praktykami.

Ponadto wydaje się, że praca niniejsza służyć może projektantom jako zbiór wskazówek do wyboru języka programowania, odpowiedniego do określonego rodzaju zastosowań oraz do wyboru typu maszyny.

Podane informacje historyczne oparte są oczywiście na literaturze specjalistycznej, zaś próby analizy porównawczej języków podejmowałem na własne ryzyko.

Zdaje sobie sprawę z tego, że może nie udało mi się ująć wszystkich istotnych faktów historycznych dotyczących rozwoju programowania i uniknąć potknięć. Wiąże się to z trudnościami uchwycenia obfitego dorobku praktycznego i teoretycznego, ponadto rozrzuconego po wielu publikacjach często dla mnie niedostępnych.

Wreszcie wynika to ze zmienności, jakiej z upływem czasu ulegają języki i koncepcje teoretyczne. Ponadto skromna objętość opracowania nie pozwalała na omówienie szeregu spraw.

Odrębne zagadnienia stanowią trudności terminologiczne w sytuacji, kiedy każdy prawie język (i producent maszyn) operuje różnym aparatem pojęciowym, niekiedy jasno nie zdefiniowanym.

Pewną ilustracją kłopotów formalnych mogą być trudności w ustaleniu dat powstania poszczególnych języków. Różne źródła podają różne daty, często bez dostatecznego komentarza. Tymczasem nie bardzo wiadomo, co uważać za datę powstania języka: czy opracowanie pierwszej teoretycznej koncepcji (opisu formalnego) czy też wdrożenia translatora.

Praca niniejsza nie powstała jako rezultat działalności instytucji (naukowej czy innej), jak również nie stanowiła przedmiotu konsultacji czy współpracy. Jest po prostu owocem kilkuletniego hobbystycznego wysiłku jednej osoby.

Będę bardzo zobowiązany i wdzięczny za rzeczową krytykę, uwagi i uzupełnienia, które wzbogacą moją wiedzę zawodową oraz pozwolą na poprawienie materiałów w ewentualnym następnym wydaniu. Proszę o nadsyłanie ich na adres: Kraków, ul.Radzikowskiego 66 m. 112.

Na zakończenie niech mi będzie wolno złożyć podziękowania mojej żonie LUDMILE, bowiem praca niniejsza powstała kosztem naszego czasu rodzinnego.

I. PRZESŁANKI I PIERWSZE KONCEPCJE AUTOMATYCZNEGO LICZENIA

Pierwszą maszyną cyfrową, w której operacje wykonywane były za pomocą układów elektronicznych, był ENIAC (Electronic Numerical Integrator And Computer) zbudowany w Stanach Zjednoczonych pod koniec II wojny światowej.

Dopiero jednak kilka lat później powstały elektroniczne maszyny liczące o nowoczesnej organizacji działania: przechowujące program w pamięci, liczące w dwójkowym systemie, etc.

Podstawą budowy tych maszyn były nie tylko postępy elektroniki i mechaniki, lecz również logiki i teorii automatycznego liczenia, pochodzące niekiedy sprzed kilkuset lat. Maszyny liczące od dawna stanowiły obiekt praktycznych koncepcji i filozoficznych spekulacji. Pierwszymi konstruktorami mechanizmów liczących byli słynni filozofowie i uczeni: Pascal, Leibniz, Łomonosow, Czebyszew. Niezależnie od praktycznych osiągnięć (sumator) Pascal w "Myślach" wypowiada sąd o maszynach arytmetycznych, porównując ich działanie do myślenia żywych istot.

Prawdopodobnie pierwszym autorem częściowo zautomatyzowanej maszyny był Müller, który w 1786 roku przedłożył projekt maszyny do obliczania algebraicznych funkcji różnicowych. Maszyny tej przypuszczalnie nie zbudowano.

Uważa się, że ojcem rachunku automatycznego był natomiast Charles BABBAGE (1792-1871), angielski matematyk kierujący w latach 1828-1839 katedrą matematyki w Cambridge. Zasługi Babbage'a dla rozwoju maszyn liczących były ogromne.

Dlatego też poświęcimy im sporo miejsca. Uczony ten 60 lat swojego życia poświęcił na opracowanie koncepcji i budowę fenomenalnych jak na XIX wiek mechanicznych automatycznych maszyn liczących. Niestety, nie dane mu było siągnąć celu. Pierwsza maszyna licząca na miarę jego koncepcji zbudowana została dopiero pod koniec lat czterdziestych naszego stulecia (1949 - EDSAC). Zbudowano ją z elementów elektronicznych nie istniejących w czasach Babbage'a. Projekt angielskiego uczonego zawierał główne koncepcje współczesnego komputera: pamięć, urządzenie arytmetyczne, wejście z maszynowego nośnika informacji (z kart dziurkowanych), wyjście poprzez urządzenie piszące itp.

Idee Babbage'a ujmowane są następująco. (C - 1):

a/ maszyna licząca powinna wykonywać wszystkie operacje arytmetyczne,

b/ program wprowadza się na kartach dziurkowanych,

c/ maszyna powinna przechowywać rezultaty w celu późniejszego ich wykorzystania,

d/ należy zapewnić możność wyboru pomiędzy działaniami w zależności od wyniku obliczeń.

Babbage stale pracował nad doskonaleniem swych pomysłów. Po 10 latach pracy nad projektem "maszyny różnicowej" w 1823 roku przy dotacji rządowej rozpoczął jej budowę. Po kolejnych 10 latach budowę przerwano z różnorakich powodów (zatarg z inżynierem prowadzącym budowę, trudności finansowe, kłopoty technologiczne, a przede wszystkim pomysł nowej doskonalszej maszyny). Nowa maszyna zwana analityczną (analitical engine) miała wykonywać dowolne obliczenia oraz pracować według innych idei Babbage'a.

Pamięć ("magazyn") tej maszyny składać się miała z 1000 rejestrów po 50 kół cyfrowych w każdym. W zależności od rozkazu każde koło mogło się łączyć z arytmometrem ("fabryką") lub innymi częściami maszyny. Jako ciekawostkę można podać to, że Babbage za największe swoje osiągnięcie uważał nie koncepcję maszyny analitycznej, lecz opracowanie algebry opisującej ruchy poszczególnych części maszyny. Można ten wysiłek uważać za próbę sformułowania teorii działania maszyn liczących.

W tym samym czasie żył również w Anglii jeden z najwybitniejszych logików XIX wieku George Boole (1815-1864). Stworzył on algebrę logiki, znajdującą szerokie zastosowanie w teorii elektronicznych maszyn cyfrowych. Właśnie od Boole'a pochodzi znakowanie operacji alternatywy ("dodawania" logicznego) i koniunkcji ("mnożenia" logicznego). Kontynuatorami algebry Boole'a byli m.in. de Morgan , Porecki i Schröder oraz Jevons. Ten ostatni, oprócz rozpraw teoretycznych posiada w dorobku pierwszą (1869) maszynę do rozwiązywania zadań logicznych. Zastosowanie algebry Boole'a do opisu działań obwodów przełączających zaproponował w 1938 roku Claude Shannon (jak wiadomo, obwody równoległe przedstawiają operację "lub" zaś szeregowe - operację "i").

Duże znaczenie dla rozwoju automatyzacji programowania posiadają prace wybitnego polskiego logika Jana Łukasiewicza (1878-1956), który w 1910 roku pracą "Zasada sprzeczności u Arystotelesa" zapoczątkował logikę matematyczną w Polsce. Uczony ten stworzył w Warszawie ośrodek logistyczny o światowej sławie. Jego koncepcja beznawiasowego zapisu wzorów algebraicznych wykorzystana została do translacji wielu języków programowania. Znana jest w świecie jako tzw. polish notation (polski zapis). Rozróżnia się zapis prosty (direct polish notation) oraz

zapis odwrotny (reverse polish notation). Właśnie ten ostatni używany jest z reguły w technice kompilacji.
Wyrażenie: A x B - C x D w polskiej notacji w zapisie prostym wygląda - x A B x C D. W zapisie prostym znaki działań podawane są przed każdą parą argumentów, zaś w zapisie odwrotnym - po parze (AB x CD x -). Od strony technicznej zapis Łukasiewicza realizowany jest za pomocą tzw. pamięci stosowej (push-down memory), przedstawiającej jak gdyby "stos" rejestrów, z których tylko jeden (górny) może kontaktować się z otoczeniem. Każde wejście-wyjście powoduje więc przemieszczenie się danych w rejestrach.

Po wielu latach zapomnienia idee Babbage'a odżyły. W 1930 roku dr Howard Aiken z uniwersytetu w Harvard opisał model automatycznej maszyny liczącej opartej o XIX-wieczne koncepcje angielskiego matematyka. Opis ten wykorzystano przy budowie maszyny przekaźnikowej MARK I (1937-1944).

Również w latach 30-tych A.M.Turing podał koncepcję abstrakcyjnej maszyny liczącej nazwanej później "maszyną Turinga". W odczycie wygłoszonym w 1936 roku w Londyńskim Towarzystwie Matematycznym przedstawił on teoretyczny model uniwersalnej abstrakcyjnej maszyny liczącej[1/]. Maszyna ta wyposażona była w nieskończenie (z obu stron) długą taśmę z zapisanymi symbolami. Wzdłuż taśmy przesuwała się (!) głowica czytająco-pisząca sprzężona z układem sterującym. Praca maszyny polegała na odczytaniu symbolu z klatki na taśmie, a następnie na porównaniu go z dotych-

---

1/ Dalsze prace nad koncepcją "maszyny Turinga" zostały przerwane przez II wojnę światową. Po wojnie, pracując w National Physical Laboratory, Turing uczestniczył w budowie maszyny lampowej ACE (Automatic Computing Engine). Posiada on również pewne zasługi w pionierskim zastosowaniu koncepcji biblioteki podprogramów, ważnego etapu w rozwoju programowania.

czasowym "swoim" stanem. W przypadku różnic następowało przejście do nowego stanu i obliczenie nowego symbolu, który był zapisywany na taśmie w miejsce symbolu poprzedniego.

Przez usunięcie założenia o nieskończoności taśmy stworzono pojęcie automatu skończonego. Do automatu skończonego wprowadza się jakiś tekst i po odpowiednim przetworzeniu otrzymujemy nowy ciąg symboli. Wykorzystując zasadę "czarnej skrzynki", dokonuje się np. modelowania maszyn matematycznych na innej maszynie.

Największe zasługi dla rozwoju teorii działania nowoczesnych maszyn posiada John von NEUMANN (1903-1957). Ten amerykański matematyk węgierskiego pochodzenia już w 1936 roku, niezależnie od Francuza Couffignala, zaproponował zastosowanie dwójkowego systemu liczenia (zamiast dziesiętnego) w maszynach liczących. Będąc konsultantem w Moore School of Electrical Engineering w Filadelfii, pracował nad ulepszeniem maszyny ENIAC, zbudowanej przez Eckerta i Mauchly'ego. Jak wiadomo, ENIAC posiadał bardzo małą pamięć (dla 20 liczb) i dlatego nie mógł przechowywać programu. Konstruktorzy maszyny zdawali sobie sprawę z tej podstawowej wady i zaproponowali użycie dla następnych maszyn pamięci na ultradźwiękowych liniach opóźniających.

Niemniej jednak wszechstronne badania nad optymalnymi charakterystykami maszyn zostały przeprowadzone dopiero przez Neumanna i jego współpracowników (Goldstine, Burks). Badania te wykazały, że pojemność pamięci powinna wynosić (w owym czasie) co najmniej 4096 słów (o długości 10-12 znaków dziesiętnych), że powinna ona przechowywać zarówno dane jak i program oraz że może wyniknąć potrzeba zastosowania pamięci pomocniczej.

Grupa Neumanna stworzyła podstawy projektowe wielu maszyn (EDVAC,

SEAC, EDSAC i inne) John von Neumann był wszechstronnym uczonym i wniósł duży wkład w rozwój takich dziedzin jak mechanika kwantowa, logika matematyczna i teoria gier. W serii tzw. raportów publikowanych na przestrzeni lat 1945-1947 podał on fundamentalne zasady budowy maszyn matematycznych:

a/ zarówno dane jak i program mogą być wyrażone w systemie binarnym,

b/ program powinien być przechowywany w pamięci,

c/ ponieważ rozkazy są kodowane i przechowywane tak samo jak liczby, można je przetwarzać arytmetycznie w celu otrzymania nowych rozkazów.

Niezależnie od Neumanna, w 1949 roku Wilkes wskazał, że należy dokonywać operacji arytmetycznych na adresach, osiągając przez to znaczne skrócenie wielkości programu (jak wiadomo, modyfikacja adresów pozwala na tworzenie tzw. pętli w programie).

Jak więc widzimy, nie od razu odkrywano oczywiste dzisiaj dla nas zasady działania komputerów. Była to droga ewolucji, którą kiedyś zapoczątkował Charles Babbage a przyspieszył Allan Turing, Howard Aiken i John von Neumann.

II. PIERWSZE KONCEPCJE JĘZYKÓW WYŻSZEGO RZĘDU

Wraz z rozwojem urządzeń pamięciowych i pilną potrzebą ułatwienia programowania pojawiły się koncepcje ujęcia czynności sterujących (operacyjnych) oraz obliczeniowych i logicznych, nie tylko w postaci układów konstrukcyjnych (tj.mikroprogramów) i kodów maszynowych, lecz również za pomocą instrukcji specjalnych wymagających uprzedniego tłumaczenia.

W zależności od sposobu dokonywania tłumaczenia takich instrukcji rozróżniamy systemy interpretacyjne i kompilacyjne. Systemy interpretacyjne stanowią rozwinięcie idei podprogramów. Po rozpoznaniu każdej instrukcji wywoływany jest odpowiedni podprogram i następuje jego wykonanie, po czym pobierana jest następna instrukcja.

W systemach kompilacyjnych cały program źródłowy (source program) jest najpierw tłumaczony na język wewnętrzny (object program), a dopiero później wykonywane są obliczenia w zakresie całego programu.

Pierwsze interpretacyjne programy zastosowano w celu wykonywania operacji w zmiennym przecinku na maszynach pracujących w stałym przecinku oraz symulacji pracy maszyny wieloadresowej na maszynie jednoadresowej. Jednym z pierwszych języków interpretacyjnych był interpretacyjny system kodowania wyrażeń algebraicznych opracowany w latach 1952-1953 przez J.H.Laninga i W.Zieglera w M.I.T. dla maszyny Whirlwind.

Podobne prace dla systemów kompilacyjnych prowadził w Szwajcarii Heinz Rutishauser zatrudniony w Federalnym Instytucie Technologicznym w Zurichu. Już w 1952 roku przedstawił

on metody umożliwiające wprowadzenie do maszyny wyrażeń algebraicznych w zwyczajnej formie. Ponadto Rutishauser zaproponował bardzo prostą i zwięzłą instrukcję do organizacji pętli "for k = /i/ 10".

Ograniczoną próbą zastosowania matematycznej notacji na wejściu był system SHORT CODE zaproponowany w 1949 roku przez J.Mauchley'ego /współtwórcy ENIACa/ a zastosowany w 1952 roku dla maszyn UNIVAC i BINAC.

W 1953 roku dla maszyny IBM 701 opracowano Speed Coding System, w opracowaniu którego uczestniczył J.Backus.

W 1956 roku pojawiają się pierwsze publikacje na temat MANCHESTER UNIVERSITY AUTOCODE. Firma GEC Computers na bazie tego języka opracowała w 1962 roku "autokod" (autocode) dla maszyny FERRANTI MERCURY.

Termin kompilator został wprowadzony przez Dr Grace HOPPER, zatrudnioną w firmie Remington Rand jako szef programowania. Zaproponowała ona użycie języka angielskiego do pisania instrukcji.

Już w 1952 roku opisała działanie kompilatora a następnie kierowała pracami, w wyniku których powstały zapewne pierwsze w świecie kompilatory A0 i Ai. w 1955 roku powstaje wersja A2, a następnie AT3 (nazwana ARITH-MATIC). Pierwszy opis tego języka znany był w 1957 roku. W tym samym roku pojawił się opis języka FLOW-MATIC, ukierunkowanego na zastosowanie ekonomiczno-administracyjne.

FLOW-MATIC, zwany początkowo jako język B-0, charakteryzował się szerokim użyciem języka angielskiego zarówno do pisania instrukcji jak i do opisu danych. Język ten stanowił

później jedną z podstaw języka COBOL (1959). Na poparcie tego twierdzenia przytoczymy parę instrukcji języka FLOW-MATIC, prawie żywcem wziętych do COBOLu :

ADD A B C, DIVIDE nazwa-danych -i BY nazwa-danych-2 GIVING nazwa-danych-3, MOVE n-d-i TO n-d-2, READ ITEM n-d IF END OF DATA GO TO OPERATION.....

Ważnym problemem programowania jest algorytmizacja zadania, tj. jednoznaczne określenie reguł działania procesu obliczeniowego, uwzględniające wszystkie możliwe kombinacje i etapy. Przy okazji warto zaznaczyć, że słowo "algorytm" pochodzi od nazwiska żyjącego w IX wieku uzbeckiego matematyka AL-HOREZMI (T - 3).

Na lata 1952-1953 przypadają początki opracowania w ZSRR (w Matematycznym Instytucie AN ZSRR) pod kierownictwem A.A. Liapunowa - tzw.operatorowego opisu algorytmów. U podstaw metody leży założenie, że algorytm składa się z szeregu powtarzalnych arytmetycznych i logicznych etapów. Każdy z nich jest realizowany za pomocą tzw. operatora, reprezentującego grupę elementarnych działań, wykonanie których może przebiegać automatycznie w powiązaniu z tzw. programującym programem. Program zewnętrzny (pisany przez programistę) zawiera takie informacje jak:

a/ typ operatora (A- arytmetyczny, P - logiczny, F/i/ - przeadresowanie, E /m;n/ - skok do operatorów z numerami m,m-1, ...n) itp. nr porządkowy operatora, kierunki skoków,

b/ wzory matematyczne dla operatorów arytmetycznych,

c/ parametry przeadresowania,

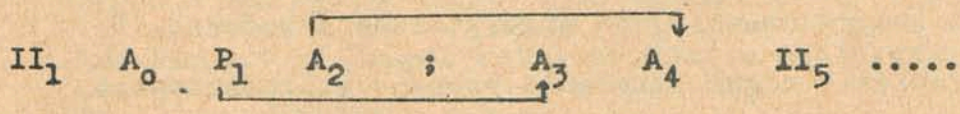
d/ podprogramy,

itp.

Są to informacje o operatorach standardowych. Operatory niestan-

dardowe muszą być całkowicie zaprogramowane w języku wewnętrznym. Główne czynności sprowadzają się do wyboru metody obliczeń, sporządzenia logicznego schematu programu (przez wypisanie ciągu operatorów), określeniu wzorów i parametrów oraz zaprogramowania niestandardowych fragmentów programów.

Oto przykład logicznego schematu w metodzie operatorowej:



Programujący program opracowano m.in. dla maszyny Strieła-4.

Opracowanie języka programowania wymaga zdefiniowania szeregu takich elementów, jak: alfabet (wykaz dozwolonych znaków), wyrażenie (instrukcja) i zdanie (zespół instrukcji zakończony znakiem końca, np. kropka), deklaracje (np. dotyczące opisu danych), dyrektywy operacyjne (związane z organizacją translacji), reguły tworzenia procedur (bloków, paragrafów), reguły wzywania podprogramów i umieszczania wstawek w innych językach itp.

Potrzebę ograniczonego słownika znaków sygnalizował już Turing w koncepcji swojej abstrakcyjnej maszyny. Ze znaków tych proponował tworzenie dowolnych kombinacji.

Problemy konstrukcji, interpretacji i stosowalności języka ujmowane są jako składnia, semantyka i pragmatyka.
Składnia określa, jakie sekwencje znaków są dopuszczalne (np. w postaci instrukcji), przykładowo A + B może stanowić prawidłową konstrukcję w języku "XX", podczas gdy +AB jest nielegalne. Semantyka obejmuje reguły określania znaczenia dopuszczalnych kombinacji, zaś pragmatyka - reguły stosowalności zarówno seman-

tyki jak i składni.

Wszystkie trzy powyższe działy powinny być opisane w sposób jednoznaczny, a jest to możliwe dzięki zastosowaniu metod sformalizowanego zapisu. Tak się złożyło, że dotychczasowe badania dotyczyły głównie składni, zaś semantyka i pragmatyka czekają dopiero na opracowanie, a właściwie stworzenie.

Badania nad składnią zapoczątkował w 1956 roku Chomsky. Zdefiniował on podstawowe elementy pragmatyki formalnej:

a/ podzbiór terminalny, tj. słownik zwrotów języka (małe litery),

b/ słownik nazw strukturalnych, tj. zmiennych (duże litery),

c/ produkcje tj. prawa podstawiania.

Odnośnie tego ostatniego elementu warto dodać, że pojęcie podstawienia, zdefiniowane przez Posta i Thuego, stanowi kluczowy kanon w metodach zapisu składni.

Duże zasługi w rozwoju notacji posiada BACKUS. W notacji (1959) swojej zmienił on znak podstawienia Chomskyego z $\rightarrow$ na : : =, zaś wprowadzenie nawiasów $\langle , \rangle$ pozwoliło na zamianę dużych liter przez opisowe słowa lub frazy.

Notacje Backusa wykorzystano w ALGOL REPORT /1960/ zredagowanym przez Naura. Stąd też pochodzi skrótowa nazwa notacji BNF (Backus Normal Form lub Backus Naur Form). Rozwinięciem zapisu BNF jest metoda van Wijngaardena, przewidująca nieskończoną liczbę produkcji. Niekiedy spotkać można oznaczenie PNF (Paniani Backus

Form). Paniani był nauczycielem perskim, żyjącym ok. 600 lat p.n.e., który opracował system notacji zbliżony do formy BNF (B - 4).

K.E.Iverson zaproponował (1964) notację w pewnym sensie pośrednią pomiędzy "rozwlekłym" zapisem Backusa i "anonimowym" zapisem Chomsky'ego. Notacja Iversona umożliwia precyzyjny opis złożonych procesów sekwencyjnych. Oto jej przykłady:

a/ operacje arytmetyczne: $+ - \times \div$

b/ operacje logiczne: "I" $W \leftarrow U \wedge V$

"LUB" $W \leftarrow U \vee V$

C/ inne: "przesuń na lewo X o K miejsc/ $Z \leftarrow K \uparrow X$

"przesuń na prawo X o K miejsc/ $Z \leftarrow K \downarrow X$

W ośrodku IBM w Wiedniu opracowano (publ. 1968) tzw. wiedeńską metodę opisu języków programowania. Użyto jej m.in. do opisu języków PL/I i ALGOL 60. W metodzie tej zastosowano aparat formalny tzw. obiektów abstrakcyjnych. Do opisu struktury syntaktycznej oraz procesów używa się grafów (w postaci drzew). Każdy wierzchołek grafu procesu związany jest z instrukcją. Wykonanie instrukcji powoduje usunięcie związanego z nią wierzchołka drzewa sterowania.

Również o graficzną metodę przedstawiania programu opiera się metoda Floyda. Stosuje się w niej schematy blokowe, składające się z bloków operacyjnych i warunkowych, połączonych strzałkami. Dla każdej strzałki w sposób formalny (stosując specjalrą notację) podaje się warunki dotyczące zmiennych. Program sprawdzany jest przez przypisanie zmiennym aktualnych wartcści i porównanie ich z warunkami. Metoda ta stanowi więc pewną analogię do znanej metody sprawdzania programów przez tzw. suche przebiegi. Opis metody Floyda podany jest w publikacji A.Mazurkiewicza /M - 2/.

Pierwszy system oznaczeń graficznych (schematów blokowych) został zaproponowany przez Burksa, Goldstine'a i von Neumanna, przy czym był on wygodny dla maszyn jednoadresowych, podobnych do tych, które zostały zaprojektowane w Institute of Advanced Study. Mimo, iż schematy te były ukierunkowane na konkretne maszyny, zapoczątkowały one metodę uniwersalną, pozwalając na dokładne wyrażenie logiki programów niezależnie od rodzaju maszyny. Jest to więc pewne przybliżenie do języka uniwersalnego.

III. PRZYCZYNY PRAC NAD AUTOMATYZACJĄ PROGRAMOWANIA ORAZ TRUDNOŚCI WDRAŻANIA JĘZYKÓW PROGRAMOWANIA

Do dziś jeszcze spotykamy przeciwników posługiwania się autokodami. Przy okazji wyjaśniamy, że przez "autokod" rozumiemy tutaj język AUTOmatycznego KODowania, tj. taki język, który na podstawie jednej zewnętrznej instrukcji generuje sekwencję rozkazów w języku wewnętrznym. Tak więc definicja ta nie obejmuje języków symbolicznych o współczynniku kodowania "jeden do jeden", jako że stanowią one po prostu zmodyfikowaną wersję języka wewnętrznego (przykładem takiego języka jest w zasadzie SSK - Sistiema Simwoliczeskowo Kodirowanija - dla emc Mińsk 32). Wyraz "autokod" pochodzi prawdopodobnie od nazwy jednego z pierwszych języków MANCHESTER UNIVERSITY AUTOCODE, opracowanego w II połowie lat 50-tych w Wielkiej Brytanii. Często, naszym zdaniem, niesłusznie pojęcie autokodu utożsamiane jest wyłącznie z językami symbolicznymi, co zostało być może spowodowane nazwą IBM-owskiego języka symbolicznego AUTOCODER.

Podobnie, jak przy każdym innym rodzaju postępu, wprowadzanie

Pierwszy system oznaczeń graficznych (schematów blokowych) został zaproponowany przez Burksa, Goldstine'a i von Neumanna, przy czym był on wygodny dla maszyn jednoadresowych, podobnych do tych, które zostały zaprojektowane w Institute of Advanced Study. Mimo, iż schematy te były ukierunkowane na konkretne maszyny, zapoczątkowały one metodę uniwersalną, pozwalając na dokładne wyrażenie logiki programów niezależnie od rodzaju maszyny. Jest to więc pewne przybliżenie do języka uniwersalnego.

III. PRZYCZYNY PRAC NAD AUTOMATYZACJĄ PROGRAMOWANIA ORAZ TRUDNOŚCI WDRAŻANIA JĘZYKÓW PROGRAMOWANIA

Do dziś jeszcze spotykamy przeciwników posługiwania się autokodami. Przy okazji wyjaśniamy, że przez "autokod" rozumiemy tutaj język AUTOmatycznego KODowania, tj. taki język, który na podstawie jednej zewnętrznej instrukcji generuje sekwencję rozkazów w języku wewnętrznym. Tak więc definicja ta nie obejmuje języków symbolicznych o współczynniku kodowania "jeden do jeden", jako że stanowią one po prostu zmodyfikowaną wersję języka wewnętrznego (przykładem takiego języka jest w zasadzie SSK - Sistiema Simwoliczeskowo Kodirowanija - dla emc Mińsk 32). Wyraz "autokod" pochodzi prawdopodobnie od nazwy jednego z pierwszych języków MANCHESTER UNIVERSITY AUTOCODE, opracowanego w II połowie lat 50-tych w Wielkiej Brytanii. Często, naszym zdaniem, niesłusznie pojęcie autokodu utożsamiane jest wyłącznie z językami symbolicznymi, co zostało być może spowodowane nazwą IBM-owskiego języka symbolicznego AUTOCODER.

Podobnie, jak przy każdym innym rodzaju postępu, wprowadzanie

autokodów natrafiało na duże trudności, noszące przede wszystkim charakter psychologiczny.
Obecnie pojęcie "Autokod" traktowane jest w literaturze jako archaizm, użyty do nazwania pierwszych eksperymentalnych języków z FORTRANEM I włącznie, w których - dzięki zastosowaniu notacji zbliżonej do matematycznej wyeliminowano współczynnik kodowania 1:1 oraz pozostawiono translatorom sprawę alokacji pamięci.

Już w 1953 roku, a więc wtedy, kiedy przeprowadzono dopiero pierwsze eksperymenty nad automatyzacją programowania, na jednym z pierwszych posiedzeń ACM (Association for Computing Machinery) omawiano sprawę tzw. "prymitywistów" (zwolenników języka wewnętrznego) oraz "żołnierzy postępu" (space cadets).
Ci ostatni posądzani byli o propagowanie luksusu programowania wg "śmiesznych" schematów ograniczających myślenie.

Zarówno jedni jak i drudzy dysponowali pewnymi argumentami. Również i dzisiaj sprawa poziomu programowania nie jest zupełnie (przynajmniej dla niektórych) wolna od nieporozumień. Wydaje się jednak, że sama praktyka (popularność zewnętrznych języków programowania) spór ten rozstrzygnęła.

Najpoważniejszym argumentem przeciwko autokodom (a w szczególności przeciwko językom wyższego rzędu typu ALGOL, COBOL) jest mała efektywność programu wewnętrznego, jaki generują, oraz większe zapotrzebowanie na pamięć operacyjną. Argumenty te, aczkolwiek całkiem słuszne (jeśli poniekąd rozpatrywać je abstrakcyjnie), nie przekonywują, jeśli zważymy, że w ostatnim okresie nastąpił ogromny wzrost szybkości działania maszyny i powiększenie pojemności pamięci operacyjnej, wspomaganej przez pamięci

pomocnicze o dostępie wyrywkowym. Opieranie się na porównaniach pracy małych maszyn (dla których autokody są rzeczywiście mało efektywne) jest nie do przyjęcia, zważywszy, że produkcja ich należy do przeszłości (mamy na myśli maszyny o szybkości kilkuset operacji czy kilku tysięcy operacji na sekundę oraz wyposażenie w 4-8K pamięci operacyjnej). Straty czasu tak znaczne przy kompilatorach lat 60-tych (X - i) zostały znacznie (niekiedy kilkakrotnie) zmniejszone.

Niemniej jednak w pewnych sytuacjach wybór elementarnego języka programowania można uznać za słuszny. W szczególności dotyczy to programów standardowych czy też translatorów (niekoniecznie) oraz niektórych programów obliczeniowych o dużej częstotliwości przetwarzania (np.codziennie) i operujących na dużej ilości danych.

Porównując efektywność języków należy wziąć pod uwagę kwalifikacje programistów i jakość translatorów, jako elementy zmienne przy każdym porównaniu. Spotyka się opinie, że program przetłumaczony z języka wyższego rzędu (przez dobry kompilator) dorównuje klasą programowi w języku elementarnym napisanym przez programistę średniej klasy.

Szczególny opór, np. w stosunku do COBOLu, przejawiają programiści, którzy uprzednio długo programowali w języku symbolicznym lub wewnętrznym. Przestawienie się na inne sposoby programowania stanowi dla nich trudną barierę.

W poniższej tabeli przedstawimy zależność kosztu i efektywności od poziomu języka (wg "Management Standards for Data Processing" D.H.Brandom, 1963):

		I poziom język symb. 1:1	II poziom język symbolicz. + makro-instr.	III poziom typ COBOL ALGOL	IV poziom generatory
Dobry programista	efektywność programu w jęz.wewn.	95%	90%	60-85%	45-65%
	koszt (indeks)	100	90	40-60	20-40
Zły programista	efektywność	70%	75%	50-70%	40-60%
	koszt	110	100	50-70	30-50

Wg tego samego źródła, struktura zastosowań poszczególnych poziomów języków przedstawia się następująco:

Poziom języka	1962	przewid. 1972
1. Symboliczny 1 : 1	65%	25%
2. Symboliczny z makroinstr.	20%	15%
3. Kompilatory	5%	40%
4. Generatory	10%	20%

Inne późniejsze (1969) badania amerykańskie (B - 5) wykazały, że w zastosowaniach administracyjno-ekonomicznych (poza podprogramami standardowymi) język wyższego rzędu np. COBOL (lub inny do przetwarzania danych) dorównuje językom symbolicznym.

Przy porównywaniu języków programowania należy stosować nie jedno kryterium, lecz brać pod uwagę cały zespół czynników:

1. Czas przejścia od "problemu" do programu, czyli uczenie się języka oraz zdefiniowanie algorytmu odpowiednio do wymogów języka. Jest to czynnik, na który bardzo rzadko zwraca się uwagę.
2. Czas pisania programów.
3. Czas uruchomienia programów.
4. Efektywność translatora:
   - czas translacji,
   - pojemność pamięci operacyjnej niezbędna dla translacji,
   - efektywność programu w języku wewnętrznym (po translacji) którą można określić jako ilość rozkazów wewnętrznych, pojemność pamięci koniecznej do przechowywania programu i danych oraz czas biegu programu (czyli czas przetwarzania).

Zalety języków wyższego rzędu ująć można następująco:

1/ pracochłonność programowania jest co najmniej kilkakrotnie niższa,

2/ języki te z reguły są łatwiejsze do opanowania (m.in. ze względu na użycie słów naturalnego języka),

3/ łatwiej i krócej uruchamia się programy (krótszy program zewnętrzny, łatwiejsza diagnostyka błędów, mniej pomyłek programisty, łatwiejsze korekty itp.),

4/ program stanowi sam w sobie nieźłą dokumentację (odnosi się to szczególnie do COBOLu), co nie jest bez znaczenia dla organizacji podziału pracy, fluktuacji kadr, wprowadzania zmian do programu oraz wymiany doświadczeń,

5/ względna niezależność programu (napisanego w języku powszechnego użycia: COBOL, FORTRAN, ALGOL) od typu maszyny, co umożliwia użytkownikowi zachowanie ciągłości przetwarzania przy zmianie typu maszyny (po dokonaniu pewnych przeróbek nie zmieniających jednak podstawowych rozwiązań programowych) zaś programiście - zachowanie poprzednich kwalifikacji,

6/ logika złożonego programu może zostać przedstawiona stosunkowo prosto (syntetycznie), co ułatwia tzw. suche (przy biurku) sprawdzanie biegu programu,

7/ programista może bardziej koncenrtować się na treści problemu, dla którego należy opracować program, niż na technice programowania i właściwościach konstrukcyjnych maszyny; dlatego też możliwe jest posiadanie dobrych analityków, będących równocześnie dobrymi programistami w COBOLu,

8/ języki wyższego rzędu są niezastąpione w systemach konwersacyjnych człowiek-maszyna typu pytanie-odpowiedź, gdzie chodzi o łatwy i szybki dostęp do maszyny dla nieprogramistów.

Niektóre z wyżej podanych zalet stają się bardzo ważne w sytuacji ostrego deficytu kadr specjalistów oraz postępu technicznego (zmiany maszyn,a więc i języków maszynowo zorientowanych). Ponadto języki wyższego rzędu są szczególnie przydatne w przypadku konieczności szybkiego oprogramowania systemu i przy tworzeniu tzw. pierwszych wersji programów (mających na celu sprawdzenie ich logiki) wyprzedzających prace długofalowe nad optymalnymi pakietowymi rozwiązaniami.

Jeśli mimo tych zalet, występowały duże trudności wdrażania języków wyższego rzędu, to wynikało to nie tyle z oporu użytkowników ile niechęci producentów. Można twierdzić, że właściwie żaden język nie uzyskiwał "prawa obywatelstwa" bez walki.

Oto przykłady:

Firmowy język IBM - FORTRAN opracowany w latach 1954 - 1955 (niewątpliwie jeden z najstarszych języków programowania wśród obecnie stosowanych) aż do 1961 roku był ignorowany przez innych producentów (1961 - pierwsza wersja FORTRANu I dla UNIVAC Solid State 80), zaś późniejsza akceptacja języka powodowana była głównie chęcią przejęcia klientów od IBM.

Z kolei firma IBM usiłowała zbojkotować język COBOL; mimo własnego udziału w pracach nad jego opracowaniem. Twierdząc, że język ten jest niedostatecznie zdefiniowany (co było częściowo prawdą), firma ta opracowała własną kontrpropozycję: języki COMMERCIAL TRANSLATOR i AUTOCODER. Zważywszy, że w owym czasie 80% ogółu użytkowników maszyn było klientami IBM, sprawa przybrała poważny obrót.

Nie tylko zresztą IBM występował przeciwko COBOLowi. Firma NCR pozostawała przy języku NEAT (National Electronic Autocoding Techniques), zaś Honeywell zaproponował inny język do przetwarzania danych FACT (Fully Automatic Compiling Technique), który pod względem swoich możliwości niekiedy przewyższał COBOL.

W pierwszym okresie istnienia COBOLu (1960) jedynie firmy RCA i Remington Rand opracowały translatory dla tego języka.

Chcąc pokonać opór producentów rząd USA ogłosił w 1964 roku blokadę zakupów (dla rządowych instytucji) maszyn bez translatorów języka COBOL i ogłosił politykę konkursowych zakupów maszyn wyposażonych w COBOL (chodziło o zakup priorytetowy maszyn z najlepszymi translatorami). Ponieważ zakupy rządowe były dosyć znaczne, np. tylko w 1963 roku US Air Force zakupił 500 maszyn, taktyka ta przyniosła pełny sukces.

Podwójną grę w stosunku do COBOLu początkowo prowadziła

firma HONEYWELL. Mimo, iż przedstawiciele tej firmy wchodzili w skład zespołów roboczych COBOLu, utrzymywała ona w tajemnicy prace prowadzone nad językiem FACT. Pierwszy opis tego języka opublikowany na jesieni 1959 roku był kompletnym zaskoczeniem dla Komitetu COBOLu. Pod wpływem presji rządu firma ta zaakceptowała jednak COBOL, a nawet rozpoczęła prace nad rozwojem tego języka (dochodząc do bardzo dobrych translatorów). Język FACT został wdrożony jedynie na maszynie H 800.

Mimo początkowych trudności wdrożeniowych stosunkowo szybko przystąpiono do opracowywania różnego rodzaju języków automatycznego programowania (czy też raczej należałoby użyć określenia - kodowania) i w krótce doszło nawet do wyraźnej ich nadprodukcji. Języki tworzono bez dostatecznej podstawy teoretycznej i często do użytku ad hoc.

Na początku 1963 roku występowało około 290 języków, z tego:

a/ 74 zorientowanych maszynowo,

b/ 56 przestarzałych (wycofanych),

c/ 108 do obliczeń matematycznych,

d/ 16 do zastosowań administracyjno-ekonomicznych,

e/ 6 do zastosowań algebraiczno-ekonomicznych,

f/ 7 do symulacji procesów technologicznych.

W owym czasie istniało 39 translatorów języka COBOL.

W 1966 roku naliczono już 1200 języków automatycznego programowania, nie licząc podzbiorów ALGOLu, COBOLu itp. Pojawił się również uniwersalny język programowania PL/I.

IV. KLASYFIKACJA JĘZYKÓW PROGRAMOWANIA

Do klasyfikowania języków programowania używać można wielu kryteriów. Niektóre z nich podano w niniejszym opracowaniu. I tak można te języki klasyfikować na języki:

1. wg techniki tłumaczenia -
   a/ interpretujące (interpretatory, interpretery, ang. interpreters),
   b/ kompilujące (kompilatory, kompilery, ang. compilers),
2. wg poziomu programowania -
   a/ wewnętrzne (maszynowe),
   b/ mnemoniczne operujące na adresach cyfrowych i stosujące symbole literowe do oznaczania kodu operacji,
   c/ mnemoniczne pełne (symboliczne, ang. assembly languages) posiadające symbole literowe operacji i argumentów,
   d/ symboliczne z makrorozkazami (o współczynniku tłumaczenia np. 4:1),
   e/ wyższego rzędu, tj. nie zorientowane maszynowo.
3. wg funkcji -
   a/ do przetwarzania informacji,
   b/ do obsługi przetwarzania (systemy operacyjne),
   c/ do pisania translatorów,
   d/ do opisywania języków (metajęzyki),
   e/ do innych celów,

(Powyższe grupy klasyfikacyjne niekiedy zachodzą na siebie. W szczególności dotyczy to grup a, c, d).

4. wg pojęcia proceduralności -charakteryzujące się tym, że programista podaje kolejność procedur czy rozkazów

(do tego typu należą więc prawie wszystkie języki),

a/ języki nieproceduralne tj. takie, w których programista podaje jedynie specyfikacje wejścia-wyjścia oraz wzory obliczeniowe, bez precyzowania sekwencji obliczeń (przykładem takiego języka może być REPORT PROGRAM GENERATOR).

Ze względu na ograniczoną objętość niniejszej pracy nie możemy omówić szczegółowo poszczególnych układów klasyfikacyjnych. Uwagę skoncentrujemy na językach do przetwarzania informacji.

ad 1. INTERPRETER charakteryzuje się tym, że natychmiast po dokonaniu tłumaczenia najmniejszej jednostki znaczeniowej programu (instrukcji) jest ona wykonywana, czyli otrzymujemy wynik. Jest to niewątpliwie zaleta z punktu widzenia użytkownika, wada zaś polega na tym, że trzeba dokonywać wielokrotnego tłumaczenia tych samych wyrażeń, np. przy obliczeniach cyklicznych.

KOMPILER tłumaczy najpierw cały program, a później dopiero po tzw. kompozycji (konsolidacji) program staje się gotowy do realizacji. Wprowadzenie zmian do programu pociąga za sobą konieczność rekompilacji.

Systemy interpretacyjne stosowane są z reguły w maszynach z małą pamięcią i dużą szybkością obliczeń.

Istnieją języki łączące w sobie zarówno cechy interpretatora jak i kompilatora (np. QUIKTRAN). Większość języków pracuje w systemach kompilacji. Jako przykład specjalnego systemu interpretacyjnego wymienić można systemy operacyjne.

Do systemu interpretacyjnego należą m.in. języki: MUMPS (MGH Utility Multi-Programming System) opracowany

na maszynę PDP-9 pod kątem wykorzystania wieloprogramowości oraz GOTRAN, stanowiący wersję FORTRANu na IBM 1620. Język MUMPS wykorzystuje zaletę interpretatora polegającą na tym, że programy są krótkie, i w związku z tym nawet przy niedużej maszynie można w pamięci przechowywać równocześnie kilka programów. Pozwala to na efektywne wykorzystanie właściwości podziału czasu (time-sharing) bez pośrednictwa pamięci dyskowej. Ponadto ułatwione jest uruchamianie programów, ponieważ istnieje możliwość bieżącej modyfikacji bez przebiegów rekompilacyjnych.

ad 2. Niewątpliwie podstawowym kryterium podziału języków jest dla programisty poziom programowania. Pisanie programów w języku wewnętrznym w maszynach III generacji (typu IBM seria 360, System 4, Seria 1900) nie jest (względnie prawie nie jest) stosowane. Wpływa na to kilka czynników (niezależnie od powyżej omówionych zalet języków wyższego rzędu):

a/ przydział adresów rzeczywistych w warunkach pracy wieloprogramowej realizowany jest nie przez programistę, lecz automatycznie przez system operacyjny,

b/ obsługa różnorodnych urządzeń wejścia-wyjścia wymaga zastosowania wielu procedur specjalnych i rozkazów dostosowanych do współpracy z systemem operacyjnym, zaprogramowanie których w programach użytkowych ("zastosowaniowych") za pomocą cyfrowych (wewnętrznych) instrukcji nastręczałoby poważne trudności.

c/ ponieważ łatwo jest o pomyłkę adresów i kodów operacji wyrażanych cyfrowo, oraz trudno je wykryć kontrolą formalną, uruchomienie złożonych programów trwałoby bar-

dzo długo, co oczywiście byłoby kosztowne (koszt pracy maszyny i programisty),

d/ z powodu długiego szkolenia i dużej pracochłonności programowania, byłoby trudno w krótkim czasie obciążyć maszynę w dostatecznym stopniu.

Stąd też zamieniono adresy i kody operacji symbolami łatwymi do zapamiętania (nazwami mnemonicznymi) oraz wprowadzono tzw. makrorozkazy zwalniające programistę od pisania typowych procedur.

Struktura instrukcji języka symbolicznego (poza makrorozkazami) zbliżona jest do struktury rozkazu języka wewnętrznego:

Etykieta (adres) instrukcji bloku	nazwa operacji	nazwy argumentów (operandów) lub adresy względne	nr indeksu (B-rejestru)

Jeśli chodzi o nazwy (symbole) operandów, to występuje ich tyle, ile jest adresów w rozkazie wewnętrznym. Programista posiada dostęp do poszczególnych rejestrów. Są to cechy szczególne, nie występujące w językach wyższego rzędu. Jedynie makrorozkazy traktować można jako swoisty rodzaj wstawek języka wyższego rzędu (jak wiadomo, języki wyższego rzędu budowane są z makrorozkazów).

Oto przykłady języków mnemonicznych:

a/ język mnemoniczny niepełny - City and Guilds Mnemonic Code:
ADD 302,5 oznacza - dodaj zawartość komórki o adresie (302 plus zawartość B rejestru Nr 5),

b/ mnemoniczny pełny (plus makrorozkazy)

- SAS (System Adresów Symbolicznych dla ZAM-41):
  SKO G8 - skocz do procedury z etykietą G8,
  PISZ, CZYTAJ - makrorozkazy (występują one również
  w języku wyższego rzędu SAKO)
- Assembler IBM 360:
  SAMPLE START X, Ø — start programu od adresu wzgl. Ø,
  START BALR 9, Ø — załad. rej. B zawartością adres. Ø,
  USING =, 9 — zdefiniuj rej. 9 jako rejestr bazowy,
- USERCODE dla Systemu 4:
  MP WORK (6) PRICE (2) mnóż WORK przez PRICE, wynik jest
  przechowywany w WORK,
  ED DEST (11), VALUE — pole DEST jest redagowane w ten sposób,
  że nieznaczące zera są zamieniane przez
  znaki "spacje" lub "blank" względnie
  gwiazdkę,
- OPEN nazwa zbioru, nazwa zbioru ....., nazwa zbioru (do 16)
  jest to przykład makrorozkazu otwarcia
  zbioru powodującego sprawdzenie metryki
  zbioru, itp. (lub zapisanie),
  GET makrorozkaz przywołania rekordu ze zbioru wejściowego.

ad 3. Języki do przetwarzania informacji można podzielić na:

a/ uniwersalne, np. PL/I,

b/ specjalizowane (problemowe), np. COBOL, FORTRAN, ALGOL,

c/ wąskospecjalizowane, np. języki do:

- symulacji,
- przetwarzamia struktur listowych (napisów, symboli),
- redagowania wydawnictw graficznych i obsługi display'ów,
- sterowania obrabiarkami,

- tłumaczenia tablic decyzyjnych itp.

Jeśli chodzi o języki do przetwarzania struktur listowych, to istnieje ich na świecie kilkadziesiąt, przy czym do najpopularniejszych należą: LISP, COMIT, SNOBOL. W Polsce opracowano języki LOGOL i EOL. Niektóre właściwości tych języków mogą być z powodzeniem wykorzystane do translacji z języka na język.

Specjalną grupę tworzą języki do operowania na formułach FORMAC (umożliwiają rozwijanie wielomianów, ich dzielenie): ALPAK, FORMULA, ALGOL, SAINT.

Do popularnych kwestii należy porównywanie języków do obliczeń numerycznych (a więc ALGOLu, FORTRANu) z językiem do przetwarzania danych ekonomiczno-administracyjnych (czyli z COBOLem).

Języki pierwszej grupy stosują wyłącznie zapis algebraiczny do oznaczania operacji arytmetycznych, podczas gdy COBOL oprócz tego zapisu (stosowanego po instrukcji COMPUTE) zapewnia również instrukcje słowne w rodzaju DIVIDE, MULTIPLY, ADD, SUBTRACT.

Składnia języków jest całkowicie odmienna. W COBOLu wprowadzono rozdzielanie opisu danych od procedur, stosując dwa odrębne rozdziały: DATA DIVISION i PROCEDURE DIVISION.

Ponadto specjalny rozdział służy do opisu i rezerwacji urządzeń (ENVIRONMENT DIVISION). Najważniejsze znaczenie z punktu widzenia translatora posiada rozdział identyfikacji IDENTIFICATION DIVISION. Tego rodzaju podział funkcjonalny nie istnieje w językach do obliczeń numerycznych, nastawionych na konstruowanie procedur i podających jedynie podstawowe informacje na temat danych (deklaracje INTEGER, REAL, FORMAT, DIMENSION).

Opis danych stosowany w COBOLu pozwala na operowanie nazwami zbiorów (w instrukcjach OPEN, READ), rekordów, pól grupowych

i pól elementarnych, a więc wyraża złożoność "montażową" (strukturę) danych administracyjnych. ALGOL i FORTRAN operują nazwami jednostkowymi (niepodzielnymi), dając możność stosowania wielokrotności pól jednorodnych (indeksowanych). Tę samą właściwość posiada zresztą również COBOL.

Z punktu widzenia alokacji danych język COBOL zapewnia z reguły bardziej efektywny program (dzięki możliwościom "pakowania" danych). Natomiast generuje dłuższe programy dla takich samych zadań (np. w porównaniu z FORTRANem).

Teoretyczne wersje ALGOLu pomijają sprawę formalnego opisu przydziału urządzeń wejścia-wyjścia oraz tzw. edytowania (redagowania) danych wynikowych, pozostawiając to producentom.

Efektem tego jest zmniejszenie stopnia zamienności programów pisanych dla różnych maszyn.

Oto parę instrukcji wydawniczych firmy ICL dla Systemu 4-50:

WRITE / 30, FORMAT / '/ 'ND' / ' / ,X/, gdzie: 30 jest nr urządzenia tj. drukarki wierszowej, ND oznacza wydruk 2 cyfr, X nazwa pola),

PAGE /30,1/ oznacza zmianę strony,

NEWLINE /30,1/ " skok do nowego wiersza.

Są to instrukcje "firmowe" bez odpowiednika w innych wersjach. Ponadto ALGOL nie uwzględnia sprawy rozpoznania stałego i zmiennego przecinka, pozostawiając to maszynie.

W niektórych wersjach REAL oznacza liczbę zmiennoprzecinkową.

ALGOL dla Systemu 4-50 dysponuje bogatymi możliwościami wydawniczymi, porównywalnymi do podobnych cech COBOLu, mianowicie w deklaracji FORMAT można m.in. używać następujących

symboli:

D - cyfra,

N - drukowanie "najstarszej" cyfry jeśli jest znacząca (tj. nie równa zeru),

S - spacja,

. - kropka dziesiętna,

"+" "-" - znaki arytmetyczne.

Przykłady redagowania:

opis	pole przed zredagowaniem	pole po zredagowaniu
7S ≠ NDDD	12	␣␣␣␣␣␣␣ + ␣ ␣ 12
3D4S.5D	123.456789	123␣␣␣␣.45678

Również języki wzorowane na ALGOLu wprowadzają czasem własne środki opisu danych, jak np. ALGEM (ALGOritmiczeskij jazyk dla programmirowania Ekonomiczeskich i Matimaticzeskich zadacz):

9 - cyfra w systemie dziesiętnym,

7 - cyfra w systemie ósemkowym,

1 - cyfra w systemie dwójkowym,

C - dowolny znak,

A - litera rosyjskiego alfabetu,

L - litera łacińskiego alfabetu.

. - kropka dziesiętna w postaci jawnej (dziurkowana),

Δ - kropka dziesiętna "założona",

/7 - zamiana zer nieznaczących przez spację

± - znaki arytmetyczne.

Język ALGOL posiada dogodne właściwości operowania na zbiorach jednorodnych danych, czyli masywach (array). Przynajmniej teoretycznie nie ma tutaj ograniczeń co do ilości (głębokości) indeksów i możliwy jest dynamiczny podział pamięci (granice zbiorów mogą być wzorami arytmetycznymi).

Firmowe wersje ALGOLu wychodzą czasem dalej niż wersje formalne (publikowane jako REPORT ALGOL). Dotyczy to nie tylko wejścia i wyjścia, gdzie m.in. chodzi o powiązania z systemami operacyjnymi, lecz również o możliwość przywołania segmentów napisanych w języku symbolicznym (np. ALGOL 4-50 przywołuje segmenty napisane w USERCODE).

Język COBOL jest szczególnie przydatny w maszynach II generacji bez systemu operacyjnego, a to z tego względu, że operuje odpowiednim aparatem formalnym do opisu konfiguracji maszyny niezbędnej do wykonania zadań, umożliwia automatyczne sprawdzanie (i zapisywanie) metryk oraz deklaruje przydział urządzeń we-wy dla poszczególnych zbiorów.

Języki do obliczeń numerycznych są prostsze, a więc i łatwiejsze do wyuczenia się. Opracowuje się do nich krótsze translatory i w związku z powyższym można ich używać nawet w maszynach z małą pamięcią operacyjną (8 - 16 K).

Wdrożenie pierwszych kompilatorów na małych maszynach napotykało na spore trudności. Wyrazem tego były przypadki, kiedy do skompilowania programów trzeba było ponad 40 przebiegów (A - 1).

Mimo sztywności programu (obowiązkowe sekcje, rozdziały) i tłumaczenia go techniką kompilacyjną, język COBOL jest niekiedy stosowany w systemach abonenckich.

Wg danych pewnej ankiety (S-4, 1969 r.), którą objęto 497 użytkowników w/w systemów, struktura używanych przez nich języków programowania była następująca:

FORTRAN 39,6%,
BASIC 31%,
COBOL 7,6%,
PL/I 7,0%,
ALGOL 2,8%.

Jeśli chodzi o język BASIC (Beginner's All-purpose Symbolic Instruction Code), to opracowano go w Dartmouth College (w okresie 1964-1965) i wdrożono na maszynie GE 225. Pomyślany był, jako język do wstępnej nauki programowania przed przystąpieniem do ALGOLu lub FORTRANu. Posiada proste konstrukcje formalne, będąc w ten sposób bardzo łatwy do wyuczenia (w zasadzie programuje się już po pierwszym dniu nauki). Uruchamianie programów jest ułatwione. W dowolnej chwili można wprowadzać poprawki, podając jedynie nr sekwencji (zdania) i nowe jej brzmienie. Jako inne języki konwersacyjne wymienić można AMTRAN, JOSS, CAL, QUIKTRAN. Częściowo słusznym zarzutem (z dokumentacyjnego punktu widzenia można mu się przeciwstawić) w stosunku do COBOLu jest zbytnia obfitość części opisowych, jakiej musi używać programista. Szczególnie odnosi się to do opisu danych, wyrażania operacji arytmetycznych (jeśli brak instrukcji COMPUTE). Wada ta daje się częściowo usunąć, jeśli stosowane są instrukcje COPY (do ściągania z biblioteki zarówno opisu danych jak i procedur, wspólnych dla kilku programów). oraz dozwolone jest użycie takich skrótów, jak PIC do oznaczenia PICTURE, COMP zamiast COMPUTATIONAL itp.

Znane są specjalne wersje COBOLu, używające jedynie minimalnych oznaczeń. Na przykład wersja COAX stosuje jednoliterowe

oznaczenia :

- F' oznacza FILLER /w DATA DIVISION/ lub
  FROM / w PROCEDURE DIVISION/
- A' ADD
- G' GREATER,

itp.

Inne usiłowania polegają na zastosowaniu specjalnych formularzy z gotowymi nadrukami (np. RAPIDWRITE).

Werbalność COBOLu związana jest niewątpliwie z celem, jaki był stawiany przed tym językiem, kiedy chodziło o zbliżenie specyfiki programowania do specyfiki problemu (a więc do ludzi pracujących w określonym zawodzie).

Ponadto okazało się, że z pozoru proste tematy administracyjne są bardzo pracochłonne w programowaniu i należało stworzyć programiście takie narzędzie pracy, które pozwala szybko uruchamiać bardzo duże programy(to znaczy łatwo je analizować i szybko odnajdywać błędy) i będzie równocześnie stanowić dobrą dokumentację do kontynuacji pracy przez innych programistów.

COBOL bardziej uwzględnia wymogi nowoczesnej technologii przetwarzania, wykorzystując np. walor pracy równoczesnej (overlapping), dzięki możności podwójnego buforowania każdego wejścia i wyjścia oraz obowiązkowemu wprowadzenia odrębnych obszarów wejściowych i wyjściowych.

Pod względem czasu przetwarzania zadań angażujących duże zbiory i złożone rekordy język COBOL zdecydowanie wyprzedza języki do obliczeń numerycznych (w tym również FORTRAN posiadający jakie takie możliwości opisu wprowadzanych danych - za pomocą deklaracji FORMAT).

Oto wyniki pewnych testów (J - 1) przeprowadzonych na maszynie IBM 7094 w połowie lat 60-tych:

zadanie	COBOL	FORTRAN II
1. Czytanie zbiorów i sprawdzanie jakości pól rekordów		
a/ wielkość bloku 10 rek.	2,3 min.	15 min.
b/ wielkość bloku 1 rek.	4,8	10,2
2. Kopiowanie zbioru		
rozmiar bloku 10 rek.	2,8	13,8

Można powiedzieć, że wraz ze wzrostem pojemności pamięci i szybkości działania maszyn wzrasta efektywność kompilatorów COBOLu. Tam, gdzie tego nie udaje się osiągnąć w trakcie translacji, stosowane są specjalne optymalizatory (COBOL optymiser), które przerabiają wersję otrzymaną w wyniku tłumaczenia. Przykładem takiego optymalizatora jest COBOL OPTIMISER opracowany w Capex Corp. of Phoenix Arizona (X - 2).

Nie można mówić o efektywności kompilatora w sensie ogólnym. Uważa się, że dobre kompilatory COBOLu opracowuje firma Honeywell, o czym świadczy fakt, że udział wdrożonych kompilatorów tej firmy jest znacznie wyższy od udziału w sprzedaży maszyn.

Wydaje się, że na temat przydatności COBOLu przede wszystkim powinni wypowiadać się użytkownicy. Otóż wg ankiety (X - 3) przeprowadzonej w USA wśród 724 użytkowników COBOLu, 560 z nich

(ok. 80%) potwierdziło pełną przydatność tego języka, przy czym 65% z tej grupy oświadczyło, że zakres stosowania tego języka stale się u nich zwiększa.

Prowadząca ankietowanie Auerbach Corp. podała wyniki mówiące, że użycie COBOLu jest dwukrotnie wyższe w takich zastosowaniach, jak ewidencja zapasów, rozliczenia zakupów, sprzedaży itp. - zaś język symboliczny dorównywał COBOLowi jedynie w przypadkach przetwarzania wyrywkowego (random access job.).
Największymi zwolennikami COBOLu były najczęściej ośrodki małe, zatrudniające do kilkunastu programistów.[1/]
Reasumując zalety COBOLu stwierdzić można, że jest to język stosunkowo łatwy do wyuczenia i stosowania (w porównaniu z językami symbolicznymi) oraz przedstawiający duże walory dokumentacyjne. Dzięki zastosowaniu zwrotów PERFORM ...., DEPENDING ON, itp. upraszcza się organizacja programu, zaś w czasie testowania pomocne mogą być procedury TRACE, EXIDIT itp.

Wadami COBOLu, w porównaniu z językami symbolicznymi, są:
a/ większa, np. o 30%, zajętość pamięci operacyjnej,
b/ dłuższy, np. o 10%, czas przetwarzania (T - 2),
zaś zaletami: kilkakrotne zmniejszenie pracochłonności programowania i czasu uruchamiania programów.

Skoro mówimy o językach do przetwarzania danych, nie możemy pominąć arcyciekawej próby (1959, USA) stworzenia "algebry informacji" w postaci języka LSG /Language Structure Group/.

1/ Na poparcie danych literaturowych mogę od siebie dodać, że w czasie mego 7-miesięcznego pobytu w Wielkiej Brytanii spotkałem ośrodki obliczeniowe, w których posługiwano się prawie wyłącznie językiem COBOL.

Znane są próby zastosowania tego języka do obliczania list płac.

Analiza porównawcza języków programowania jest utrudniona, ponieważ stale ulegają one zmianom, zarówno w wersjach formalnych, jak i firmowych. Obserwuje się m.in. wzajemne zapożyczenia, istnieją grupy języków wzorowane na FORTRANie, ALGOLu i COBOLu, względnie czerpiące ze wszystkich tych języków na raz.

Przykładem wpływu FORTRANu na ALGOL może być deklaracja FORMAT używana w niekórych wersjach ALGOLu po instrukcji WRITE (to też niestandardowa instrukcja) oraz instrukcja READ z numerem urządzenia jako parametrem.

Niektóre języki (czy też raczej pakiety programowe) posługują się wprost aparatem formalnym innego języka, jak to jest w przypadku BASEBALL (system pytanie-odpowiedź), używającego zwrotów języka IPL-V do przetwarzania struktur listowych.

Wyrazem pewnego podsumowania dorobku programowania wydaje się być uniwersalny język PL/I. Stanowi on zjawisko ze wszech miar godne uwagi. Nie stał się jeszcze jednak językiem powszechnego stosowania (w 1968 roku tylko 1% komputerów miało opracowany translator tego języka - wg R - 3).

W celu zbadania efektywności tego języka w zastosowaniach ekonomiczno-administracyjnych przeprowadzono testy porównawcze z COBOLem na przykładzie konkretnych programów m.in. dotyczących list płac (R - 3). Porównanie to nie podważyło pozycji COBOLu. Oto wyniki:

a/ uruchamianie programów PL/I trwało znacznie (dwukrotnie) dłużej, mimo krótszych programów zewnętrznych,

b/ czas pisania programów PL/I był tylko nieznacznie krótszy;

c/ czas przetwarzania był w przypadku COBOLu znacznie krótszy (dla dwóch programów wynosił 2/3 i 1/3 czasu PL/I);

d/ uczenie się języka PL/I jest znacznie trudniejsze, natomiast później użycie aparatu formalnego wydaje się być łatwiejsze. Jest to więc niejako język przeznaczony dla zawodowych programistów, specjalizujących się w programowaniu PL/I (wtedy rekompensują się nakłady na szkolenie);

e/ PL/I zapewnia lepsze możliwości kontroli logicznej i operowania na danych (włącznie z manipulacją na bitach, nie zawsze stosowaną w COBOLu). Natomiast COBOL posiada lepsze procedury czytania i pisania (włącznie z takim ułatwieniem wydawniczym, jakim jest REPORT-WRITER);

f/ łatwiejsza jest korekta czy też modyfikacja programów PL/I.

Tyle mówią praktyczne konfrontacje. Natomiast pod względem poziomu programowania, język PL/I przewyższa pozostałe języki. Jest próbą opracowania języka adekwatnego do hardware'u i systemu operacyjnego maszyn III generacji.

Język PL/I opracowano, starając się uwzględnić osiągnięcia istniejących języków, a w szczególności FORTRANu, ALGOLu i COBOLu.

Notacja języka PL/I jest stosunkowo zwięzła i w tym względzie PL/I opiera się raczej na tradycjach FORTRANu i ALGOLu, niż COBOLu. Język ten uważany jest za szczytowe osiągnięcie języków proceduralnych. Ponieważ ma być samowystarczalny, nie przewiduje on wstawek z innych języków (przynajmniej nie przewidywały tego pierwsze wersje).

Specyficzną cechą PL/I jest powiązanie z systemem operacyjnym i hardware'm. Wyraża się to przede wszystkim w organizowaniu pracy równoległej, zarówno w przetwarzaniu wielomaszynowym jak i w maszynie wieloprogramowej. Przykładem tego rodzaju pracy jest tzw. wieloproceduralność (lub podprogramowanie), polegająca na tym, że równolegle wykonuje się kilka procedur tego samego programu, przy czym zorganizowane może to być asynchronicznie lub w systemie przerywań.

Asynchroniczność osiągana jest m.in. poprzez stosowanie takich instrukcji, jak: wstrzymanie procedury WAIT, przerwanie procedury na określoną ilość jednostek czasu DELAY, badanie ukończenia COMPLETE itp.

Z punktu widzenia użytkownika specjalne znaczenie posiadają dwie cechy PL/I: modułowość i zasada domyślności.
Modułowość sprawia, że nie trzeba znać języka w całości, lecz stosuje się jego podzbiory stosownie do poziomu programisty i złożoności zastosowania.

Dzięki zasadzie domyślności, w przypadku wykrycia braku deklaracji translator nie sygnalizuje błędu, lecz dobiera deklarację najbardziej prawdopodobną w danym przypadku. Na przykład, jeśli przy zmiennej EXTERNAL nie ma deklaracji sposobu przydziału pamięci, translator przyjmuje STATIC.

Bardzo zwięzły w porównaniu z COBOLem jest sposób opisu struktury danych (połączony od razu z indeksowaniem):

```
DECLARE 1A/2/, 2B, 2C   oznacza:poziom 1  A/1/
                                          2 B
                                          2 C
                                        1 A/2/
                                          2 B
                                          2 C
```

W PL/I rozróżnia się dwa sposoby przesyłania danych:
a/ zorientowany na tzw. strumień,
b/ zorientowany na rekord (zapis, dokument).
Do wejścia-wyjścia strumienia używa się instrukcji GET i PUT, zaś w drugim przypadku READ i WRITE. Przy strumieniowym przesyłaniu dane są rozpatrywane jako ciągły strumień elementów w postaci znakowej. W rekordowym przesyłaniu jednostką przesyłową jest rekord, przy czym jego elementy mogą być zakodowane w różnych formach.

Głównymi pojęciami opisu danych są maszyny i struktury. Przyrównując je do tradycyjnych pojęć, struktura odpowiada rekordowi, zbiorowi jako złożona struktura), zaś masyw - tabeli jednorodnych danych, np. macierzy.

Sposób kompilacji programów PL/I różni się znacznie od dotychczasowych rozwiązań. Mianowicie obejmuje również elementy przetwarzania (np. obliczanie wspólnych lub jednorazowych wyrażeń) oraz umożliwia programiście włączenie się czynne do procesu kompilacji.

Języki wąskospecjalizowane

Języki te służą do programowania takich specjalnych zastosowań, jak:
a/ sterowanie obrabiarkami (machine tool control) np. APT,
b/ konstrukcje logiczne np. LOTIS, APL,
c/ budownictwo cywilne np. COGO, STRESS,
d/ pisanie translatorów np. CLIP, COGENT,
e/ symulacja cyfrowa np. GPSS, SIMSCRIPT, SOL,
f/ systemy pytanie-odpowiedź np. COLINGO, BASEBALL, QUERY, ADAM, DEACON,

g/ wyjścia graficzne i ekranowe np. GRAF, PENCIL, DOCUS,

h/ tłumaczenie tablic decyzyjnych.

Przeznaczeniem języków wąskospecjalizowanych jest obsługa wybranego rodzaju zastosowania, w którym są łatwiejsze w użyciu i efektywniejsze od innych. Wynika to z wyposażenia tych języków w specjalistyczne instrukcje, jak np.

a/ w APT : SPINDL/OFF (turn off spindle - obtoczyć wał)

TL RGT, GO PGT/BASE (with the tool on the right go right along the line BASE)

b/ COLINGO: UPDATE, CHANGE, EXECUTE, ALLOCATE, ANALYZE.

Skoro wspomnieliśmy o języku APT, warto dodać, że opracowany został w MIT (Massachusetts Institute of Technology) w latach 1952 - 1956. Ulepszone wersje to:

1958 APT II, 1961 APT III.

Jeśli chodzi o język do symulacji, to jednym z pierwszych języków tego typu był język DYNAMO opracowany w MIT w 1959 roku dla IBM 704. Najbardziej znanymi językami w zakresie symulacji są GPSS, SIMSCRIPT i SOL.

Opis GPSS (General Purpose Simulation System) po raz pierwszy opublikowano w 1961 roku. SIMSCRIPT (Simulation Oriented Language) został opracowany w RAND CORP. w 1960 roku.

SOL (Simulation Oriented Language) stanowi próbę połączenia dwóch różnych systemów językowych, a mianowicie systemu "blokowego" (GPSS) i "zdaniowego" (SIMSCRIPT).

Języki do pisania translatorów można podzielić na dwie grupy:

1. oparte o zasady zwykłych języków programowania, np. CLIP (Compiler Language for Information Processing) oparty został na ALGOLu 58, lecz posiada istotne uzupełnienia w zakresie manipulacji na danych (np. podanie rozmiaru nieindeksowanych danych) oraz instrukcje wejścia-wyjścia,
2. do tzw. tablicowego tłumaczenia składni (syntax directed table-driven compilation), np. COGENT (Compiler and GENeralized Translator) opracowany dla CDC 3600.

W celu pokazania specyfiki języków do pisania translatorów przytoczymy przykład instrukcji w języku TMG:
INTEGER...ZERO-MARKS DIGIT *INSTALL co oznacza: pomiń nieznaczące zera, zaznacz początek łańcucha i znajdź co najmniej jedną cyfrę, a następnie wyspacjuj wszystkie następne cyfry oraz umieść "symbol" do tabeli znaków i drzewa (grafu) wyjścia.

Generatory programów

Przez generację rozumiemy automatyczne tworzenie typowych (powtarzalnych) sekwencji operacji w oparciu o "skrótowe" dane (rodzaj sekwencji, parametry) dostarczone przez programistę. Ze względu na radykalne zmniejszanie pracochłonności programowania, zastosowanie generatorów wydaje się mieć duże perspektywy.

Wyróżnić można trzy podstawowe poziomy generacji:

a/ makrogenerator,

b/ generator programu standardowego,

c/ generator-język programowania.

a/ Makrogenerator

Generuje rozkazy elementarne na podstawie makrorozkazu i deklaracji (opisu danych itp.) umieszczonych w programie zewnę-

trznym.[1] Makrogeneratory są szczególnie przydatne do wyrażania operacji o wielu możliwych kombinacjach (np. typów operacji dodawania, zależnych od charakteru danych, żądanych zaokrągleń, kontroli formatu wyniku itp.). Kody tych operacji są <u>każdorazowo</u> generowane.

b/ <u>Generator programu "standardowego"</u>

Służy do generowania takich programów, jak np. SORT, REPORT, WRITER, które w klasycznej postaci posiadają sztywną formę programów standardowych. Zaletą metody jest wyeliminowanie ograniczeń programu standardowego i wyższa efektywność. W przypadku generacji SORT, generator na podstawie parametrów i zasadniczego szkieletu działania programu generuje program sortowania dopasowany do konkretnych wymogów (rozmiarów i typów rekordów, rodzaju i ilości kluczy sortowania itp.) oraz dostępnej konfiguracji. Jeden z pierwszych generatorów SORT został napisany przez F.Holbersona jeszcze w pierwszej połowie lat 50-tych (S - 1).

REPORT-WRITER służy do wybierania danych ze zbiorów oraz organizacji ich wydruku. Istnieją również generatory do aktualizacji zbiorów, generowania kompilatorów (tzw. compiler-compiler, compiler generator) itp.

c/ <u>Generator - język programowania</u>

Jako przykład tego rodzaju (nieproceduralność !) języka programowania wymienić można RPG-REPORT PROGRAM GENERATOR, opracowany m.in. dla maszyn IBM 1401, seria 360, UNIVAC.

1/ Inną drogą realizacji makrorozkazu - z pominięciem generacji- jest przywołanie gotowego podprogramu.

RPG umożliwia proste zaprogramowanie nawet złożonego problemu z dziedziny przetwarzania danych administracyjno-ekonomicznych (obejmującego szereg operacji obliczeniowych, wydruki wg różnych stopni kontroli itp.).

Programista posługuje się standardowymi arkuszami programowania o nadrukach odrębnych dla:

1/ opisu zbiorów (rodzaje),

2/ opisu struktury rekordów wejściowych,

3/ opisu danych wyjściowych na drukarkę wierszową (pola, tytuły, sumy wg stopni kontroli, znaki wydawnicze),

4/ opisu obliczeń (rodzaje operacji, sposób kontroli obliczeń).

Zwolennicy RPG twierdzą (R - 6), że 80 - 90% programów przetwarzania danych może być z powodzeniem napisane w tym języku. Optymiści (X - 4) oczekują, że w najbliższej przyszłości 75% wszystkich programów przetwarzania danych zostanie napisanych w RPG, podczas gdy tylko 15% przypadnie na COBOL a 10% na BAL i FORTRAN.

Systemy operacyjne (CONTROL PROGRAM, CONTROL LANGUAGE, CONTROL SOFTWARE, OPERATING SYSTEM)

Opracowanie systemów operacyjnych stanowi problem równie ważny jak rozwój "użytkowych" języków programowania. Chodzi przede wszystkim o zmniejszenie przestojów maszyny (poprzez zredukowanie do minimum czynności operatora, automatyczne przełączanie pracy maszyny z zadania na zadanie, harmonogramowanie pracy urządzeń) oraz stworzenie języka komunikacji operatora z maszyną. Wymogi systemów operacyjnych coraz bardziej rzutują na konstrukcję języków programowania.

System operacyjny jest niezbędny w maszynach wieloprogramowych, a znaczenie jego szczególnie wzrasta w wielomaszynowym przetwarzaniu. Dzięki niemu uzyskać można tzw. POLIMORFICZNOŚĆ, polegającą na tym, że zestaw komputerowy zmienia swą "postać" stosownie do problemów, poprzez zmianę połączeń pomiędzy elementami składowymi konfiguracji oraz wprowadzanie zmian strukturalnych (zmiana długości słowa, akceptowanie innej listy rozkazów itp).

Pierwsze publikacje na temat systemów operacyjnych pojawiły się w 1953 roku i dotyczyły systemów SOS (Share Operating System) i FMS (Fortran Monitor System) opracowanych dla maszyn IBM 709 i 704 (T - 2).

Firma Honneywell pierwszy swój system operacyjny opracowała w 1957 roku dla maszyny D-1000, zaś w 1960 roku stworzyła pakiet EXECUTIVE do kierowania pracą wieloprogramową (do 7 programów / X - 5 /.

Proste systemy operacyjne (zwane w IBM dyrygentami, w odróżnieniu od szerszego pojęcia systemu operacyjnego), składające się z kilkuset rozkazów, służyły głównie do wzywania standardowych podprogramów umieszczonych na taśmie bibliotecznej. Przykładem takiego dyrygenta jest MONITOR-70 dla maszyn IBM 7070/7074 (F-1).

Nieco większe systemy operacyjne sterowały również pracą urządzeń wejścia-wyjścia oraz realizowały kontakt operatora z maszyną (i odwrotnie). Systemy operacyjne maszyn III generacji kierują pracą wieloprogramową, przydzielają urządzenia wejścia-wyjścia, dokonują przydziału obszarów pamięci operacyjnej itp.

Przez pojęcie systemu operacyjnego firma IBM i firma Honeywell rozumieją cały kompleks oprogramowania sterującego. Przybliżeniem do tej koncepcji był system MONITOR MACDONALD dla maszyn IBM

709/7090, obejmujący swym zasięgiem kierowanie tłumaczeniami. Jedną z pierwszych wersji systemu operacyjnego IBM był SO dla IBM 1410/7010, zaś jej pełnym wcieleniem - oprogramowanie serii 360.

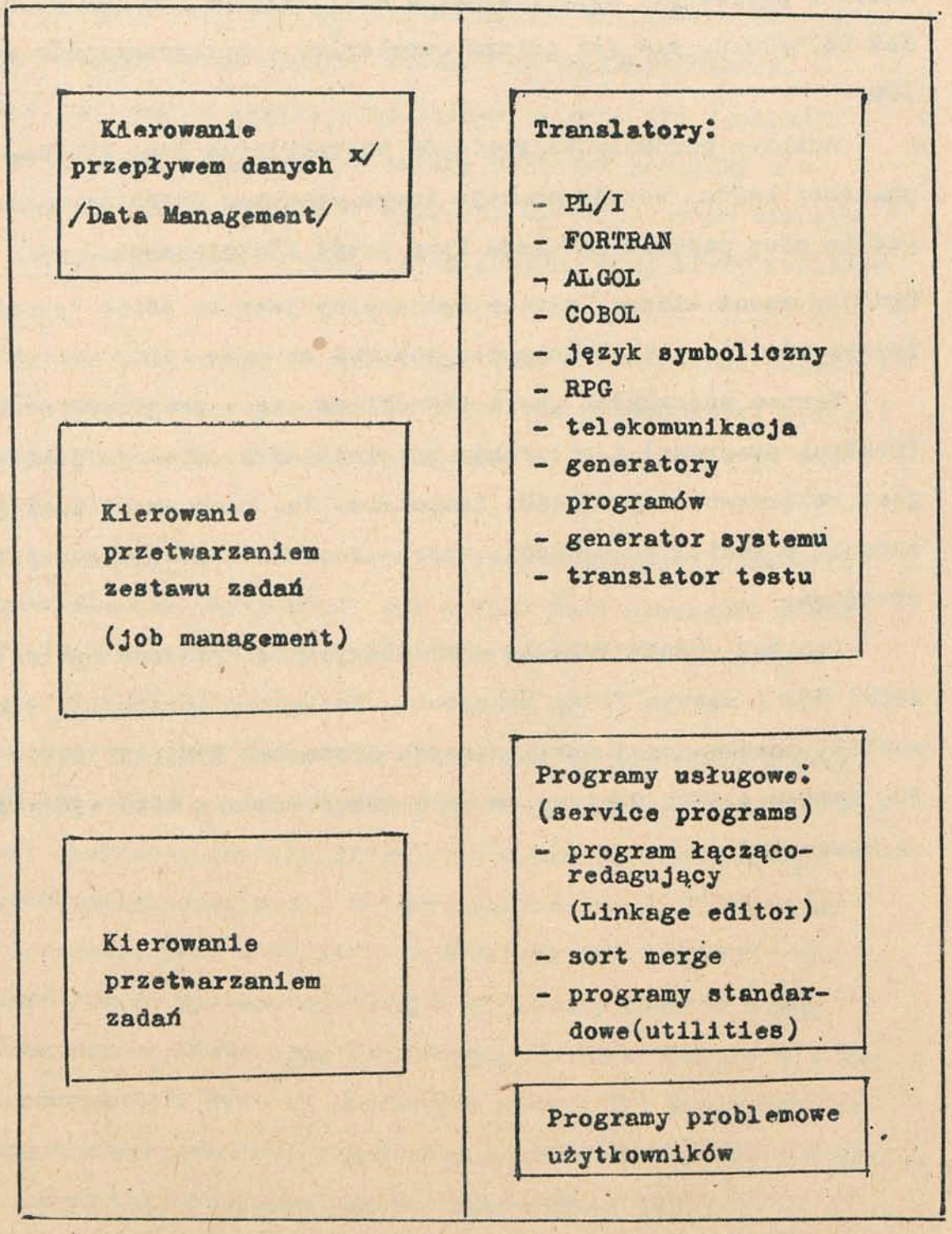
Analiza porównawcza systemów operacyjnych jest trudna, ponieważ każda wersja operuje innym aparatem pojęciowym, zaś pod te same pojęcia podkłada inną treść funkcjonalną.

Ogólnie rzecz biorąc, system operacyjny jest to zbiór "prawideł" (programów), czyniący komputer zdolnym do pracy.

System operacyjny serii 360 składa się z programów sterowania (Control programs) i programów przetwarzania. Głównym jego celem jest zwiększenie wydajności komputera. Np. Management task grupuje zadania w takiej kolejności, która zapewni lepsze wykorzystanie urządzeń.

Poniżej podamy zwięzłą charakterystykę systemów operacyjnych serii 360 i maszyn firmy Honeywell. Następnie spróbujemy dokonać analizy porównawczej następujących systemów: IBM, ICL seria 1900, ICL system 4, NCR Century, w celu zilustrowania istniejących rozbieżności.

Tabela 1. system operacyjny IBM 360



x/ W literaturze krajowej spotyka się też synonimiczne terminy angielskiego pojęcia: "data management", a mianowicie: "operowanie danymi", "sterowanie danymi" lub "sterowanie ruchem danych" (Przyp.Red.)

Tabela 2. System operacyjny Honeywell OS/200

I. S U P E R V I S O R

1. MONITOR STAŁY /Resident Monitor/
   - wprowadzanie programów,
   - kierowanie pracą wieloprogramową jedno lub dwustrumieniową,
   - kierowanie wejściem-wyjściem /alokacja kanałów przesyłowych i urządzeń,
   - kierowanie konwersją danych /karty-taśma magn., taśma magnet.-drukarka, taśma magn.-dziurkarka taśmy,
   - kierowanie komunikacją

2. MONITOR PRZEJSCIOWY /Transitional Monitor/
   - wczytywanie kart sterujących następnego programu i przekazanie parametrów do monitora stałego,
   - inicjowanie komunikacji,
   - wykonywanie dumpingu /PAO, dysku, TM/

3. KIEROWANIE ZBIORAMI BAZY DANYCH - składnik opcjonalny /data base file management/

II. K O M P O N E N T Y Z A L E Ż N E /Dependent Components/

1. PODSYSTEM KIEROWANIA DANYMI /dla zbiorów w papięci dyskowej/
   - kierowanie dostępem, założenie i aktualizacja rozmieszczenia zbiorów, zamiana ścieżek, załadowanie zbiorów, listowanie rozmieszczenia /map/
   - podprogramy wejścia-wyjścia: pakiety logiczny i fizyczny we-wy, itp.

2. PODSYSTEM JĘZYKÓW I PROGRAMÓW
   - translatory, archiwowanie programów źródłowych i wewnętrznych.

3. PROGRAMY STANDARDOWE

Tabela 3. Zestawienie porównawcze systemów operacyjnych

IBM: BOS, DOS, TOS	ICL: seria 1900	ICL: system 4	NCR: Century
1. IPL Loader /initiate program load./ program inicjujący wprowadzanie 2. SUPERVISOR -obsługa przerywań, - sterowanie wejściem-wyjściem, - sygnalizowanie błędów, - koordynacja przejścia od task control do job control 3. JOB CONTROL	1. EXECUTIVE - kontakt z operatorem, - ładowanie programów, - przydzielanie urządzeń, - organizacja pracy wieloprogramowej. 2a. AUTOMATYCZNY OPERATOR 2b. GEORGE /General Organisational Environment/ stanowi rozwinięcie funkcji EXECUTIVE GEORGE IV: - stronicowanie, - harmonogramowanie prac, - rejestracja wykorzystania jedn.centr. i urządzeń we-wy przez poszczególne programy GEORGE III: zawiera m.in. MOP /Multiplo On-line Programming Module/ umożliwiający pracę w wielodostępie dla 60 abonentów.	1. EXECUTIVE a/Job Control - przydział pamięci, - przydział urządzeń - wprowadzanie programu /Job input, Job scheduler Allocator Deallocator/ b/Supervisor - kontakt z operatorem, - system przerywania, - kronika błędów, - nadzór nad pracą wieloprogram. /S05J-6 strumieni, 14 programów/ 2. PROGRAM TRIALS SYSTEM 3. UTILITIES /program standard./	1. MONITOR - wprowadzanie programu, - kierowanie kolejnością pracy. 2. INPUT-OUTPUT EXECUTIVE - przydział pamięci, - przydział urządzeń wejścia-wyjścia, - kontrola błędów. 3. KIEROWANIE PRZEPŁYWAMI DANYCH /data traffic control/ - harmonogramowanie, - system przerywań.

Jak wynika z powyższego, rozbieżności dotyczą szczególnie takich pojęć, jak SUITERVISOR i EXECUTIVE, które w każdym przypadku znaczą właściwie co innego przy zachowaniu pewnych wspólnych funkcji.

Ogólnie rzecz biorąc system operacyjny spełnia następujące funkcje:

- wprowadzenie programów i kontrola ich realizacji,
- interpretacja rozkazów operatora i ich wykonanie,
- raportowanie odchyleń operatorowi (np. nadmiaru, przekroczenia limitu pamięci itp),
- interpretacja kart sterowania (CONTROL CARDS, zwane też Steering Cards),
- harmonogramowanie przebiegów,
- kontrola pracy urządzeń peryferyjnych,
- przydział urządzeń peryferyjnych do programów i zwalnianie tych urządzeń,
- zatrzymanie programu, o ile dalsza jego realizacja nie może być kontynuowana oraz powtórne uruchomienie po usunięciu przyczyny zatrzymania,
- kierowanie pracą wieloprogramową.

Złożoność systemu operacyjnego SO stale wzrasta np. SO dla maszyny IBM 360-91 zawiera ponad półtora miliona instrukcji. GEORGE III wraz z EXECUTIVE zajmuje około 12 tysięcy słów pamięci operacyjnej.

System operacyjny MASTER (CDC 3300) stosowany jest w maszynach wyposażonych w co najmniej 64 K PAO. Dzięki temu systemowi, uzyskano tak sprawną pracę wieloprogramową, że przepustowość maszyny zwiększyła się o 40% (X - 6).

System operacyjny SIPROS (SImultaneous PRocessing Operating System) dla CDC 6000 stanowi przykład kierowania bardzo złożoną konfiguracją 11 jednostek przetwarzania, w której występuje równoczesność: wykonywania programów i instrukcji, dostępu do poszczególnych modułów pamięci operacyjnej oraz wykonywana jest

tzw. rekonfiguracja (dodanie nowego składnika, usunięcie uszkodzonego).

V. GENEZA JĘZYKÓW POWSZECHNEGO ZASTOSOWANIA

Pisząc o językach powszechnego zastosowania mamy na myśli nie ich uniwersalność, lecz liczbę wdrożeń translatorów i uruchomionych programów.

Na pewno za takie języki można uważać ALGOL, FORTRAN i COBOL, zaś język PL/I (jako uniwersalny i nowoczesny) wydaje się mieć szanse zostania takim językiem.

Dużą przeszkodą w osiągnięciu powszechności języków programowania były zaściankowe interesy firm. Po pierwszym okresie przykrych doświadczeń specjaliści dochodzą do wniosku, że znaczenie języków programowania wykracza daleko poza rolę narzędzia walki konkurencyjnej i kwestia jakości (oraz powszechności) języków nie jest sprawą wewnętrzną jednej firmy. W ten sposób doszło do powstania języków ALGOL i COBOL, jako produktu współpracy przedstawicieli firm i naukowców.

1. Geneza ALGOLu

ALGOL (ALGOrythmic Language) został opracowany przez przedstawicieli Anglii, Danii, Francji, Holandii, NRF, Stanów Zjednoczonych i Szwajcarii. Wersję języka przygotowano na szeregu spotkań odbytych na terenie Europy i Stanów Zjednoczonych.

Największe zasługi wydają się mieć dwie organizacje:

tzw. rekonfiguracja (dodanie nowego składnika, usunięcie uszkodzonego).

V. GENEZA JĘZYKÓW POWSZECHNEGO ZASTOSOWANIA

Pisząc o językach powszechnego zastosowania mamy na myśli nie ich uniwersalność, lecz liczbę wdrożeń translatorów i uruchomionych programów.

Na pewno za takie języki można uważać ALGOL, FORTRAN i COBOL, zaś język PL/I (jako uniwersalny i nowoczesny) wydaje się mieć szanse zostania takim językiem.

Dużą przeszkodą w osiągnięciu powszechności języków programowania były zaściankowe interesy firm. Po pierwszym okresie przykrych doświadczeń specjaliści dochodzą do wniosku, że znaczenie języków programowania wykracza daleko poza rolę narzędzia walki konkurencyjnej i kwestia jakości (oraz powszechności) języków nie jest sprawą wewnętrzną jednej firmy. W ten sposób doszło do powstania języków ALGOL i COBOL, jako produktu współpracy przedstawicieli firm i naukowców.

1. Geneza ALGOLu

ALGOL (ALGOrythmic Language) został opracowany przez przedstawicieli Anglii, Danii, Francji, Holandii, NRF, Stanów Zjednoczonych i Szwajcarii. Wersję języka przygotowano na szeregu spotkań odbytych na terenie Europy i Stanów Zjednoczonych.

Największe zasługi wydają się mieć dwie organizacje:

GAMM (Geselschaft für Angewandte Mathematik und Mechanik) oraz ACM (Association for Computing Machinery).
GAMM już w 1955 roku powołał zespół do prac nad sprawami translacji wzorów matematycznych. W 1957 roku zawarte zostało porozumienie pomiędzy ACM i GAMM w sprawie opracowania wspólnego języka.

Grupa Europejska składała się z przedstawicieli organizacji GAMM, Association Fransaise de Calcul, British Computer Society i Nederlands Rekenmachine Genootschap. W skład grupy Amerykańskiej wchodziły organizacje: ACM,USE,SHARE i DUO.

W roku 1958 nastąpiło w Zurichu spotkanie obu tych grup. Na łamach "Communications of the ACM" postanowiono drukować rezultaty prac. W 1958 roku opublikowano pierwszą wersję ALGOLu, zwanego wówczas IAL (International Algebraic Language).

Wersję ALGOL 60 poprzedziło spotkanie w 1959 roku w Paryżu, na którym John Backus zaprezentował formalną metodę definiowania składni, nazwaną później BNF (Backus Normal Form lub Backus Naur Form) oraz przedstawił użycie tej metody do definiowania ALGOLu. Na następnym spotkaniu w Paryżu w tym samym roku Naur zreferował wersję ALGOL 60.

W 1962 roku ulepszono język poprzez usunięcie błędów i niejasności tworząc "Revised Report on the Algorythmic Language ALGOL 60". Wersja ta wraz z dodatkowymi specyfikacjami wejścia-wyjścia została zatwierdzona jako norma ISO.
Na wiosnę 1968 roku opublikowano opis ALGOLu 68 pod redakcją van Wijngaardena. Algol 68 zdefiniowany został przez Grupę Roboczą 2.1.IFIP. Wersja ta zawiera szereg nowych możliwości: przetwarzanie równoczesne (parallel processing), segmentowa kompilacja, możliwość deklarowania nowych struktur danych,

operowania na liczbach podwójnej i potrójnej precyzji, tablice o zmiennych granicach, priorytety, format danych itp.

Zasługi ALGOLu dla rozwoju teorii i praktyki programowania są znaczne. Przede wszystkim po raz pierwszy zaproponowany został język stanowiący formalną metodę definiowania składni, będący równocześnie językiem publikacji. ALGOL umożliwia proste i zwięzłe zaprogramowanie procedur obliczeniowych, "nieograniczone" indeksowanie elementów tablic, procedury rekursywne itp. Dzięki tym właściwościom ALGOL stał się podstawą konstrukcji szeregu języków problemowych.

Na bazie ALGOLu 58 opracowano języki: MAD, NELIAC, JOVIAL. Ten ostatni przerodził się w język o wielu dodatkowych możliwościach.

Ponadto w oparciu o ALGOL powstały tak różnorodne języki problemowe, jak:

a/ SIMULA (język do sterowania),

b/ DIAMAG (time-sharing system),

c/ LISP 2 (obliczenia algebraiczne, analiza lingwistyczna),

d/ GPL (general purpose language),

e/ AED (obliczenia konstrukcyjne),

f/ SFD-ALGOL (System Function Description - ALGOL).

Niektóre wersje ALGOLu noszą inne nazwy np. SMALGOL (Small ALGOL), BALGOL (Burroughs ALGOL).

Na szczególne podkreślenie zasługuje JOVIAL (Jules'Own Version of the International Algebraic Language), posiadający właściwość tzw. COMPOOL (COMmunication POOL) - pracy równoległej (jedyny język z tą właściwością do momentu pojawienia się PL/I) oraz możliwość użycia do pisania własnego kompilatora.

2. Geneza FORTRANu

FORTRAN (FORmula TRANslator) należy do najstarszych języków programowania. Dzięki kolejnym udoskonaleniom (FORTRAN II, FORTRAN IV) ten firmowy język IBMu utrzymuje się do dzisiejszego dnia, jako jeden z najpopularniejszych języków programowania zarówno w USA jak i w Europie. Język ten okazał pewien wpływ na pierwsze wersje ALGOLu (jeden z twórców ALGOLu J.Backus uczestniczył w opracowywaniu FORTRANu), zaś z kolei ALGOL oddziaływał na FORTRAN IV.

Pierwszy dokument na temat FORTRANu znany był już w roku 1954, zaś podręcznik powstał w okresie 1956-1957.
W 1958 roku opracowano FORTRAN II, który różnił się od pierwszej wersji m.in. dodaniem END i możliwością przywoływania podprogramów (poprzez CALL, SUBROUTINE). Obie wersje były mocno ukierunkowane na sprzęt (WRITE, TAPE, READ DRUM).

W 1962 roku powstaje FORTRAN IV dla maszyny IBM 7030 (STRETCH), w której to wersji można używać deklaracji LOGICAL, DOUBLE-PRECISION, REAL, INTEGER, logicznej instrukcji IF (obok arytmetycznej) itp. oraz usunięte zostały wyrażenia związane z przełącznikami (kluczami) i słowa TAPE, DRUM z instrukcji READ i WRITE.

FORTRAN był jednym z pierwszych języków objętych standaryzacją przez ASA (American Standards Association, później USASI - United States Standards Institute, obecnie ANSI - American National Standards Institute).

W latach 1962-1964 opracowano w IBM język FORMAC (FORmula MAnipulation Compiler), jako uzupełnienie FORTRANu IV w za-

kresie manipulacji algebraicznych. Później okazało się, że FORMAC z powodzeniem może współpracować z językiem PL/I.

Rozważając wpływ FORTRANu na inne języki należy dodać, że już pierwsze wersje tego języka (FORTRAN I, FORTRAN II) stanowiły podstawę opracowania szeregu fortranopodobnych kompilatorów: FORTRANSIT (IBM 650), GOTRAN (1620), Honeywell Algebraic Compiler (H-800), ALTAC (Philco 2000), itp.

Do języków specjalnych wzorowanych na FORTRANie zalicza się m.in. QUIKTRAN (on-line time sharing language), GRAF (GRaphic Addition to Fortran), DSL/90 itp.

3. Geneza COBOLu

Opracowywanie COBOLu (COmmon Business Oriented Language) rozpoczęto później niż FORTRANu i ALGOLu.

Mimo dużej złożoności języka pierwszą wersję opracowano stosunkowo szybko, bo w przeciągu jednego roku. Głównym inicjatorem COBOLu była Marynarka Wojenna USA, będąca jednym z największych użytkowników maszyn matematycznych. Pierwsze spotkanie "zainteresowanych" odbyło się 8 marca 1959 roku w Uniwersytecie Pensylwania, zaś 28-29 maja zorganizowano naradę (40 osób) w Departamencie Obrony (DOD) USA. Uczestnikami narady byli przedstawiciele szeregu firm produkujących komputery m.in. IBM, Durroughs, Honeywell, Univac, RCA) oraz przedstawiciele organizatora i instytucji rządowych.

Problem polegał na tym, że instytucje rządowe (a w szczególności United States Navy) posiadały maszyny różnych producentów i istniały w związku z tym kłopoty z wymianą doświadczeń (kadr i oprogramowania) pomiędzy poszczególnymi ośrodkami.

Zaproponowano więc opracowanie języka wspólnego dla różnych typów maszyn, przystosowanego do programowania problemów administracyjnych.

Na wspomnianej naradzie powołano komitety robocze CODASYLu (Committee On DAta SYstems Languages a mianowicie: Short, Intermediate i Long Range Committee.

Prace nad COBOLem rozpoczęto od studiowania takich języków, jak FLOW-MATIC, AIMACO, COMTRAN (COmercial TRANslator). W późniejszych pracach uwzględniono niektóre cechy języka FACT. Już we wrześniu otrzymno pierwszą wersję języka zaś w styczniu 1960 roku została ona zatwierdzona przez Komitet Wykonawczy CODASYLu i w kwietniu opublikowana.

Trzeba podkreślić, że tak szybkie rezultaty osiągnięto przechodząc od dyskusji w szerokim gronie do intensywnej pracy małej liczby specjalistów (w końcowych pracach brało udział jedynie 6 osób !).

Pirwsze kompilatory powstały bardzo szybko, bo już w 1960 roku opracowane zostały przez firmy ADR (Applied Data Research Inc.), Remington Rand i RCA (vide "Zestawienie pirwszych kompilatorów języka COBOL").

COBOL 61 był rezultatem działania specjalnej grupy (special task group), której zadanie polegało na "zrewidowaniu" pierwszej wersji COBOLu. W 1962 roku opublikowano COBOL 61 Extended, do którego dodano, wzorując się na języku FACT, m.in. Report Writer i SORT. Rozpoczęto wówczas również prace na oprogramowaniem pamięci masowych o dostępie selektywnym (wyrywkowym). W 1966 roku opublikowano COBOL-65 zawierający właściwości operowania na zbiorach w tego rodzaju pamięci. Do roku 1967 jedynym posiadaczem kompilatora tej wersji była firma IBM.

Od 1963 roku rozpoczyna się standaryzacja COBOLu przez ASA - American Standards Association. W 1960 roku powołano w ASA /przy ścisłej współpracy z the Business and Equipment Manufacturers Association/ komitet standardów w dziedzinie komputerów i przetwarzania informacji Committee X3.

Komitet ten nie miał za zadanie opracować nowych wersji COBOLu, lecz tylko czuwać nad standaryzacją rezultatów działania komitetu PLC /Programming Language Committee/ działającego w ramach CODASYLu. ASA /USASI, ANSI/ Standard COBOL powinien być więc podzbiorem wersji CODASYL COBOL SPECIFICATION, uznawanej przez użytkowników COBOLu jako jedyna "oficjalna" wersja tego języka.

Organizacja opisu COBOLu jest inna w wydawnictwach ANSI /USASI/ niż CODASYLu. W pierwszym przypadku stosowany jest mianowicie tzw. modułowy opis /a nie rozdziałowy/. Rozróżnia się 8 modułów: nucleus, report writer, sort, segmentation, random access, sequential access, library, table. Każdy moduł dzieli się na 2-3 poziomy zależnie od stopnia złożoności.
ANSI COBOL 68 składa się z dwóch części: COBOLu właściwego i zestawu programów kontrolnych /a set of audit routines/. Zadaniem zestawu kontrolnego jest kontrola dowolnej wersji COBOLu na zgodność z wersją standardową ANSI.
Wersja ta posiada szereg nowości, np. deklaracja SYNCHRONIZED może być używana na poziomie rekordu /01/, występuje instrukcja WRITE...AFTER....POSITIONING.

ZESTAWIENIE PIERWSZYCH KOMPILATORÓW JĘZYKA COBOL

Maszyna	Data wprowadzenia do eksploat.	Wersja	U w a g i
RCA 501	wrzesień 1960	COBOL 60	
ICT 1301		COBOL 60	publ. październik 1960
B - 5000		COBOL 61	Burroughs, wrzesień 1961
NCR 304 GE	grudz.1961		
UNIVAC 490	wrzes.1962		
UNIVAC 1105	wrzes.1961	COBOL 61	HQ. AIR FORCE LOGIS COMMAND
H - 400	wrzes.1962	COBOL 61	HONEYWELL + CSC
IBM 705-III	1961	COBOL 61 CODASYL	
IBM 705-II	luty 1962		
IBM 1401	marzec 1962		
IBM 1410	styczn.1962		wykorzyst. IOCS
MOBIDIC	lip.1962		Sylwania Electric
GAMMA 30	"	COBOL RCA 301	RCA
ICT 1500	1962	RAPIDWRITE	na bazie COBOLu 61
KDF 9	grudz.1963	COBOL 61	English Electric

ZESTAWIENIE PIERWSZYCH JĘZYKOW DO PRZETWARZANIA DANYCH (X - 9)
(poza językiem COBOL)

Język	Data wprow. do eksploatacji	Maszyna	U w a g i
GECOM firma GE	wrzesień 1961	GE-225	dopuszcza TABSOL, COBOL 61, niektóre cechy ALGOLu i FRINGE, 12 przebiegów lub mniej
FACT HONEYWELL + Comp. Sciences	grudzień 1961	H-800	8 przebiegów, 40 oper. min, sort., redagowanie
FLOWMATIC Sperry Rand	1956	UNIVAC I UNIVAC II	
UNICODE Sperry Rand	listop.1959	UNIVAC 1103A	automatyczna segmentacja
JOVIAL	koniec 1961	IBM 7090 Philco 2000 CDC 1604 Q 7	rozwinięcie ALGOLu
NEBULA Ferranti	koniec 1962	ORION ATLAS	
ADAPT Comp. Sciences	październik 1961		
CODEL		ICT 1301	wycofany na rzecz COBOLu

Ponieważ omawiamy genezę COBOLu, warto wymienić konkretne zapożyczenia tego języka oraz oddziaływania na inne języki. Tak na przykład, instrukcja warunkowa IF....THEN oraz PICTURE pochodzą z COMMERCIAL TRANSLATOR (pierwsza specyfikacja tego języka - styczeń 1958), stanowiąc obecnie typowe zwroty COBOLu. Istotnym źródłem dla zespołu roboczego COBOLu (short range committee) był język FLOW-MATIC (pierwsza wersja - 1957), zawierający takie wyrażenia jak: DIVIDE....BY.....GIVING., MOVE.. TO...., IF.....GO.TO... oraz strukturę rozdziałową (rozdzielanie opisu danych od procedur).

Próbą rozszerzenia COBOLu był język IDS (Integrated Data STORE) opracowany w General Electric. Uwzględniał on w szczególności właściwości "łańcuchowego" (Chaining) adresowania w pamięciach dyskowych. Do opisu danych wprowadzone zostało tzw. "pole łącznika" oraz takie deklaracje jak: RETRIEVAL VIA CALC CHAIN, PLACE NEAR n-d CHAIN, PAGE-RANGE, instrukcje: STORE (powoduje utworzenie łańcucha), RETRIEVE,MODIFY, DELETE.
Dla maszyn PDP-8I, PDP-8L firma Digital Equipment Corp. opracowała język DIBOL (DIgital equipment Business-Oriented Language), klasyfikowany jako cobolopodobny (cobol-like) i składający się z trzech komponentów: procesora, systemu operowania danymi (do operowania na zbiorach bez dodatkowego programowania przez programistę piszącego program użytkowy) i monitora, który łączy poprzednie części w jedną całość i umożliwia użytkownikowi łatwe posługiwanie się językiem.

W listopadzie 1963 roku organizacja ECMA (European Computer Manufacturers Association) wydała propozycje COBOLu dla małych komputerów, określone jako COMPACT COBOL. Wersja ta zawiera mniej więcej połowę zwrotów pełnego języka. Kompilator

dla niej opracowała firma ICT, poprzedzając tym prace nad właściwym COBOLem.

Pod względem zakresu COMPACT COBOL odpowiada COBOLOWI B Honeywell (potrzebującemu jedynie 8K znaków pamięci operacyjnej).

W 1962 roku firma ICT opracowała dla maszyny ICT 1500 system RAPIDWRITE, przedstawiający sobą sposób "szybkiego zapisu" programu zewnętrznego z użyciem najistotniejszych (niezbędnych) członów wyrażeń COBOLu. Program pisze się na specjalnych arkuszach z nadrukami. Zadaniem translatora RAPIDWRITE jest przetłumaczenie zdań uproszczonych na zdania pełne COBOLu.

Podobny do COBOLu jest język ADAPT, opracowany w 1961 roku przez Computer Sciences dla IBM 1401.

Koncepcją wykorzystującą dorobki różnych języków jest system GECOM (GEneral COMpiler) opracowany przez General Electric dla maszyny GE 225 i wdrożony w 1961 roku. System ten oparty został o języki: COBOL, ALGOL, FRINGE (do sortowania, wydruków, aktualizacji zbiorów) i TABSOL (do tłumaczenia tablic decyzyjnych).

4. Geneza języka PL/I

PL/I (w zasadzie skrót nieprzetłumaczalny, czasem interpretowany (H-1) jako Programming Language No 1) powstał poniekąd przypadkowo. Otóż w związku z projektowaniem serii 360 firma IBM postanowiła ulepszyć FORTRAN, który posiadał znaczne niedostatki w operowaniu znakami i danymi alfanumerycznymi oraz nie bardzo nadawał się do współdziałania z nowoczesnym systemem operacyjnym maszyn wieloprogramowych.

We wrześniu 1963 roku organizacja użytkowników FORTRANu

noszącą nazwę SHARE i firma IBM powołały, zgodnie z projektem SHARE FORTRAN ADVENCED LANGUAGE DEVELOPMENT COMMITTEE. Wkrótce okazało się, że produkt pracy zespołu wbrew założeniom nie będzie kompatibilny (zamienny) z FORTRANem i należy opracować odrębny nowy uniwersalny język programowania, który nazwano NPL (New Programming Language). Nazwa ta nie utrzymała się, ponieważ National Physical Laboratory (NPL) zgłosiła protest przeciwko używaniu skrótu NPL i nazwę języka zmieniono na PL/I.

W trakcie opracowywania NPL przestudiowano zalety takich języków jak ALGOL, COBOL i JOVIAL (oprócz FORTRANu). Po raz pierwszy zaprezentowano opis języka w marcu 1964 roku , zaś w kwietniu,czerwcu i grudniu podano następne wersje. W 1965 roku powstał pierwszy oficjalny podręcznik programowania.

Do roku 1966 powstało 6 wersji języka: NPL, PL/I:0,1,2,3,4. Dopiero po wszechstronnej krytycznej dyskusji firma IBM zdecydowała się na opracowanie kilku translatorów. Pierwszy translator PL/I został opracowany przez Zakład Badawczy (Laboratory) IBM w Hursley (Anglia) i ukończono go w sierpniu 1966 roku. Zakład ten oraz Vienna Laboratory posiadają duże zasługi również w pracach nad zdefiniowaniem języka.

Pierwsze wersje języka PL/I wzbudziły duże zainteresowanie wśród producentów i już pod koniec 1965 roku mały podzbiór PL/I zwany NICOL I został wdrożony przez Massachusetts Computer Associates na maszynie IBM 7094, BELL Laboratories opracował wersje APL, zaś Massachusetts Institute of Technology (MIT) stworzył MULTICS. Niemniej jednak do 1959 roku język PL/I wdrożono jedynie na małej liczbie maszyn dużej wielkości i był on dostępny w zasadzie jedynie dla członków organizacji SHARE i GUIDE (X - 12).

VI. ZAMIENNOŚĆ JĘZYKÓW I PROGRAMÓW

Zamienność języków i programów (napisanych w różnych wersjach tego samego języka) stanowi kluczowy problem rozwoju programowania. Chodzi o to, by nie dopuścić do sytuacji, iż będzie tyle języków, ile jest (i było) maszyn. Pozytywne rozwiązanie problemu umożliwi wykorzystanie dotychczasowego dorobku programowania i kwalifikacji posiadanej kadry kilkuset tysięcy (w skali światowej) programistów.

Stąd usiłowania standaryzacji języków prowadzone przez organizacje międzynarodowe (ISO, ECMA) i amerykańskie (CODASYL, USASI) oraz wysiłki niektórych rządów (np.USA w przypadku COBOLu). Działalność ta, z takich czy innych powodów, nie jest jeszcze należycie uwzględniana przez producentów, posiadających swoje własne ambicje i dla dobra walki konkurencyjnej (a niekiedy i postępu), stale wprowadzających innowacje, nie zawsze do końca przemyślane. Inaczej nie mielibyśmy paru tysięcy języków wyższego rzędu, z których wiele stawia sobie takie same zadania, częściowo dubluje składnię oraz .... błędy poprzedników. Ponadto "oficjalne" (teoretyczne) wersje niektórych języków tak abstrahują od technicznej i organizacyjnej strony przetwarzania (np. ALGOL w zakresie wejścia-wyjścia), że ich wdrożenie jest możliwe jedynie poprzez dopracowanie przez producentów i instytucje software'owe.

Najpoważniejszym czynnikiem różnicującym wersje są różnice w wyposażaniu komputerów. W przypadku bardziej złożonych języków (jak COBOL, PL/I), wymagających pojemnych pamięci operacyjnych, często następuje "obcinanie" pełnych wersji, przystosowując je do aktualnych możliwości sprzętu.

W warunkach dużej dynamiki rozwoju konstrukcji i programowania standaryzacja jest bardzo utrudniona. Można powiedzieć, że jeśli stosuje się ją przedwcześnie, to "konserwuje" ona elementy błędne i mało rozwojowe, hamując tym samym dalszy rozwój dziedziny.

Na pewno zamienność absolutna jest nie do osiągnięcia. Potrzebny jest kompromis ukształtowany pod wpływem kosztu przedsięwzięcia, efektywności kompilatorów itp.

Wydaje się, że standaryzacja programowania powinna być poprzedzona standaryzacją pewnych podstawowych elementów hardware'owych jak np. standaryzacja taśm magnetycznych, rodzajów kodów znaków (łącznie z problemem starszeństwa) itp. Ponadto więcej uwagi należałoby poświęcić rozwojowi technik reprogramowania z maszyny na maszynę (emulacja, symulacja itp.).

1. Koncepcja uniwersalnego języka pośredniego

Wzrost liczby języków programowania oraz typów maszyn powoduje znaczne zwiększenie pracochłonności opracowania translatorów. Jest ona proporcjonalna do iloczynu tych dwóch elementów (tj. liczby języków przez liczbę typów maszyn).

Gdyby udało się we wszystkich maszynach zastosować jeden uniwersalny język pośredni, to liczba translatorów tego języka równałaby się liczbie maszyn (LM). Z kolei wymagane byłoby przetłumaczenie każdego języka użytkowego na uniwersalny język pośredni i do tego celu potrzeba byłoby tyle translatorów, ile jest języków (LJ). Ogólna liczba translatorów równa się więc sumie LM + LJ, podczas gdy w pierwszym przypadku występował iloczyn. Różnica jest więc oczywista (nie tylko arytmetycznie), nawet jeśli uwzględnimy, że to liczbowe ujęcie jest niedokładne,

ponieważ nie uwzględnia różnic w pracochłonności opracowania translatora języka uniwersalnego i translatora języka specjalistycznego (w iloczynie oba argumenty przedstawiają ilości języków specjalistycznych, podczas gdy w sumie argumenty są mieszane).

Koncepcja języka uniwersalnego UNCOL (Universal Computer Oriented Language) pojawiła się pod koniec lat pięćdziesiątych (lata 1958-1961) w artykułach Convey'a M.E., Steel'a T.B., Stronga J. Niestety, języka takiego nie opracowano, a na przeszkodzie stanął zakres pracy, jaki należało wykonać: uzgodnienie, jakie elementy mają być uniwersalne, wybór poziomu programowania oraz aktualizacja pierwszych wersji w związku ze zmianami konstrukcji maszyn.

Koncepcja języka pośredniego nie została jednak całkowicie zarzucona. W charakterze takiego języka w ramach poszczególnych firm występuje zwykle język symboliczny (np. język PLAN dla serii 1900). Niekiedy językiem pośrednim jest język wyższego rzędu, np. FORTRAN stosowany jest jako język pośredni dla APT. Programy APT są za pomocą specjalnego podzbioru APT redagowane wstępnie (zmiana składni) na formę akceptowaną przez kompilator FORTRANu (G - 3).

2. Emulacja

Problemy zamienności języków i programów są równocześnie problemami zamienności komputerów. Stąd też są rozwiązywane m.in. metodą opartą zarówno o środki programowe jak i hardware'owe, zwaną emulacją. Środkiem programowym jest specjalny program, zaś hardware'owym - specjalna pamięć i dodatkowe rejestry. W "emulator" wyposaża się zwykle maszynę nowszą (zakupioną na

miejsce poprzedniej), po to, by mogła akceptować dotychczasowe programy opracowane dla maszyny wycofywanej (starszej). Emulacja nie zapewnia idealnej zamienności: doprowadza do zamienności kody operacji itp., a więc w przypadku zastosowania różnych urządzeń peryferyjnych i systemów operacyjnych pewne przepracowanie programów będzie konieczne.

Oto "zamienne" maszyny firmy IBM w stosunku do serii 360 (wg X - 13):

1620	model 30
1401, 1440, 1460, 1410, 7010	40
7070/7074	50
705/7080 709/7090	65

Hardware'owym elementem emulacji dla serii 360 jest pamięć pasywna ROM-Read Only Memory (zwana też ROS - Read Only Storage), zawierająca mikroprogramy. W miejsce klasycznego wykonywania instrukcji, pamięć ta powoduje otwieranie i zamykanie szeregu elektronicznych "bramek" w systemie obiegu impulsów. Elementem software'owym jest program emulatora, umieszczony w pamięci ferrytowej, służący do wykonywania programu obcej maszyny (również znajdującego się w tej pamięci) w konwencji serii 360. "Obcy" program znajduje się pod kontrolą ROM dopóki nie zostanie napotkana instrukcja, która nie może być hardware'owo interpretowana. Wtedy sterowanie przekazywane jest do programu emulatora.

Pod względem funkcjonalnym emulacja podobna jest do symulacji. Wykonywana jest jednak efektywniej dzięki mikroprogramom, które wykonują funkcje: interpretacji instrukcji, dekodowania adresu, wybierania zawartości adresu

i wykonania wskazanej operacji. Czynności te są najbardziej pracochłonne i pochłaniają dużo czasu przy symulacji.

Wykonanie programu techniką emulacji wymaga większej pamięci operacyjnej (do przechowywania specjalnego programu), zaś efektywność programu adaptowanego na serię 360 powinna być nie mniejsza, niż na maszynie pierwotnej (mimo, iż możliwości serii 360 nie są w pełni wykorzystywane).

Pamięć ROM zbudowana być może z różnych elementów. Model 30 serii 360 w charakterze ROM posiada specjalne miedziane karty perforowane. W innych maszynach (np.firmy Burroughs) ROM zbudowana jest na układach scalonych.

3. Symulacja

Słownik IFIP (V - 1) pod hasłem A2 podaje następującą definicję symulacji: "Symulacja. Reprezentowanie pewnych właściwości zachowania się jednego systemu poprzez działanie innego systemu". Symulacja działa podobnie jak emulacja lecz bez środków hardware'owych. W pamięci przechowywany jest program obcy i program symulatora. Każdy rozkaz może być interpretowany i wykonywany, i w tym przypadku nie trzeba w pamięci przechowywać całego programu obcego.

Firma ICL opracowała następujące symulatory (dla serii 1900):

- Elliott 803 Simulator (xME4, xME8).
  Przeznaczony jest do symulacji emc Elliott 803 z 4 K PAO (lub 8K). Na maszynie 1905 programy Elliott 803 są wykonywane przeciętnie dwukrotmie szybciej (przy czym zapotrzebowanie na PAO wynosi 9600 słów lub 17792).
- Pegasus Simulator (xMP4 dla 4K Pegasus i 12K 1900
  xMP7 7K " 18K ").

Symulator ten staje się niepraktyczny w przypadku używania ręcznych kluczy (przełączników na pulpicie) przez programy maszyny Pegasus.

4. Translacja z języka na język

Jest to najmniej efektywny sposób przerabiania programu z języka na język, ponieważ polega na "mechanicznym" przekształcaniu instrukcji, bez uwzględniania specyfiki nowej maszyny. Wymaga też większej pamięci, ponieważ najpierw tłumaczony jest cały program, a dopiero później wykonywany.

Jako przykłady translatorów wymienić można następujące prace firmy ICL:

- ATLAS ALGOL na 1900 ALGOL (xAC2),
- MPL (1300) na PLAN 1900 (xJP3).

  W xJP3 rozkład danych na bębnie jest "symulowany" w pamięci ferrytowej. Nie podlega tłumaczeniu E funkcja oraz zmodyfikowane instrukcje I. Zakłada się, że program pisany w MPL jest bezbłędny (nie ma kontroli diagnostycznej).

Z publikacji (0 - 1) znana jest koncepcja systemu XACT (X - Automatic Code Translation, przy czym X oznacza dowolną maszynę), opracowanego od 1961 roku przez Celestron Associates, Inc. na bazie pary maszyn 7090 - 1604. Maszyna 7090 jest maszyną źródłową (source), zaś 1604 - docelową (target). Problem polega na tym, by tak przekształcić programy 7090, by mogły być realizowane na 1604. Ocenia się, że opracowanie translatora dla tej pary wymaga 10-15 osobolat pracy programistów. Ciekawostką jest to, że translacja przebiega (przynajmniej częściowo) nie na maszynie docelowej, lecz na maszynie źródłowej (!), do której wprowadza się poza programem również opis

danych (liczb). W pierwszej fazie kompilacji tworzony jest maszynowo niezorientowany opis funkcji do wykonania, zaś w drugiej powstaje program w kodzie maszyny docelowej.

5. Uwagi ogólne

Można wyróżnić następujące typy tłumaczeń z języka na język:

1/ języka maszynowego jednej maszyny na maszynowy język innej maszyny (szczególne trudności występują tutaj m.in. przy tłumaczeniu programów wejścia-wyjścia),

2/ języka maszynowego na język zewnętrzny (mamy tutaj do czynienia z dekompilacją, czyli procesem odwrotnym do kompilacji)

3/ języka zewnętrznego na wewnętrzny,

4/ języka zewnętrznego na język zewnętrzny,

5/ translatora na translator (być może ten rodzaj tłumaczenia ma większe perspektywy rozwojowe, niż tłumaczenie konkretnych programów użytkowych).

Automatyczne tłumaczenie wymaga wstępnej selekcji materiału, odrzucającej zwroty nieprzetłumaczalne i błędne. Dlatego też systemy tłumaczące powinny posiadać organizację umożliwiającą programoście ingerencję w proces tłumaczenia oraz zapewniającą sygnalizację elementów błędnych lub do ręcznej konwersji.

Istnieje szereg sposobów postępowania w przypadku napotkania trudności w tłumaczeniu. W systemie tłumaczenia z języka symbolicznego IBM 7090 na język maszynowy IBM 7040, nieprzetłumaczalnym bezpośrednio zwrotom przydziela się "fikcyjne" makrorozkazy (ekstrakody), które są rozszyfrowywane dopiero w następnych przebiegach. Oto inne rodzaje trudności (G-4):

a/ cały program lub bloki programów są całkowicie nieprzetłumaczalne - ma to miejsce wtedy, gdy konstrukcja maszyny

jest częścią algorytmu (np. program testowania pamięci operacyjnej maszyny CDC 1604),

b/ samomodyfikacja programu (dynamiczna struktura programu),

c/ wyrażenia idiomatyczne (gdy niektóre rozkazy lub ich sekwencje mają sens jedynie wtedy, gdy rozpatrywane są na tle całego programu),

d/ różnice konstrukcyjne pomiędzy maszynami (np. słowo 24-bitowe i 36-bitowe).

6. Zamienność programów pisanych w COBOLu

Wersje oficjalne (np. ogłoszone przez CODASYL) COBOLu są co prawda maszynowo niezależne, lecz stanowią one dla twórców kompilatorów jedynie bazę koncepcyjną, do której wprowadzają własne pomysły, utrudniające zamienną eksploatację programów.

Pomysły te stanowią odbicie specyfiki hardware'owej lub też są po prostu innowacją formalną w składni języka (np. w wersji COBOLu na maszynę ZAM41 opracowanej przez Instytut Maszyn Matematycznych w Warszawie).

Rozważając problemy zamienności programów, trzeba wspomnieć o przypadkach obiektywnej niezamienności, np. gdy program napisany w wersji dyskowej (z zastosowaniem indeksowania lub randomizacji) chcemy eksploatować na maszynie wyposażonej tylko w taśmy magnetyczne.

Już od samego początku podejmowano starania, by COBOLowi zapewnić wysoki stopień zamienności. Tak np. w grudniu 1960 roku demonstrowano zamienność programów COBOLu pomiędzy maszynami RCA 501 i UNIVAC II (B-10). Później nadzór nad składnią tego języka przejęły takie organizacje międzynarodowe /poza

CODASYLem), jak ASA/USASI, ANSI/ISO, ECMA.

Zakres wersji kompilatora COBOLu i jego efektywność zależą przede wszystkim od wielkości dostępnej (po odjęciu obszaru dla systemu operacyjnego itp.) pamięci operacyjnej.

W celu sklasyfikowania poszczególnych wersji firmy IBM i Honeywell wprowadziły pojęcie poziomu (level) języka: Honeywell - B, D, H, I, F, zaś IBM - E, F. Każdy poziom języka różni się zakresem słownika dopuszczalnych zwrotów.

Firma ICL (seria 1900) usiłowała natomiast stosować jedną wersję języka, zmniejszając efektywność kompilacji przy mniej dogodnych konfiguracjach (użycie wielokrotnego wzywania segmentów wymiennych).

Dla programisty najważniejszym problemem jest niewątpliwie zakres czynności, jakie ma wykonać, chcąc zaadoptować program COBOLu pisany dla innej maszyny. Brakuje tego rodzaju informacji w publikacjach krajowych i obcych. Poniżej podamy pewne zasady oparte o własne doświadczenia w programowaniu na maszyny ICL 1900, ICL System 4-50 oraz ZAM-41:

1/ porównanie konfiguracji maszyn (w szczególności pojemności pamięci, rodzajów urządzeń wejścia-wyjścia i pamięci masowej),

2/ porównanie listy wyrażeń obu kompilatorów oraz sprawdzenie zakresów działania takich samych zwrotów (m.in. dotyczy to deklaracji COMPUTATIONAL, DISPLAY-n ściśle związanych z postacią liczb w maszynach - postać binarna, pojedyncza lub podwójna precyzja, stały lub zmienny przecinek, obliczenia w angielskim dziesiętnym systemie walutowym itp. W COBOLu i Honeywell I deklaracje COMP, COMP-1 posiadają identyczne znaczenie),

3/ sprawdzenie, czy program jest segmentowany i ile miejsca zajmuje w PAO w dotychczasowej posegmentowanej postaci,

4/ sprawdzenie, czy program posiada wstawki napisane w innych językach (w języku symbolicznym lub wyższego rzędu np. w FORTRANie lub języku symbolicznym dla COBOLu Honeywell). Jeśli takie wstawki wystąpują, to należy zapewnić odpowiednik wstawki w "macierzystym" języku wstawek i wprowadzić odpowiedni aparat formalny związany z użyciem instrukcji ENTER, CALL itp. Można oczywiście zamiast wstawki napisać fragment programu w języku COBOL.

5/ porównanie sposobów wejścia danych; należy ustalić, czy występuje zapis pozycyjny i niepozycyjny. Jeśli stosowany jest zapis niepozycyjny, to należy sprawdzić, jakie znaki są stosowane w charakterze standardowych ograniczników.

6/ porównanie treści metryk standardowych (w szczególności dotyczy to zbiorów na dyskach i taśmach magnetycznych),

7/ porównanie parametrów drukarek wierszowych (długość wiersza, znaki specjalne, funkcje poszczególnych kanałów taśmy sterującej itp.).

Jako przykład firmowych odrębności można podać zwroty występujące w COBOLu Honeywell:

- STOP "JOB" (zakończenie ciągu zadań) obok STOP RUN (np. w celu przejścia do podprogramu),
- CANCEL (zwolnienie obszaru pamięci operacyjnej, zajętego przez podprogram),
- LOAD (załadowanie podprogramu do pamięci operacyjnej).

Można powiedzieć, że ani jednego programu (z wyjątkiem trywialnych) COBOLu nie można mechanicznie przenieść na inny typ maszyny bez dokonania pewnych zmian.

Najbardziej zmienną częścią COBOLu jest ENVIRONMENT DIVISION, a więc rozdział opisujący konfigurację maszyny, najmniej - co nie oznacza, że w ogóle nie ulega zmianom - PROCEDURE DIVISION. DATA DIVISION jest z omawianego punktu widzenia czymś pośrednim: mogą wystąpić różnice w opisie metryk zbiorów, w klauzulach typów danych, w długości wiersza na tabulogramie itp.

Jeśli chodzi o część proceduralną, to nie we wszystkich wersjach występują bardziej złożone warianty instrukcji PERFORM (z AFTER), zwrot DEPENDING ON (w połączeniu z GO TO), MOVE CORRESPONDING itp. Pewien kłopot sprawić może uzyskanie takiej samej dokładności wyniku przy operacjach arytmetycznych. Do innych rezultatów doprowadzić mogą operacje porównywania pól znaków alfanumerycznych w sytuacji, gdy porównywane maszyny posiadają inną sekwencję starszeństwa znaków (np. w niektórych maszynach litery "są" "większe" od cyfr, w innych - mniejsze). Odrębne zagadnienie stanowi wpływ systemu operacyjnego maszyny, na którą opracowano program, stosowanie pakietów REPORT WRITER, procedur diagnostycznych typu READY TRACE, RESET TRACE, EXIBIT....

Pomocą w ocenie zamienności kompilatorów mogą być specjalistyczne generatory programów, np. CCVS - Cobol Compiler Validation System opracowany przez Air Force USA (H - 7), dostarczające odpowiedzi na najistotniejsze pytania:

a/ czy kompilator spełnia wymogi poziomu, do którego został zaliczony, tzn. czy rzeczywiście wykonywane są wszystkie funkcje przewidziane przez standard poziomu,

b/ czy program wewnętrzny otrzymany po kompilacji dostarcza wyniki odpowiadające określonym standardom.

Analiza zamienności różnych kompilatorów stanowi intersujący

Przykłady różnic pomiędzy wersjami języka COBOL

Lp.	Element	ICL seria 1900	ICL System 4-50	IMM ZAM 41	U w a g i
1.	Nazwy danych i nazwy paragrafów	1-szy znak musi być literą	w nazwie danych musi być co najmniej jedna litera, ale nie musi być 1-szy znak, nazwa paragrafu może być liczbą	1-szy znak musi być literą	
2.	Nazwy urządzeń np. czytnika kart	CARD-READER	UNIT - RECORD C-READER	INPUT /nr/	IBM 360: UNIT-RECORD 1402R
3.	Deklaracje np.				
	COMPUTATIONAL	liczba dziesiętna kodowana dwójkowo /binary decimal/ 1/	postać binarna	—	1/ nie jest to typowa postać liczby dziesiętnej kodowanej dwójkowo, ponieważ co prawda 1 znak zajmuje 4 bity, lecz 2 znaki - 7b, 3 znaki - 10 bitów itp.
4.	COMPUTATIONAL-1	liczba binarna	liczba zmienno-przecinkowa	—	
	FIXED	—	—	liczba całkowita binarna	
	FLOAT	—	—	liczba zm. przec. w postaci binarnej	

i obszerny problem, kwalifikujący się do odrębnego opracowania. Ze względu na ograniczoną objętość niniejszej pracy, nie możemy poświęcić temu więcej miejsca.

7. Zamienność programów pisanych w FORTRANie

Podstawową wskazówką zamienności jest rodzaj podstawowej wersji: FORTRAN, FORTRAN II, FORTRAN IV. Ponadto w ramach tych samych wersji mogą występować znaczne różnice, np. kompilatory FORTRANu dla IBM 1620, IBM 650 nie zawierały deklaracji FORMAT i nie dawały możliwości wezwania podprogramów.
Następujące zwroty FORTRANu IV nie występowały w FORTRANIE II (G - 2): DATA, BLOCK DATA, NAMELIST, logiczne IF, numery zdań w charakterze argumentów podprogramów, wielokrotne wejścia do podprogramów.

Oprócz tego, istnieją następujące różnice w instrukcjach wejścia-wyjścia:

FORTRAN II	FORTRAN IV
READ INPUT TAPE j,i,parametry	READ (j,i) parametry
WRITE OUTPUT TAPE j,i, parametry	WRITE (j,i) parametry
READ TAPE j, param.	READ (j) param.
READ DRUM i,j, param.	READ (k) param.
WRITE DRUM i,j, param.	WRITE (k) param.

Opracowany został translator SIFT (Share Internal FORTRAN Translator) do modyfikacji programów FORTRANu II na programy FORTRANu IV.

8. Zamienność programów pisanych w ALGOLu

Kłopoty z zamiennością programów dotyczą przede wszystkim procedur wejść-wyjść, opisywanych nie przez wersje teoretyczne, lecz przez translatory. Pogląd ten zilustrujemy w poniższym zestawieniu.

Lp.	Procedury	ICL 1900	ICL system 4-50	ODRA 1204	GIER ALGOL	Ural 2
1.	Przydział urządzeń	select input/n/ select output/n/	OPEN/n/ SET /n,m/	set input/n/ set output /n/		
2.	Zwolnienie urządzeń	free input free outpur	CLOSE /n/			
3.	Operacje czytania, pisania itp.	print /E,m,n/ output /E/ write boolean/E/ write text /.../ copy text /.../ N: = read	WRITE /n,FORMAT .....n-d/ n-d : =READ/n/	print /n_1...n_n/ ininteger inreal inchar print/n_1..n_n/ outchar line /n/ read /n_1...n_n/	write writetext write cr write char output outtext outchar outsum input inone inchar setchar lyn typein typechar	writetexst outtext outer writeer outchar outelear input inone typein

VII. ROZWÓJ PROGRAMOWANIA A ROZWÓJ KONSTRUKCJI MASZYN

Programowanie i konstrukcja stanowią dwa wzajemnie warunkujące się elementy składowe tego, co nazywamy komputerem. Rozwój tych elementów nie odbywa się jednak w idealnej harmonii. W początkach rozwoju komputerów, kiedy dopiero wykrywano ich możliwości, sprzężenie zwrotne pomiędzy tymi elementami było szybsze. Rozwiązania przebiegały w stosunkowo małej skali (niewiele koncepcji, mało indywidualnych konstrukcji, brak walki konkurencyjnej dużych firm) stąd nietrudno było o koordynację i szybkie rezultaty. W miarę rozkręcania wyścigu o rekordy szybkości działania coraz mniej uwagi udzielano sprawom programowania. Ponadto opóźnienie tego ostatniego wynikało z naturalnej kolejności opracowania komputera /mimo pewnej równoległości prac, software opracowywuje się w zasadzie dla gotowych konstrukcji). Ponieważ cykl opracowania software'u jest o wiele dłuższy od konstrukcyjnego przygotowania, często nie nadążano po prostu z dopracowaniem go przed zejściem nowego modelu komputera z taśmy produkcyjnej. Wg I.L. Auerbacha (A - 2) system operacyjny IBM 360 jestopóźniony o 5 lat w stosunku do rozwiązań konstrukcyjnych.

W pewnych przypadkach konstrukcje są wyprzedzane przez pomysły software'owe (np. pamięci asocjacyjne).

Tworzy się maszyny IV generacji o wysokim stopniu modularności, obejmującej oprócz dotychczas stosowanych procesorów wejścia-wyjścia, również niezależne procesory centralne i niezależne (o równoczesnym dostępie) moduły pamięci operacyjnej. Stwarza to nowe problemy zarówno hardware'owe jak i software'owe.

A. Spróbujmy prześledzić rozwój urządzeń pamięciowych, stanowiących główną przesłankę rozwoju programowania (służą do przechowywania programów użytkowych, translatorów, systemu operacyjnego-dyrygenta, biblioteki programów itp.):

1. Początkowe do przechowywania informacji używane były przerzutniki (np. w ENIACu - przerzutniki lampowe - jako pamięć operacyjna). Następnie do połowy lat pięćdziesiątych stosowano pamięci dynamiczne na rtęciowych (po raz pierwszy - 1949 EDSAC) oraz niklowych (od 1951) liniach opóźniających. Można wspomnieć również o pamięciach elektrostatycznych na lampach kineskopowych (po raz pierwszy budowanych przez Prof. Williamsa w Manchester University oraz w MIT).
   Pamięci dynamiczne i elektrostatyczne, aczkolwiek niedoskonałe (wolne i małopojemne - 512, 1024 słów) stanowiły w porównaniu z ENIACem duży krok naprzód, ponieważ umożliwiały przechowywanie programów w pamięci. Wystąpiła więc możliwość stosowania instrukcji skoków oraz wzywania podprogramów.
2. Jeszcze przed pamięciami na liniach opóźniających, bo mniej więcej od 1947 roku używano bębna magnetycznego, niemniej jednak z powodu dużego czasu dostępu nie bardzo nadawał się na pamięć operacyjną.
   Oto pierwsze zastosowania bębna (koniec lat 40-tych):
   - SEC (Simple Electronic Computer), zbudowany na uniwersytecie londyńskim,
   - ERA 1101, zbudowana w 1950 roku przez Engineering Research Associates,
   - MARK III, MARK IV, zbudowane w Harvard University w latach 40-tych.

Pierwszą maszyną używającą bębna w charakterze pamięci pomocniczej była maszyna WHIRLWIND I, której budowę zakończono w 1951 roku w MIT. Pamięć bębnową posiadała maszyna IBM 650, będąca najpopularniejszą maszyną na świecie w latach 50-tych. Maszyna ta posiadała środki programowe (system SOAP) do optymalizacji wybierania danych z bębna.

3. Głównym rodzajem pamięci operacyjnej, niezastąpionym od lat kilkunastu, jest pamięć ferrytowa. Teoretyczne artykuły na temat tego typu pamięci zaczęły pojawiać się dopiero na początku lat pięćdziesiątych. Niemniej jednak już w 1951 roku firma Jacob Instrument Company wprowadziła tę pamięć do maszyny Jain Comp D-1 (M - 3). Poważne prace badawcze nad pamięciami ferrytowymi prowadzone były w MIT od mniej więcej 1953 roku.

Firma IBM użyła po raz pierwszy rdzeni ferrytowych w maszynach 704 i 705 (około 1955 r.). W marcu 1956 roku zastosowała je firma Remington Rand w maszynie UNIVAC 1103A.

Wprowadzenie pamięci ferrytowych zapewniło maszynom dużą szybkość (mniej więcej tysiąckrotnie wyższą niż w przypadku bębna), umożliwiło zastosowanie podziału czasu (a więc i pracy wieloprogramowej). Dzięki dużym pojemnościom (od kilku tysięcy do kilku milionów słów) zaistniała możliwość użycia złożonych języków programowania i systemów operacyjnych. Ze względu na ciągłe wzbogacanie translatorów (m.in. o powiązania z systemem operacyjnym) uważa się, że obecny "standard" 32 K, w maszynach IV generacji zostanie podwyższony do 64 K.

Pamięci ferrytowe budowane mogą być "wyrazowo" lub "znakowo" względnie posiadać mogą konstrukcję mieszaną pozwalającą na bezpośrednie adresowanie zarówno znaków (lub byte'ów) jak i

słów (komórek). Do niedawna stosowano najczęściej jednostki pamięciowe 24-bitowe i 8 (lub 6-)-bitowe. Ostatnio obserwujemy tendencje pośrednie, a mianowicie wprowadzanie pamięci o jednostce adresowania 16-bitowej. Jest to kompromis pomiędzy dwoma sprzecznymi walorami. Krótsze jednostki adresowania umożliwiają sprawniejsze manipulacje danymi (co jest szczególnie przydatne przy tłumaczeniach) oraz lepsze wykorzystanie pamięci (mniej pustych bitów). Pamięci "wyrazowe" natomiast zapewniają wyższe szybkości działania dzięki równoległemu przesyłaniu większej liczby bitów.

Podkreślaliśmy znaczenie pojemnej pamięci operacyjnej. Istnieją środki hardware'owe-programowe, określane jako tzw. stronicowanie[1/] (paging), które umożliwiają programiście traktowanie pamięci, jakby była nieskończona. Na przykład, stronicowanie w maszynie IRIS 80 pozwala na pisanie programów o wielkości 16 milionów słów przy maksymalnej fizycznej wielkości pamięci 1 ml słów. Nie chodzi tutaj o klasyczną segmentację programów. Programista posługuje się teoretycznymi obszarami pamięci, zwanymi "stronami", podczas gdy grupa danych, znajdująca się fizycznie w pamięci, nazywana jest "blokiem" (niekiedy stosuje się odwrotne przyporządkowanie pojęć strona i blok). Problem polega na tym, by stronie przyporządkować odpowiedni blok, tj. dokonać konwersji adresu strony na adres bloku. Jeśli "strony" nie ma aktualnie w pamięci, to wtedy następuje automatyczne jej przesłanie bez udziału programisty. Do przechowywania informacji o aktualnie używanych stronach służą specjalne rejestry (w ICL 1906A jest ich 16), które wszystkie mogą być równocześnie angażowane. Jeśli żądany adres strony nie występuje w żadnym z nich, wtedy szuka się jej odpowiednika

---

x/Używa się też terminu "paginowanie" (Przyp.Red.).

za pomocą specjalnej tabeli (special-look-up-table).

Duże nadzieje wiąże się z pamięciami budowanych na układach scalonych. W szczególności dotyczy to tzw. pamięci notatnikowych (scratch-pad memory). Czas dostępu do takiej pamięci jest minimalny: w maszynie SDS Sigma 7 - 150 ns (z roku 1966), w maszynie IRIS-80 - 60 ns (dla rejestru 16-bitowego).

4. Podstawowe znaczenie dla przetwarzania danych ekonomiczno-administracyjnych posiadają masowe pamięci pomocnicze.
Za datę pierwszego użycia taśmy magnetycznej zwykło się podawać rok 1954, kiedy to zastosowano ją w maszynie UNIVAC I. Niemniej jednak już w 1949 roku (!) została ona wprowadzona do maszyny BINAC, zbudowanej tak, jak UNIVAC I przez Eckerta i Mauchly'ego (M - 3).

W maju 1955 roku firma IBM poinformowała o stworzeniu dysku magnetycznego (model 305). Pojemność dysku wynosiła 5-6 milionów znaków, zaś czas dostępu był dość znaczny (200-600 mlsek).
W 1957 roku wprowadzono (K-2) pamięć karuzelową (rodzaj pamięci taśmowej), zaś w 1961 roku zastosowano karty magnetyczne.

Prace nad pamięciami masowymi zmierzały w kierunku zwiększenia pojemności, zmniejszenia czasu dostępu i obniżenia kosztu.

Pojawienie się pamięci masowych postawiło przed programistami nowe zadania. W przypadku pamięci taśmowych opracowano szereg algorytmów sortowania (np.wg rozkładu Fibbonaci'ego, metodą kaskadową i polifazową) oraz ustalono tzw. standardowe metryki zbiorów. Pamięci dyskowe okazały się trudne do opanowania od strony programowej ze względu na różnorodność sposobów przechowywania i wybierania informacji (sposoby te podajemy w terminologii angielskiej w celu uniknięcia nieporozumień:

serial, sequential, self-indexing, partial indexing, full indexing, randomizing).

Eksploatacja pamięci dyskowych wymaga oprogramowania "serwisowego", obejmującego m.in. procedury reorganizacji zbiorów (mających na celu likwidację nadmiarów -overflow - na poszczególnych sektorach), programy "oczyszczania" dysku, dumping na taśmę magnetyczną itp. Koncepcja cylindra (seek area) stwarza możliwości optymalizacji (z punktu widzenia czasu dostępu) rozmieszczenia zbiorów dotyczących tego samego programu poprzez lokowanie ich na jednym cylindrze, tj. obszarze poszukiwań. O trudnościach software'owego opracowania pamięci dyskowych świadczy np. słabość dotychczas stosowanych algorytmów randomizacji (nierównomierne obciążenie sektorów) oraz fakt, że dopiero w 10 lat po stworzeniu pierwszej pamięci dyskowej powstała "oficjalna" dyskowa wersja COBOLu, którą z kolei do roku 1967 potrafiła wdrożyć jedynie firma IBM.

5. Mówiąc o wzajemnym oddziaływaniu hardware'u i software'u nie sposób pominąć konkretnego rezultatu tych oddziaływań, a mianowicie mikroprogramowania.

Posiadać ono będzie szczególne znaczenie, jako tzw. firmware, w maszynach IV generacji, kiedy to użytkownik będzie mógł sam tworzyć wymienne pakiety mikroprogramów (stosując np. specjalne karty dziurkowane o trójkątnych otworach) i w ten sposób dostosowywać niektóre parametry konstrukcyjne do konkretnych zastosowań (A - 2).

Pierwsze publikacje na temat mikroprogramowania pojawiły się już w 1953 roku (L - 3). W 1956 roku na konferencji w MIT przyjęto niejako oficjalną nazwę mikroprogramowanie dla określenia techniki budowy układów logicznych, realizujących ope-

racje za pomocą sekwencji impulsów. Każdy impuls można nazwać mikrorozkazem. Mikroprogramy mogą być realizowane "pionowo" tzn. jako sekwencja następujących po sobie mikrorozkazów lub "poziomo", tj. kiedy programista decyduje o otwarciu odpowiednich bramek poprzez zastosowanie odpwiedniej struktury bitowej w kodach operacji (I - 16). Ważną zaletą mikroprogramów jest to, że realizacja ich nie "obciąża" jednostki centralnej, ponieważ działają one niejako automatycznie i co ważniejsze, mogą przebiegać równocześnie (in one pulse time).

Mikroprogramy są przedstawiane za pomocą pamięci pasywnych (read-only memory, non-volatile store, fixed store). Jedną z pierwszych maszyn programowanych był EDSAC II, w którym mikroprogramowanie realizowane było poprzez odpowiednie łączenia w pamięci ferrytowej. Znaczne mikroprogramowanie posiadała maszyna TX-0, zbudowana w Lincoln Laboratories M I T (w maszynie tej wykorzystano dorobek wspomnianej konferencji na temat mikroprogramowania).

Mikroprogramowanie stanowi ważny element ujednolicania architektury maszyn (np.modeli serii IBM 360), wchodząc w skład środków emulacji (rozdział VI, pkt.2).

6. Krokiem zmierzającym do wyeliminowania strat pamięci potrzebnej na przechowywanie adresów i ułatwienia techniki programowania zadań w zakresie wyszukiwania informacji jest propozycja pamięci asocjacyjnej. Koncepcję pamięci asocjacyjnej po raz pierwszy sformułowali prawdopodobnie Newell A., Shaw J.C., Simon H.A. (C - 3), wykorzystując metodę pośredniego adresowania.

Istnieją różne koncepcje organizacji i konstrukcji tego typu pamięci (K - 1).

Hardware'owe rozwiązanie pamięci asocjacyjnej posiada znaczne walory w porównaniu z tzw. software'owym typem pamięci asocjacyjnej. Każda komórka posiada możność wykonania operacji porównania ze z góry zadanymi wartościami (oczywiście bez angażowania urządzenia arytmetycznego). Poszukiwanie określonych wartości odbywa się bez pośrednictwa adresów i wykonywane jest równocześnie we wszystkich komórkach. Rezultat porównań zapamiętywany jest w spacjalnych wskaźnikach. Koncepcja działania tego typu pamięci szokuje oryginalnością i efektywnością. Jest z technicznego punktu widzenia trudna do zrealizowania.

Sposób software'owej organizacji pamięci asocjacyjnej polega na zastosowaniu tzw. list adresowych (łańcuchowego adresowania).

Ogólnie rzecz biorąc, pamięć asocjacyjna adresowana jest zawartością, tzn. informacja odszukiwana jest w pamięci na podstawie treści a nie adresów. Poza usprawnianiem procesów wybierania, zalety pamięci asocjacyjnej wykorzystywane być mogą do przyśpieszenia sortowania (dzięki ominięciu pracy sekwencyjnej).

Nie tylko pamięć asocjacyjna jest próbą przełamania adresowości. Podobną rolę spełnia również pamięć stosowa (push-down memory, stack stare).

§. 1. Jedną z podstawowych charakterystyk języków wewnętrznych i symbolicznych jest adresowa struktura rozkazów. Liczba części adresowych rozkazu waha się od jednego do pięciu (najczęściej spotyka się maszyny jedno i dwu-adresowe) i określana jest przez konstrukcję każdej maszyny. Maszyny UNIVAC I, II, seria IBM 700, większość maszyn typu

"Princeton Class" (a więc opracowanych wg propozycji Johna von Neumanna) były maszynami jednoadresowymi. Inne, takie jak np. UNIVAC Scientific ERA-1103, należały do maszyn dwuadresowych. Nadmienić można, że rozróżnia się dwa rodzaje systemu dwuadresowego: jeden, gdzie ukazywany jest adres argumentu i adres rezultatu, i drugi, tzw. 1½1 adresowy, gdzie ukazywany jest adres argumentu i adres następnego rozkazu. Adresowe rozkazy jeden plus jeden są szczególnie przydatne w maszynach z pamięcią bębnową, ze względu na możliwość usprawnienia wybierania (stosowane były w maszynach IBM 650, ODRA 1003). Konstruowane też były maszyny trójadresowe (NORC, MIDAC, STRIEŁA), czteroadresowe (EDVAC, SEAC), pięcioadresowe (maszyna SAPO-CSR - wybór czwartego lub piątego adresu następował alternatywnie w zależności od tego, czy wynik operacji był dodatni czy ujemny).

Około roku 1960 publikowane były w różnych krajach (w tym również w Polsce) koncepcje tzw. maszyn bezadresowych (P - 1). Generalnym założeniem tych maszyn miało być znaczne uproszczenie zarówno konstrukcji jak i programowania. Aczkolwiek poczyniono pierwsze eksperymenty (m.in. w Polsce i Australii), nie doszło do przemysłowej produkcji tych maszyn. Maszyny bezadresowe przeznaczone były w zasadzie do wykonywania prostych i powtarzalnych obliczeń technicznych. Być może idee te zostaną znowu podjęte, jeśli na szerszą skalę zacznie się stosować niezależne moduły asocjacyjnej pamięci operacyjnej.

2. Wiadomo, że elektroniczna maszyna cyfrowa to zespół zarówno środków hardware'owych (konstrukcyjnych - w dosłownym tłumaczeniu "żelaznych") i software'owych (programowych - "miękkich"). Dokładna definicja komputera nie została właściwie przez nikogo

podana, mimo, iż wszyscy mniej więcej zdajemy sobie sprawę, o co chodzi, gdy mówimy o komputerze. Po to, by unaocznić software'owe i konstrukcyjne strony nowoczesnego komputera podajemy jego następującą definicję: "Komputer jest to środek o dużej złożoności konstrukcyjnej (zespół różnych urządzeń) automatycznie wykonujący ciągi operacji arytmetycznych, logicznych, przesyłowych, sterujący pracę własnych urządzeń, sygnalizujący własne błędy, kontaktujący się z operatorem, posiadający własny język porozumiewania się (programowania) i język liczenia dwójkowy, dwójkowo-dziesiętny)".

Jak z powyższych definicji wynika, elementy "zmienne" (software'owe) odgrywają w maszynie decydującą rolę (przynajmniej z punktu widzenia użytkownika). Nie zawsze jednak tak było.

Pierwsza elektroniczna maszyna cyfrowa ENIAC właściwie komputerem jeszcze nie była. Składała się ona prawie wyłącznie z hardware'u (nawet program nie był przechowywany, m.in. z powodu małej pamięci, składającej się z 20 rejestrów po 10 cyfr). Już przed tą maszyną istniały próby budowy automatycznych przekaźnikowych maszyn liczących, np. MARK I - ASCC (Automatic Sequnce Controlled Calculator) budowany w Harvard University w latach 1939 - 1944 pod kierownictwem prof. H.Aikena oraz Complex Computer skonstruowany w Bell Telephone przez dra G.R.Stibitza.

Zainteresowanie software'm znacznie wzrosło dopiero przy maszynach III generacji, wyposażonych w time-sharing, wielodostępnych i wieloprogramowych. Jeszcze w 1957 roku udział kosztów software'u w ogólnym koszcie maszyny wynosił (K - 4) zaledwie 25%, zaś na przestrzeni lat 1967 - 1972 ma wzrosnąć do 70%.

Z technicznego punktu widzenia przyjęto wyróżniać następujące generacje komputerów:

zerowa - przekaźnikowa (lata czterdzieste),

pierwsza - lampowa (ENIAC - 1945, ODRA 1001 - 1960, ZAM-2 - 1961)

druga - tranzystorowa (mniej więcej od 1954 r - TRADIC).

Rozróżnienie dalszych generacji na podstawie powyższego kryterium jest w zasadzie niemożliwe ze względu na wzrost znaczenia organizacji działania komputerów.

Z programowego punktu widzenia pierwsza generacja charakteryzuje się stosowaniem języków wewnętrznych i symbolicznych prostych (1:1), zaś w drugiej dochodzą języki wyższego rzędu, języki symboliczne z makrorozkazami oraz proste systemy operacyjne.

Nie ukształtowały się jeszcze definicje maszyn III i IV generacji. przytoczymy więc tylko niektóre typowe ich cechy:

III generacja

1. praca w podziale czasu (time sharing)
   czyli zdolność współpracy - w kolejnych krótkich jednostkach czasu - jednostki centralnej z wieloma urządzeniami wejścia - wyjścia,
2. wieloprogramowość
   czyli zdolność realizacji kilku programów znajdujących się równocześnie w maszynie,
3. wielodostępność
   czyli zdolność współpracy maszyny z wieloma operatorami, przy czym każdy z nich ma wrażenie, że jest wyłącznym jej użytkownikiem,
4. minimalna pojemność pamięci operacyjnej 32K słów (o ile nie

ma masowej pamięci pomocniczej o dostępie wyrywkowym); z reguły stosowana jest byte'owa lub znakowa organizacja pamięci (zamiast wyrazowej),

5. praca asynchroniczna
   m.in. dzięki budowie modularnej, w której moduły wejścia-wyjścia kontrolują urządzenia transferowe bez angażowania jednostki centralnej,
6. złożony system operacyjny do sterowania pracą asynchroniczną, komunikacji z operatorem itp.

IV generacja obejmuje cechy III generacji plus:

1. firmware
   stosowany zarówno w emulacji, jak i w programach użytkowych,
2. wielka integracja elektroniczna (L S I - large scale integeation),
3. rozbudowa pamięci operacyjnej do 1 ml słów (lub 4 ml byte'ów),
4. wzrost stopnia wielodostępności i środków komunikacji człowiek-maszyna (w szczególności opracowanie praktycznych języków konwersacyjnych),
5. wieloprocesorowość
   oznaczająca zastosowanie kilku jednostek centralnych, (np. cztery jednostki w maszynach IRIS 80 lub K 202),
6. wzrost stopnia modularności
   oznaczający m.in. stosowanie kilku modułów pamięci operacyjnej o niezależnym dostępie, modułów niezależnych (nie powiązanych hierarchicznie) jednostek centralnych, niezależnych modułów wejść-wyjść, itp.

Cechą charakterystyczną IV generacji jest znaczny wzrost szybkości maszyny głównie poprzez ulepszenie organizacji działania

(a nie zmianę parametrów technicznych poszczególnych układów elektronicznych).

Maszyny IV generacji powinny być wzajemnie bardziej zamienne ze względu na elastyczność organizacji działania: modułowość, wymienne pakiety firmware'u, rozwinięte stronicowanie, software do kierowania przepływem danych itp.
Trudno jest w tej chwili mówić o ostatecznym kształcie maszyn IV generacji , tym bardziej, że właściwie nie rozpoczęto jeszcze ich produkcji na skalę przemysłową. Jeszcze pare lat temu generacja ta stanowiła "szyld" tego wszystkiego, co może być ulepszone w komputerach.

W 1967 roku redakcja znanego fachowego czasopisma amerykańskiego DATAMATION (X - 17) przeprowadziła ankietę na temat maszyn IV generacji. 50% pytanych oświadczyło, że produkcja komputerów tej generacji rozpocznie się w latach 1971-1975, 25% - pod koniec 1970 roku, zaś pozostałe 25% oświadczyło, że komputery te w ogóle ... nie zostaną wyprodukowane. Prognozowanie rozwoju komputerów jest bardzo zawodne. Oto przykłady:

1. skonstruowanie perceptronu w 1958 roku stało się podstawą przepowiedni o zastąpieniu programowania przez systemy samoorganizujące,
2. kriotrony (1959) miały stać się jedną z podstawowych pamięci,
3. od paru lat mówi się o możliwości wykorzystania światła laserowego w urządzeniach pamięciowych,
4. kiedy w 1953 roku dokonano tłumaczenia z rosyjskiego na angielski, wydawało się, że otwiera się droga do automatyzacji tłumaczeń na szeroką skalę.

3. Jedną z ciekawszych tendencji rozwoju konstrukcji maszyn jest praca wieloprocesorowa (w ramach jednej maszyny) i praca wielomaszynowa.

Przez system wielomaszynowy rozumiemy zespół kilku (nastu, dziesięciu) maszyn pracujących równocześnie i komunikujących się wzajemnie np. za pośrednictwem centralnego systemu operacyjnego, wymieniających dane pomiędzy sobą itp. Granice pomiędzy systemem wieloprocesorowym i wielomaszynowym są czasem trudne do uchwycenia, jeśli rozpatrujemy je jedynie na tle funkcjonalnym i możliwości przetwarzania. Jakimś, w miarę pomocnym kryterium może być rozrzucenie terytorialne poszczególnych jednostek w systemach wielomaszynowych.

W pracy niniejszej używamy pojęcia "system wielomaszynowy" zamiennie do pojęcia "wielosystem"[1/]. W publikacji (G - 5) za system wielomaszynowy uważa się taki rodzaj wielosystemu, w którym wszystkie "maszyny" są równoprawne i posiadają identyczną wydajność (w odróżnieniu od systemów hierarchicznych, w których rozróżnia się maszyny nadrzędne i podrzędne). Na marginesie tej definicji można rzucić uwagę, że maszyny wieloprocesorowe IV generacji najczęściej posiadają właśnie taką organizację działania, a mimo to nie uważa się ich za systemy wielomaszynowe.

Jako przykład systemu wielomaszynowego wymienić można system 29 maszyn pracujących dla IPC (International Paper Company), w którym maszyny są rozrzucone w 25 miejscach i połączone siecią transmisji danych.

1/ Termin "wielosystem", podany przez autora budzi zastrzeżenia. Zdaniem Redakcji, wystarczy operowanie terminem "systemu wielomaszynowego". (Przyp.Red.).

Ilustracją maszyny wieloprocesorowej jest ILLIAC IV. Maszyna ta posiada 256 urządzeń arytmetycznych, podłączonych do 4 jednostek sterowania. Dzięki zwielokrotnieniu urządzeń arytmetycznych maszyna ta osiąga szybkość 1 mld oper/sek (X-18).

Część centralna innej maszyny - CDC 6600 - zawiera 10 wyspecjalizowanych jednostek przetwarzania (m.in. do operacji zmiennoprzecinkowych (D - 4).

Systemy wielomaszynowe stosowane są wtedy, gdy:

a/ wymagana jest praca bezawaryjna, np. przy sterowaniu procesami technologicznymi,

b/ wyniki obliczeń muszą być bezbłędne (w tym celu na kilku maszynach-procesorach wykonywane są takie same zadania),

c/ występują różnorodne zadania, wykonanie których może być podzielone pomiędzy maszynami,

d/ ilość informacji jest tak duża, że gwarantuje pewien stopień obciążenia dla każdej z maszyn.

Niekiedy brany jest pod uwagę tylko jeden z powyższych czynników.

Historia systemów wielomaszynowych jest dosyć długa. Już jedna z pierwszych automatycznych maszyn przekaźnikowych BELL V (1944) firmy Bell Telephone Laboratories składała się z dwóch identycznych maszyn, równolegle podłączonych do wejścia, pracujących niezależnie i porównujących wyniki takich samych obliczeń.

W zbudowanej w 1958 roku w Czechosłowacji maszynie SAPO zastosowano trzy podłączone równolegle arytmometry, korygujące wzajemnie wyniki (za wynik prawidłowy uważano rezultat uzyskany co najmniej na dwóch arytmometrach).

Projekt SOLOMON, realizowany przez firmę Westinghouse (USA)

przewiduje zastosowanie około tysiąca niezależnych jednostek liczących równolegle zadany program.

System STRETCH (IBM, 1961) obejmuje dwie maszyny cyfrowe, z których jedna o działaniu szeregowym (tzn. wolniejsza) przeznaczona jest do wykonywania prac przygotowawczych, zaś druga o działaniu równoległym wykonuje podstawowe obliczenia.

W latach 50-tych w USA do sterowania środkami obrony przeciwlotniczej stosowano zestawy dwóch maszyn ANFSQ-7 (1956) pracujących równolegle w celu zabezpieczenia wysokiej pewności działania.

Również w USA stosowany jest system Meiseng, w skład którego wchodzą 3 maszyny (IBM 1620, IBM 1410, S-C 4020), przeznaczony do prac inżynieryjnych i przedstawiania wyników w postaci graficznej.

Znana jest koncepcja zastosowania kilkunastu emc w kombinacie metalurgicznym Spencer Works do obsługi wielopoziomowego kompleksu produkcyjnego. Maszyny działają na tzw. poziomach ustawionych hierarchicznie: maszyny poziomu I sprawują funkcje nadrzędne w stosunku do maszyn poziomu II itp. Różnicowanie poziomów nastąpiło nie wg etapów przetwarzania danych, lecz poziomów zarządzania. Poziom I przeznaczony był do wykonywania prac związanych z planowaniem wieloletnim i rocznym, poziom II i III zajmował się planowaniem krótkookresowym i harmonogramami produkcji, zaś poziom IV - sterowaniem procesów technologicznych.

4. Jedną z głównych cech nowoczesnego komputera jest praca wielodostępna w podziale czasu. Z punktu widzenia programowania systemy te możemy podzielić na uniwersalne i wyspecjalizowane.

Systemy uniwersalne są zdolne do realizacji dowolnego programu. Praca maszyny jest programowana bezpośrednio przez użytkownika w terminalu[1/] (względnie program jest ściągany z biblioteki), w związku z czym operator w zasadzie nie wie, co maszyna w danej chwili wykonuje.

Systemy wyspecjalizowane przeznaczone są do wykonywania jednego zespołu programów dla wielu użytkowników. Przykładem takiego systemu może być program rezerwacji miejsc w komunikacji lotniczej. Maszyny pracujące w tych systemach mogą być konstrukcyjnie (mikroprogramowo) przystosowane do określonego rodzaju pracy. Wprowadzenie uniwersalnych systemów było możliwe dzięki osiągnięciom w konstruowaniu i programowaniu maszyn cyfrowych, które z kolei zostały przygotowane przez prace teoretyczne.

W 1959 roku na konferencji UNESCO wygłoszony został przez Ch.Strachey'a pierwszy referat na temat idei time-sharingu i pracy wielodostępnej. W dwa lata później J.Mc Carthy sformułował pięć głównych wymogów podziału czasu:

1/ duża ferrytowa pamięć operacyjna,

2/ system przerywania,

3/ ciągła (non-stop) praca maszyny przy realizacji różnych programów, a więc automatyczne przechodzenie od zadania do zadania,

4/ ochrona pamięci dla poszczególnych programów,

5/ masowa pamięć pomocnicza, przeznaczona do przechowywania zbiorów współpracujących z poszczególnymi programami.

Zanim doszło do pierwszych teoretycznych rozpraw, musiała zostać postawiona sama problematyka. Konieczność zastosowania time-sharingu wyszła w trakcie budowy systemu obronnego SAGE

1/ Coraz częściej używa się terminu "urządzenie końcowe" (Przyp.Red.).

(Semi-Automatic Ground Environment) prowadzonej przez Lincoln Laboratory MIT i Rand Corp.

W listopadzie 1961 roku zademonstrowano w ośrodku obliczeniowym MIT jeden z pierwszych uniwersalnych systemów, który nie spełniał jednak wszystkich pięciu wymogów Mc Carthy'ego. W 1963 roku znane były dwa zaawansowane wielodostępne systemy uniwersalne. Jeden z nich narodził się w MIT, gdzie opracowano projekt MAC (Multiple-Access Computer) i uruchomiono system CTSS (Compatible Time-Sharing System) na maszynie IBM 7090. W tym samym roku opracowano system w System Development Corp.

W roku 1964 działało w USA około dziesięciu systemów wielodostępnych. W roku 1965 pojawiają się pierwsze seryjne maszyny spełniające wszystkie wymogi - GE 265 i UNIVAC 491.

Specjalizowane systemy wielodostępne niekoniecznie muszą pracować w podziale czasu. W 1952 roku w American Airlines uruchomiono system rezerwacji miejsc, w którym użyto specjalizowanej maszyny z pamięcią bębnową z wbudowanymi programami. Wśród systemów specjalizowanych rozróżnia się tzw. rozwinięte systemy, które, mimo ograniczeń związanych z danym rodzajem pracy, mogą być modyfikowane w zależności od potrzeb każdego użytkownika (np. systemy informacyjno-diagnostyczne w szpitalnictwie, systemy wybierania informacji itp.).

Rozwój zastosowań systemów wielodostępnych zależny był m.in. od wprowadzenia odpowiednich języków "konwersacyjnych", innych od używanych w partiowo-okresowym przetwarzaniu.

Pionierskim osiągnięciem był tutaj JOSS, opracowany w RAND CORP. Mniej więcej w tym samym czasie (1964) powstał język BASIC autorstwa J.Kemeny i T.E. Kurtz z Dartmouth College.

Do tej pory języków konwersacyjnych opracowano wiele. Pewne informacje na ten temat można znaleźć w rozdziale IV.

VIII. ROZWÓJ PROGRAMOWANIA A ZASTOSOWANIA ELEKTRONICZNYCH MASZYN CYFROWYCH

1. Uwagi ogólne o przeszłości i perspektywach zastosowań.

Jeden z ojców przemysłu komputerowego John Mauchly twierdził, że tylko 4-5 wielkich koncernów USA potrafi zagospodarować komputery. Tymczasem już w 1951 roku pracowało około 100 komputerów, zaś obecnie jest ich tysiąc razy więcej. Jest to skok ogromny, zważywszy, że komputer to nie liczydło biurowe za kilkadziesiąt złotych, lecz drogi zestaw urządzeń przeciętnie za kilkaset tysięcy dolarów. Mamy więc do czynienia z urządzeniem, które co prawda bezpośrednio w zasadzie nie tworzy dóbr produkcyjnych (materialnych) ani (przynajmniej dotychczas) nie stanowi przedmiotu prywatnego użytku szerokich rzesz obywateli, ale stało się przedmiotem działania najnowocześniejszych gałęzi przemysłu i miernikiem poziomu technicznego krajów.

Upraszczając zagadnienie, można powiedzieć, że już po pierwszej wojnie światowej istniały pewne przesłanki do zbudowania i zastosowania komputera. Znany był zapis magnetyczny (stanowiący podstawę urządzeń pamięciowych), wynaleziono lampy elektronowe oraz tak ważny układ, jakim jest przerzutnik. Babbage podał funkcje automatycznej maszyny liczącej, zaś Boole sformułował aparat formalny do opisu jej działania. Istniało duże zapotrzebowanie na obliczenia w astronomii i ilościowej analizie chemicznej.

Do tej pory języków konwersacyjnych opracowano wiele. Pewne informacje na ten temat można znaleźć w rozdziale IV.

VIII. ROZWÓJ PROGRAMOWANIA A ZASTOSOWANIA ELEKTRONICZNYCH MASZYN CYFROWYCH

1. Uwagi ogólne o przeszłości i perspektywach zastosowań.

Jeden z ojców przemysłu komputerowego John Mauchly twierdził, że tylko 4-5 wielkich koncernów USA potrafi zagospodarować komputery. Tymczasem już w 1951 roku pracowało około 100 komputerów, zaś obecnie jest ich tysiąc razy więcej. Jest to skok ogromny, zważywszy, że komputer to nie liczydło biurowe za kilkadziesiąt złotych, lecz drogi zestaw urządzeń przeciętnie za kilkaset tysięcy dolarów. Mamy więc do czynienia z urządzeniem, które co prawda bezpośrednio w zasadzie nie tworzy dóbr produkcyjnych (materialnych) ani (przynajmniej dotychczas) nie stanowi przedmiotu prywatnego użytku szerokich rzesz obywateli, ale stało się przedmiotem działania najnowocześniejszych gałęzi przemysłu i miernikiem poziomu technicznego krajów.

Upraszczając zagadnienie, można powiedzieć, że już po pierwszej wojnie światowej istniały pewne przesłanki do zbudowania i zastosowania komputera. Znany był zapis magnetyczny (stanowiący podstawę urządzeń pamięciowych), wynaleziono lampy elektronowe oraz tak ważny układ, jakim jest przerzutnik. Babbage podał funkcje automatycznej maszyny liczącej, zaś Boole sformułował aparat formalny do opisu jej działania. Istniało duże zapotrzebowanie na obliczenia w astronomii i ilościowej analizie chemicznej.

Mimo to, dopiero pod wpływem potrzeb II wojny światowej rozpoczęto prace nad budową uniwersalnej szybkiej maszyny liczącej. Co prawda, ENIAC nie zdążył już spełnić swojego przeznaczenia (obliczanie torów balistycznych), ale okazało się, że z powodzeniem może rozwiązywać inne problemy matematyczne.

Pierwsze komputery służyły głównie do obliczeń naukowych i technicznych. Problematyka administracyjna, w której występuje masa informacji i różnorodne reguły postępowania, nie mogła być na tych maszynach przetwarzana z powodu braku dużych pamięci i szybkiego aparatu piszącego.

Od roku 1949 równolegle w USA i Wielkiej Brytanii prowadzone były prace nad zbudowaniem odpowiednich maszyn. W dwa lata później powstały komputery LEO (W.B.) i UNIVAC (USA), które zapoczątkowały epokę naprawdę uniwersalnych maszyn matematycznych. Komputery te (podobnie jak i późniejsze) mogły rozwiązywać zarówno skomplikowane zadania matematyczne, jak i złożone problemy administracyjne (w rodzaju gospodarki materiałowej, płac itp.)

Maszyna LEO zbudowana została przez dużą firmę gastronomiczną (!) dla własnych potrzeb, a mianowicie do obsługi kilkuset restauracji, piekarń i kawiarń w zakresie ewidencji sprzedaży, badania popytu, obliczania płac itp. Maszynę oparto o rozwiązania konstrukcyjne maszyny EDSAC a koszt jej wynosił około 75 tysięcy funtów. Nie była to więc kwota wysoka.

Maszyna UNIVAC po raz pierwszy wykorzystywana była przez Biuro Spisu Ludności USA. Obliczono, że w pracach tego biura jedna minuta pracy komputera równoważna była pod względem wydajności 67 godzinom ręcznej pracy. UNIVAC I był pierwszym komputerem wprowadzonym do produkcji przemysłowej i używanym do wielu różnorodnych prac (do sporządzania list płac, ewidencji

materiałów, rozliczania ubezpieczeń, ewidencji i analizy sprzedaży itp.

Lista zastosowań komputerów jest bardzo szeroka. Obejmuje obecnie 1700 dziedzin techniki, ekonomiki, nauki i kultury. 80-90% komputerów znajduje zastosowanie w przedsiębiorstwach, instytucjach bankowych i handlowych, centralnych urzędach rządowych itp.

Obserwujemy w ostatnich latach tendencję odstępowania od jednotematycznych rozwiązań fragmentarycznych na rzecz wielotematycznych systemów zintegrowanych, w których tworzony jest wspólny bank danych. Systemy te mają na celu bieżące informowanie kierownictwa, zapewniając równocześnie najbardziej efektywne przetwarzanie (poprzez specjalne procedury operowania na wspólnej bazie danych).

Stosunkowo rzadko używane są komputery do sterowania procesami technologicznymi. Mimo, iż są to zastosowania z reguły bardzo opłacalne (w masowym typie produkcji o wysoce zautomatyzowanych procesach), wymagają długofalowych prac "identyfikacyjnych" i są bardzo odpowiedzialne. Musi być dokładnie sformułowany proces technologiczny (łącznie ze wszystkimi możliwymi odchyleniami), należy zastosować dokładną aparaturę kontrolno-pomiarową i wysokiej jakości komputery (długi okres pracy międzyawaryjnej).

Pierwsze zastosowanie komputera do sterowania oddziałem produkcyjnym w tzw. układzie zamkniętym miało miejsce w 1959 roku w rafinerii Port Arthur (USA).

W krajach zachodnich w warunkach ostrej walki konkurencyjnej komputery są często używane do obsługi klientów, szczególnie w sprzedaży hurtowej. Punkty sprzedaży połączone są siecią transmisji danych z ośrodkiem obliczeniowym. Komputer wydaje polecenie realizacji zamówienia na punkt zlokalizowany najbliżej siedziby klienta i posiadający wymagany asorty-

ment. Od kilkunastu lat stosuje się komputery do załatwiania rezerwacji miejsc w komunikacji lotniczej. Problem polega na tym, że pasażerowie dokonują wstępnej rezerwacji miejsc i potem często ją zmieniają w ostatniej chwili. Towarzystwa lotnicze mogłyby z tego powodu ponosić straty (niewykorzystane miejsca trudno jest klasycznymi środkami komunikacyjnymi rozprowadzić szybko pomiędzy wiele punktów sprzedaży). Kasy biletowe podają zgłoszenia (lub odwołania) rezerwacji poprzez sieć transmisji do komputera. Otrzymują odpowiedź już po kilku sekundach.

O uniwersalności komputerów świadczy fakt używania ich do planowania produkcji kwiatów w dużych kwiaciarniach. Kwiaciarnie takie dostarczają kwiaty do wielu krajów, ale nie chodzi tutaj o obsługę rozliczeń finansowych. Problem polega na tym, że kwiatów nie można magazynować, np. chryzantemy muszą być rozesłane do klientów samolotami najpóźniej po 3 tygodniach od momentu ich posadzenia. Komputery kontrolują więc wielkość zapasów i wykonują obliczenia w zakresie prognozowania ich zbytu.

Maszyny matematyczne używane są również do tłumaczenia tekstów z języka na język. Już w styczniu 1954 roku publicznie demonstrowano w Nowym Jorku tłumaczenie z rosyjskiego na angielski. W ZSRR opracowano m.in. programy tłumaczenia z francuskiego i angielskiego na rosyjski. W Japonii zbudowano komputer, który nie tylko tłumaczy z angielskiego na japoński, ale jednocześnie po japońsku czyta przetłumaczone zdania.

Komputery stosuje się dosyć szeroko w lecznictwie. 250 ekspertów z 22 krajów na naradzie zwołanej z inicjatywy międzynarodowej organizacji przetwarzania informacji IFIP orzekło optymistycznie, że już w 1980 roku większość lekarzy będzie miało dostęp do komputerów w celach konsultacji i diagnostyki.

Ostatnio komputery coraz częściej występują w roli "twórców" muzyki, poezji i malarstwa. Z uwagi na awangardowość sztuk nowoczesnych zdarzają się przypadki nieodróżniania utworu maszynowego od dzieła człowieka.

Uznanie zdobywają komputery w systemach nauczania. Maszyna przerasta nauczyciela szybkością reakcji, zdolnością zapamiętania i wybierania ogromnych ilości informacji, oraz - co najważniejsze - nie męczy się. To, czy komputer zastąpi uczniowi wartości wychowawcze i potrzeby psychiczne, wynikające z bezpośredniego kontaktu ucznia z nauczycielem, to już inna sprawa.

Szczególnie dobrze spisują się komputery w szkoleniu programistów. Ponieważ stale odczuwa się deficyt tego rodzaju kadr, w krajach zachodnich szkoli się w tym zawodzie nawet niewidomych i więźniów. Tak na przykład, więzienie waszyngtońskie posiada własną szkołę programowania. Wyselekcjonowanym więźniom stworzono swobodniejsze warunki (cele i sale wykładowe poza głównym budynkiem więziennym, odpowiednia atmosfera w czasie wykładów). Mieli oni zapewniony stały kontakt z komputerem IBM 360/40 za pośrednictwem terminali. W okresie rocznej nauki, oprócz kilku języków programowania, więźniowie opanowali również podstawy księgowości, ekonomiki i przetwarzania informacji. Absolwenci w okresie odbywania kary otrzymywali zlecenia na prace programowe od instytucji rządowych, zaś po zwolnieniu

z więzienia mogli podjąć studia specjalistyczne lub otrzymywali pracę programisty w rządowych ośrodkach obliczeniowych.

Eksperci przewidują, że ekspansja komputerów trwać będzie nadal. Być może powstaną światowe sieci komputerów, obsługujące międzynarodowe instytucje gospodarcze, handlowe, prawnicze, lekarskie i policyjne.

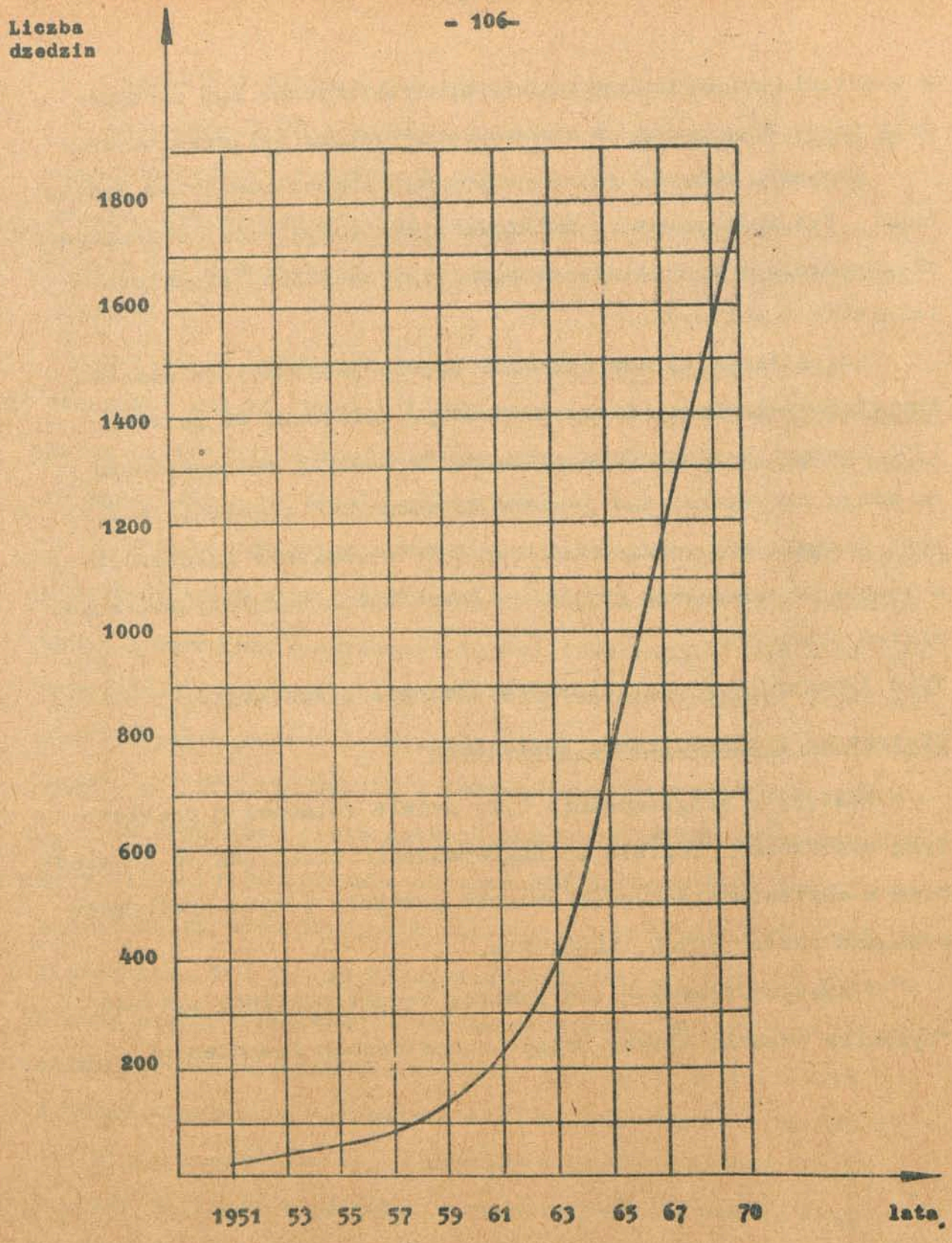
Coraz bardziej sam człowiek będzie poddawany działaniu komputerów. Eksperci firmy Rand Corp. twierdzą, że po roku 1990 nastąpi bezpośrednie podłączenie maszyny matematycznej do mózgu człowieka, zaś jeszcze do roku 2000 powstanie możliwość uczenia się poprzez bezpośrednie utrwalanie informacji w komórkach mózgowych (czyli do końca życia będziemy mogli pamiętać urazy !).

Tyle dywagacji na pograniczu futurologii i fantazji.

2. Statystyka i klasyfikacja zastosowań

Postępy w programowaniu były ściśle związane z problematyką zastosowań. Dopiero po nagromadzeniu kilkuletnich doświadczeń w zastosowaniach problemowych powstały języki problemowe wyższego rzędu: COBOL, ALGOL itp.

W naszych rozważaniach pomocny będzie poniższy wykres: "Dynamika wzrostu liczby dziedzin zastosowań komputerów".



RYS.1.

Dynamika wzrostu liczby zastosowań
/E - 5 i inne źródła /

Jak widać z wykresu, zdecydowany wzrost liczby dziedzin przypada na lata 60-te. Można to tłumaczyć tym, że pod koniec lat pięćdziesiątych opracowane zostały podstawowe języki programowania: FORTRAN, ALGOL, COBOL, dzięki czemu przedstawiciele różnych dziedzin mieli łatwiejszy dostęp do komputerów.
Z kolei użytkownicy, odczuwając specyfikę własnych zastosowań na tle powyższych, dalekich od doskonałości języków, proponowali własne języki wąskospecjalizowane. Tak więc różnorodność zastosowań znajdowała odbicie w różnorodności języków programowania (1963 - 300 języków, 1966 - 1200 języków). Wprowadzanie języków uniwersalnych (w rodzaju PL/I) powinno zahamować imponujący wzrost liczby języków programowania.

Wg oceny Diebolda (podanej w K - 4) z 1969 roku ewolucję systemów informacyjnych można przedstawić następująco:

Rok 1964 - druga generacja zastosowań

Zakres: administracja i rachunkowość,

Kryterium oceny (cel): redukcja etatów, zmniejszenie kosztu.

Rok 1968 - Trzecia generacja zastosowań

Zakres: informacja nadzoru (supervisory information)

Kryterium: zmniejszenie zapasów, stabilizacja obsady personalnej, usprawnienie obsługi klientów, kontrola kosztów.

Rok 1975 - trzecia i czwarta generacja zastosowań

Zakres: informacje dla kierownictwa średniego szczebla (middle management) oraz planowanie na szczeblu taktycznym,

Kryterium: prelimirowanie marketingu, skrócenie

okresu zwrotu nakładów, optymalizacja obciążenia urządzeń produkcyjnych, bardziej realistyczne prognozy.

Rok 1985 - piąta generacja zastosowań

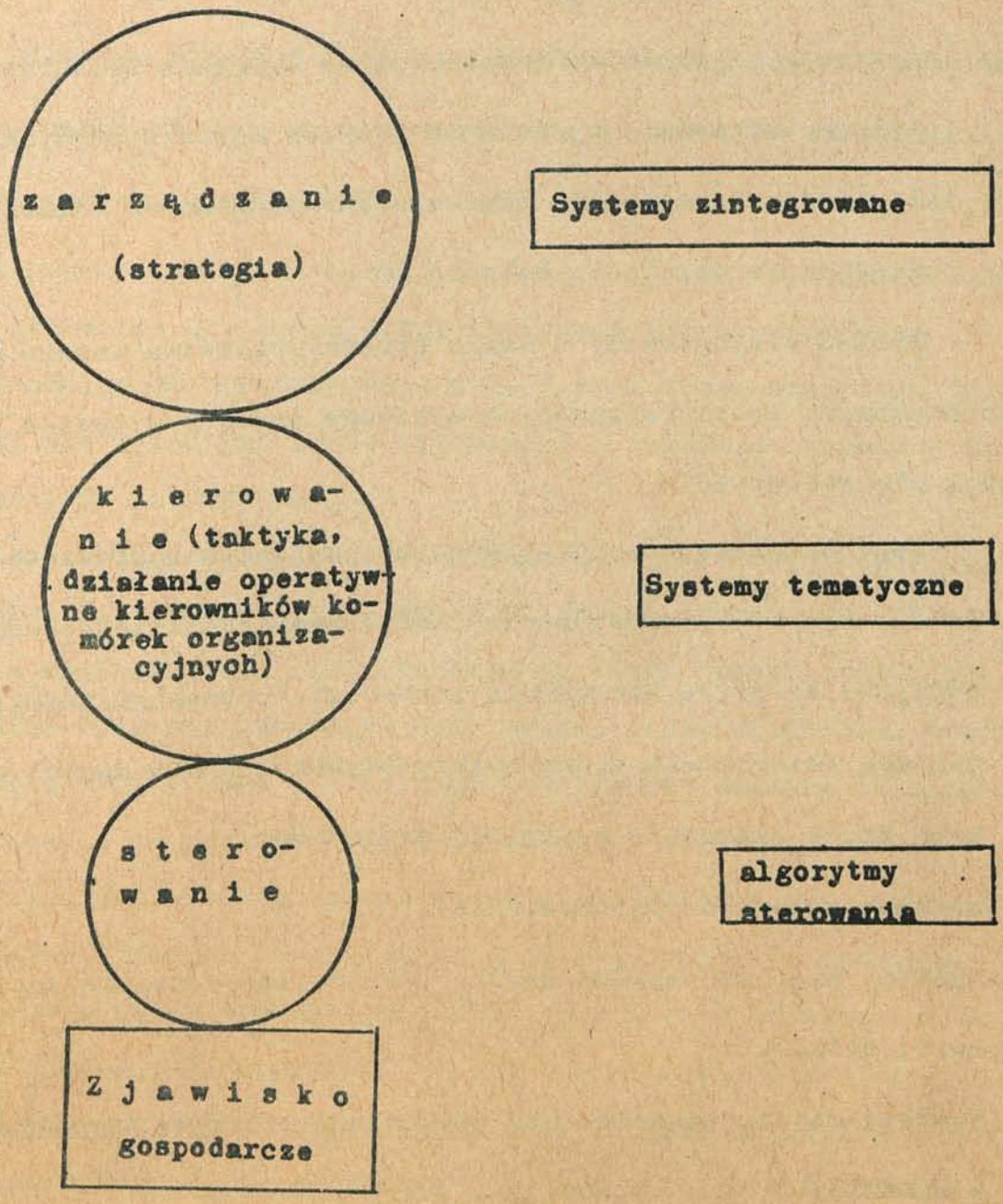
Zakres: obsługa naczelnego kierownictwa i planowanie na szczeblu strategicznym,

Kryterium: planowanie produkcji, zapotrzebowanie na kapitał, planowanie zapasów, siły roboczej itp.

Typowym poziomem zastosowań komputerów w latach 50-tych i 60-tych były systemy tematyczne, projektowane pod kątem potrzeb działania operatywnego kierownictwa poszczególnych komórek organizacyjnych przedsiębiorstwa (patrz rys. 2). Systemy te można było stosunkowo łatwo opracować i wdrożyć, ponieważ opierano je o doraźne potrzeby ewidencji i sprawozdawczości. Wobec trudności modelowania zarządzania, braku technologiczno-technicznej bazy systemów zintegrowanych (w szczególności software'u do operowania na wspólnej bazie danych) oraz trudności organizacyjnych, problematyka systemów zintegrowanych stanowiła raczej przedmiot teoretycznych rozważań niż rozwiązań praktycznych. Podstawową cechą systemów zintegrowanych jest integracja informacji (w postaci tzw. wsólnej bazy danych). Pełny system zintegrowany charakteryzuje się tym, że integracji informacji towarzyszą:

a/ integracja organizacji (złamanie sztywnych struktur wydziałowych, utrudniających wykorzystanie wspólnej bazy danych i integrację funkcjonalną),

b/ integracja środków technicznych (zastosowanie oprócz ETO różnorodnych środków małej i średniej mechanizacji,



Rys. 2. Szczeble działalności i odpowiadające im poziomy przetwarzania informacji

urządzeń transmisji danych, dostatecznie rozbudowana sieć telefoniczna i dalekopisowa),

c/ integracja użytkownika informacji ze środkami technicznymi (obsługa terminali, opanowanie języków konwersacyjnych),

d/ integracja funkcjonalna (funkcje poszczególnych komórek podporządkowane wspólnemu celowi).

System zintegrowany w skali przedsiębiorstwa nazwać można podstawowym. Do zintegrowanych systemów poziomu wyższego proponuje się zaliczyć:

- obsługę kompleksu produkcyjnego obejmującego kooperantów,
- integrację sfery zarządzania (zastosowań ekonomiczno-administracyjnych) ze sferą sterowania procesami technologicznymi, zapewniającą bezdokumentacyjne wykorzystywanie w EPD danych odbieranych przez aparaturę kontrolno-pomiarową,
- obsługę koncernu (zjednoczenia),
- systemy wieloszczeblowe handlu (obejmujące centrale zbytu, hurt, detal),
- systemy międzybranżowe (np. obejmujące przemysł konsumpcyjny i handel),
- systemy, w których badania operacyjne są zintegrowane z klasycznym przetwarzaniem danych (w zakresie danych, procedur i funkcji).

IX. JĘZYKI PROGRAMOWANIA OPRACOWANE W POLSCE

Do niedawna właściwie ani nie produkowaliśmy ani nie eksploatowaliśmy maszyn bogato oprogramowanych. W krajowych publikacjach (poza pewnymi skryptami i ogólnikowymi artykułami) nie popularyzowano naszych osiągnięć w dziedzinie programowania. Tymczasem "coś niecoś" uzbierało się przez okres minionego dziesięciolecia.
Nie pretendując do kompletności, w oparciu o fragmentaryczne dane, postaramy się przekazać pewne wiadomości o rodzimym dorobku w zakresie języków programowania.

1. Maszyną krajowej produkcji, posiadającą względnie oryginalne oprogramowanie, jest ZAM 41, wyposażony we własne języki: SAS (System Adresów Symbolicznych), PJES (Podstawowy JEzyk Symboliczny), EOL (Expression Oriented Language) oraz wersje językowe ALGOLu, COBOLu i podzbioru GPSS (CEMMA 2 - CyfrowE Modelowanie Maszyny Analogowej). Oprogramowanie to opracował w II połowie lat sześćdziesiątych (z wyjątkiem SAS, którego podstawy powstały wcześniej) Instytut Maszyn Matematycznych w Warszawie. Pierwszym językiem wyższego rzędu opracowanym w tym Instytucie był język SAKO (System Automatycznego KOdowania), którego pierwszy translator uruchomiono w 1962 roku. SAKO był w znacznej mierze językiem oryginalnym oraz posiadał polską terminologię (np. zwroty: GDY BYŁ NADMIAR: α , INACZEJ β; SKOCZ WEDŁUG I:α ,β ,ω.., ; WRÓĆ . itp)
Stosowano w nim zapis algebraiczny, np. wyrażenie $X_1 = \frac{-b - \sqrt{D}}{2A}$ było zapisywane jako X_1 = (-B - PWK (DELTA)) 2 x A.
Język EOL (Expression Oriented Language) jest językiem przeznaczo-

nym do przetwarzania symboli, w szczególności nadającym się do pisania translatorów (np. użyto go do pisania translatora COBOLu dla ZAM 41). Pierwsze wersje języka EOL powstały pod kierownictwem prof.dr Leona Łukaszewicza w Instytucie Maszyn Matematycznych w latach 1965 - 1966. Wersję EOL-3 opracowano w roku 1967 na uniwersytecie Illinois (USA) i zrealizowano w roku 1968 na maszynach IBM 7094 i IBM 360 (Ł - 1, Ł-2).

W języku EOL wykorzystano niektóre idee występujące w innych językach do przetwarzania symboli (COMIT, IPL-V). Program może być pisany w wersji polskiej i angielskiej, z tym, że w ramach jednej sekcji należy stosować wyłącznie słowa kluczowe jednego języka.

Oto przykłady niektórych instrukcji języka EOL:

WSTAW (PUT)..... oznacza pobranie wyrażenia E<n>z pamięci roboczej i umieszczenie go w P<m>w pamięci plikowej.

POBIERZ (GET) .... oznacza pobranie z P<n>i zapisanie do E<m>.

ZAMIEŃ (EXCHANGE) oznacza zamianę pomiędzy sobą wartości wyrażeń lub plików, wskazanych przez dwa argumenty rozkazu.

Wersja COBOLu dla ZAM 41 obok szeregu udogodnień zawiera cechy oryginalne, które stanowią o jej niezamienności w stosunku do wersji standardowych. Na dobro wersji zaliczyć trzeba uwzględnienie niepozycyjnego zapisu na taśmie dziurkowanej (z użyciem ograniczników pozycji), wygodne komponowanie tekstów za pomocą znaków specjalnych .SP .LT .YK .OHP, różnorodne sposoby rozpoznawania typu rekordu (wg ilości znaków, ilości pozycji - z wykorzystaniem TALLY), zastosowanie tak nowoczesnych instrukcji jak EXAMINE, itp. Elementami decydującymi o niezamienności są

deklaracje FIXED, FLOAT inspirowane przez konstrukcję maszyny (wyrazowa organizacja pamięci) i zapewne przez PL/I. Deklaracje te eliminują użycie klauzuli PICTURE (można je używać jedynie, gdy nie występują w opisie danej zwroty FIXED lub FLOAT. FIXED (FIXED 2) oznacza liczby całkowite (długie) w postaci binarnej, zaś FLOAT - liczby zmiennoprzecinkowe też w postaci binarnej.

W połowie lat 60-tych powstała w Instytucie Maszyn Matematycznych koncepcja języka LOGOL (dla maszyny ZAM-2), przeznaczonego do analizy językowej, a w szczególności do tłumaczeń z dowolnego języka o znanej gramatyce na dowolny inny język.

2. Opracowano w naszym kraju szereg translatorów języka ALGOL (L - 1):

a/ ALGUM - dla UMC1 i UMC10, 1965, Centralny Ośrodek Badań i Rozwoju Techniki Kolejnictwa,

b/ ALGOL 60 dla ZAM 41 - 1968, dla ZAM 21 - 1966, IMM.

c/ MINAL - podzbiór ALGOLu 60 dla emc ODRA 1204, 1968, Centralny Ośrodek Badań i Rozwoju Techniki Kolejnictwa.

d/ ALGOL 1204 - dla emc ODRA 1204, 1970, Katedra Metod Numerycznych Uniwersytetu Wrocławskiego.

3. Dla maszyn ODRA 1003 i 1013 opracowano język MOST, oparty o języki ALGOL i MARK I. Język ten, autorstwa grupy matematyków Katedry Metod Numerycznych Uniwersytetu Wrocławskiego i Wrocławskich Zakładów Elektronicznych ELWRO, dostosowany został do specyfiki konstrukcyjnej ODRY 1003. Wyraża się to: w możności użycia klawiatury akumulatora do drukowania śladu programu, adresowaniu z przeplotem (co siódme komórki na bębnie mają przydzielone sąsiednie adresy) poprzez wciśnięcie przycisku ZR na pulpicie

sterowania. Diagnostyka kompilatora jest dosyć uboga, zaś sygnalizacja błędów następuje poprzez wydrukowanie na dalekopisie "znaku" oraz tekstu instrukcji, w której został wykryty błąd. Znane są dwie wersje języka MOST: MOST 1 (dla emc ODRA 1003) oraz MOST F (dla emc ODRA 1013 - uwzględniający zalety pamięci ferrytowej). Jako ciekawostkę podajemy, że dla maszyny ODRA 1013 opracowano translator języka ALGOL (Czechosłowacja, Pilzno) oraz translator języka FALA 68 (Zakład Metod Numerycznych Uniwersytetu MCS w Lublinie). Język FALA 68 stanowi rozwinięcie kompilatora ALGUM i charakteryzowany jest jako język o stopniu kompilacji pośrednim pomiędzy MOSTem i ALGOLem. Zwroty tego języka mogą być pisane w języku polskim lub angielskim (z tym zastrzeżeniem, że nie wolno ich mieszać w tym samym programie). Translator bierze pod uwagę 5 pierwszych znaków nazw danych. Deklaracja REAL oznacza liczby zmiennoprzecinkowe zarówno w postaci normalnej (ciąg cyfr przedzielonych kropką dziesiętną) oraz w postaci półlogarytmicznej (z wykazaniem cechy i mantysy).

X. OCENA STANU DOTYCHCZASOWEGO I TENDENCJE ROZWOJOWE PROGRAMOWANIA

1. Wydaje się, że lata obecne zamykają pierwszy etap rozwoju programowania. Cechą widoczną tego etapu jest nadprodukcja języków programowania przy równoczesnym braku teoretycznych podstaw budowy języków (ujmujących zarówno składnię, jak i semantykę oraz pragmatykę). Dotychczas stosunkowo wiele zbadano jedynie w zakresie składni. Efektem niedorozwoju teoretycznego jest niski stopień zamienności języków, czy też nawet ich poszczególnych wersji, oraz trudności związane z symulacją i emulacją.

Dorobkiem dotychczasowych prac są przede wszystkim języki powszechnego użycia: FORTRAN, ALGOL i COBOL. Przyszłość przyniesie nam zapewne poważne prace teoretyczne, które umożliwią skonstruowanie efektywnych języków, prawdopodobnie wyspecjalizowanych (na problem,nie zaś na maszynę) lecz opartych na wspólnych podstawach teoretycznych. Dotychczas obserwowaliśmy przeważnie przypadkowe (z wyjątkiem ALGOLu i COBOLu) wysiłki poszczególnych producentów czy użytkowników ukierunkowane na tworzenie nowych języków. Większość tych języków miała krótki żywot, a często jedną (czy też jedyną) ich zasługą było wprowadzenie nowej terminologii dla już uprzednio zdefiniowanych spraw. W sytuacji, kiedy mamy kilka tysięcy języków, a prawie każdy z nich operuje innymi pojęciami, bardzo jest trudno znaleźć fachowca, który potrafiłby zapamiętać wszystkie różnice i spróbował wprowadzić wspólny aparat pojęciowy, umożliwiając tym samym szeroką analizę porównawczą.
Przykłady nadmiaru pojęć mogą być następujące:

- do określenia segmentu programu: segment, program-unit, module, section-nr.
- do określenia deklaracji: declarative, klauzula, dyrektywa.

2. Wydaje się, że w perspektywie można mówić o dwóch tendencjach rozwojowych programowania. Po pierwsze, będą opracowane metajęzyki, czyli języki teoretyczne, a celem ich będzie stworzenie kryteriów oceny języków oraz reguł budowy syntaktyki i semantyki. Po drugie, na bazie języków teoretycznych powstaną języki wąskospecjalizowane, dostatecznie efektywne a równocześnie łatwe w użyciu. Języki te powinny być zbliżone do języków naturalnych (unikając równocześnie werbalności), by umożliwić łatwy kontakt z maszyną, oraz być na tyle "fachowo", by zapewnić łatwe wyrażenie specyfiki problemu. Dostateczna efektywność programu powinna być zapewniona przez środki software'owe budowane modularnie w celu możności stosowania dowolnej kompozycji modułów i dalszej rozbudowy.

Dotychczasowe koncepcje języków (X - 2o) oparte były o tzw. budowę pozycyjną danych, wskazywaną przez specjalne deklaracje i "poziomowanie" hierarchii danych. Przewiduje się, że w przyszłości budowa pozycyjna zostanie zarzucona na rzecz oceny semantycznej.

Za próbę stworzenia metajęzyka można zapewne uważać uniwersalny generator HELP (Highly Extendible Languages Processor) ogłoszony przez organizację Advanced Computer Techniques i nazwany "general-purpose natural language generator (X - 19). HELP posiada następujące możliwości:

a/ translacja każdego języka naturalnego na inny,

b/ translacja skrótowych wersji FORTRANu, COBOLu (np.COAX) i innych języków na formy żądane do kompilacji,

Teoretycznie programista może stosować więc dowolny (nawet własny) język programowania.

3. Niekiedy spotkać można w publikacjach (np. B - 11) wzmianki o tzw. palindromicznym (odwrotnym) programowaniu.
Polega ono na tym, że program wykonywany jest nie od pierwszej lecz od ostatniej instrukcji, dając ten sam rezultat (podobnie jak czytanie słów "radar", "rotor").
Oto inne przykłady pracy palindromicznej:
a/ translacja programu wewnętrznego na program w języku zewnętrznym (symbolicznym, wyższego rzędu),
b/ procedury wykrywania błędów idąc "od tyłu" (tj. od wyniku),
c/ pakiety programowe "FLOWCHART" przeznaczone do tworzenia schematów blokowych napisanych programów.
4. Jedną z cech charakterystycznych maszyn IV generacji ma być dobre oprogramowanie komunikacyjne. Należy przypuszczać, że w związku z tym znacznie wzrośnie rola języków konwersacyjnych oraz że klasyczne języki programowania zostaną odpowiednio zmodyfikowane (wzbogacone). Jest to warunek użytkowania wspólnych baz danych w systemach bieżącego informowania kierownictwa.

Tak np. ostatnio trwają prace nad włączeniem do COBOLu właściwości CCF (Cobol Communication Facility), która ma umożliwić stosowanie COBOLu w systemach pytanie-odpowiedź.
Między innymi będzie to możliwe dzięki wprowadzeniu do DATA DIVISION specjalnej sekcji COMMUNICATION SECTION, instrukcji RECEIVE, SEND, itp.

W systemach komunikacyjnych dużą rolę powinny odgrywać

tzw. przyspieszone kompilatory (incremental compilers) (R - 8), które tłumaczą każde zdanie programu zewnętrznego natychmiast po wprowadzeniu z terminalu on-line, dając od razu diagnostykę błędów. W trakcie wprowadzania programu można eliminować poprzednie zdania, wprowadzać dodatkowe zdania itp. Do tego typu translatorów zalicza się: QUIKTRAN, RUSH (wersja PL/I), FIV (wersja FORTRANu IV), CAL oraz w dużej mierze BASIC.

5. Pewną nadzieję (już od pewnego czasu zresztą) na znaczne uproszczenie programowania stwarzają języki tablicowe (tabular languages), przeznaczone do tworzenia programów na podstawie odpowiednio opisanych tablic decyzyjnych. Skonstruowano już parę takich języków (TABSOL, LOGTAB,FORTAB, DETAB 165, GECOM), żaden jednak nie uzyskał szerszego zastosowania (podobnie, jak i same tablice decyzyjne).

Wygodne jest stosowanie tzw. przedtranslatorów, pozwalających na budowę programów mieszanych, złożonych z tablic decyzyjnych i z rozkazów COBOLu (lub FORTRANu).

Tablice decyzyjne zostały po raz pierwszy użyte prawdopodobnie w latach 50-tych w firmie GE (X - 21). W latach 60-tych rozwinięto szersze prace nad tablicami decyzyjnymi (np. Decision Tables Symposium w Nowym Jorku w 1962 roku).

Tablice decyzyjne są szczególnie pomocne do przedstawiania złożonych logicznie sytuacji, które jest trudno wyrazić graficznie na schematach blokowych. Inną zaletą jest to, że umożliwiają skontrolowanie, czy uwzględniono wszystkie możliwe kombinacje warunków i wynikające z nich czynności.

6. Docelową perspektywą programowania są systemy samoprogramujące. Podobno ma być to cecha charakterystyczna maszyn V generacji.

Wydaje się nam, że do wyeliminowania programistów nie dojdzie nigdy.

- - - ... - - -

BIBLIOGRAFIA

A.

1. Agapeyeff A.D. An introduction to commercial compilers. Computer Analysis and Programmers Ltd London, 1964 pp. 199 - 253

2. Auerbach I.L. Technology and the future. Data Systems 1/71 s.12-13

3. Amdahl G.M., Amdahl L.D. Forth-generation hardware. Datamation 1/67 s. 25-26

B.

1. Bowdon. Faster than thought. 1953

2. Booth A.D., Booth K.H.V. Automatic digital calculators. London 1956

3. Bielecki J. Maszyna Turinga Maszyny Matematyczne 1/67 s. 28-29

4. Buxton J.N. /redakcja/ Simulation programming languages. North-Holland, 1968

5. Blau H. In welcher sprache soll programmiert werden? BTA 1/70, s. 8-9.

6. Brandon D.H. - Management Standards for Data Processing. 1963

7. Bernard S.M. The c[illegible] for COBOL Computers and Automation 2/67, s 40-42

8. Borowiec J. Język PL/I: nowe cechy i elementy wyższych języków programowania. Biuletyn IMM. Nowości Techniczne, 1/67 s. 17-33

9. Borowiec J. Wprowadzenie do języka PL/I. Problemy przetwarzania informacji. WNT, Warszawa 1970

10. Bromberg H. The COBOL conclusion.
Datamation 3/67 s. 45-50

11. Bernstein W.A. Palindromie Programming.
Datamation 12/69 s. 123-124

C.

1. Cutler I.D. Introduction to computer programming
Prentice Hall. 1964

2. Cracken Me D.D. Digital Computer Programming.
John Wiley and Sons Inc. 1957

3. Carr J.W. Lectures given at the university of Michigan.
1958. /tłum.ros. Moskwa 1963/

4. Cracken Mc J.D. A guide to FORTRAN programming.
John Wiley and Sons, 1964

D.

1. Dańda J. Ryżko J. Tendencje rozwojowe organizacji pamięci maszyn cyfrowych. Problemy przetwarzania informacji, t.I, WNT 1970

2. Desmonde W.H. Computers and their uses. Prentice Hall 1964. wyd.polskie: Maszyny matematyczne i ich zastosowania. PWN, 1969

3. Dellert G.T. Jr A use of macros in translation of symbolic assembly language of one computer to another Comm. Assoc. Comp. Machinery 1965, 8 No 12, 742-748

4. Dańda J. Dziś i jutro maszyn cyfrowych.
Maszyny Matematyczne 3/67

E.

1. Empacher A.B. Maszyny liczą same?
Wiedza Powszechna i Sztandar Młodych. 1960

2. Emery G. Electronic Data Processing.
Hitman and Sons, London 1968

3. Edeleman K.E. A short guide to wonderful world of COBOL.
Datamation 12/69 s. 161-164

4. Empacher J., Maroński J., Sadowski A., Tarasiuk J.
Autokod ALGOL dla maszyny URAL-2. PWN, Warszawa 1966

5. Ewreinow E.W., Kosariow J.G. Odnorodnyje uniwersalnyje wyczislitielnyje sistiemy wysokoj proizwoditielnosti. Nowosibirsk, 1966

F.

1. Fisher F.P., Swindle G.F. Computer Programming systems.
tłum.ros. Sistiemy programmirowanija.
Moskwa 1971

G.

1. Greniewski H. Elementy logiki formalnej. PWN, 1955

2. Golden J.T. FORTRAN IV - programming and computing.
Prentice Hall 1965

3. Gabrini P. Etude D'un Systeme de programmation en commande numerique. Automatisme, 4/70 s.176-180

4. Gaines R. Stockton R. On the translation of machine language programs. Communs Assoc. Comp. Mach.
1965, 8 No 12, 736÷741

5. Głowacki B. Współczesne systemy cyfrowe
ETO Nowości 2/69 s. 3-23

6. Glauthier Computer time sharing: its orygins and development. Computers and Automation 10/67 s.23-28

H.

1. Higman Bryan. A comparative study of programming languages Mc Donald London 1968

2. Hellerman H. Digital Computer system principles. Mc Graw - Hill Book Co 1967

3. Hellwig Z. /red./ O maszynach cyfrowych. Warszawa 1970, PWE

4. Hicks H.T. Jr Modular programming in COBOL. Datamation 5/68

5. Humby. ICT COBOL Rapidwrite. Academic Press. 1964

6. Halpern M.I. Machine independence: its technology and economies. Communs Assoc. Comp. Mach. 1965, 8, No 12, 782-785

7. Hicks H.T. Jr The Air Force COBOL compiler validation system. Datamation 8/60 s. 73-74, 76-77, 81

8. Hicks H.T.Jr A communication facility for COBOL Datamation 12/69, s.148, 153-158

9. Hicks H.T.Jr ANSI COBOL. Datamation 11/1/70

I.

1. Ivoll T.E. Electronic computers. tłum. ros. Maszgiz, Moskwa 1959

J.

1. Junkier J.P., Boward G.R. COBOL vs FORTRAN, a sequel. Datamation 4/65 s. 65-67

2. Jerzykiewicz K., Szczepkowicz J. ALGOL 1204. poz.1204-VIII-2. ELWRO 1970

K.

1. Kitow A.I. Programmirowanije informacionno-logiczeskich zadacz. Sowietskoje Radio, Moskwa 1967

2. Klepacz W. Pamięci masowe maszyn cyfrowych WNT Warszawa, 1970

3. Korolew M.A. Obrabotka ekonomiczeskoj informacji na elektronnych maszinach. Ekonomika 1964

4. Kriebel C.H. The evaluation of management information systems. IAG Journal, vol.4 No 1,1971. s. 1-14

5. Klepacz W. Zastosowanie maszyn matematycznych dla automatyzacji zarządzania. WNT Warszawa, 1965

L.

1. Liapunow A.A. O logiczeskich schemach programm. Woprosy kibernetyki 1/58 s.46-127

2. Lippit. COBOL and compatibility Comm.of ACM v.5 5/62 s. 254-255

3. Ledley. Digital computer and control engineering. Mc Graw Hill 1960

4. Lombardi L. Mathematical Structure of nonarithmetic data processing procedures. Journal ACM 1962. v.9.1 136-159

Ł.

1. Łukaszewicz L. Automatyzacja programowania w Polsce do roku 1970. Informatyka 3/71

2. Łukaszewicz L. EOL - język do przetwarzania symboli. Maszyny Matematyczne 5/69 s. 8-11

M.

1. Małuszyński J., Witaszek J. Wiedeńska metoda opisu języków programowania. Informatyka 3/71

2. Mazurkiewicz A. Problemy języków formalnych w automatycznym przetwarzaniu informacji. Problemy przetwarzania informacji. Tl. WNT W-wa 1970

3. Moyle M.P. Introduction to computers for engineers. John Wiley and Sons, 1967

P.

1. Pawlak Z. Organizacja maszyn bezadresowych. PWN W-wa, 1965

2. Pawlak Z. Sygnały, symbole, maszyny. Wiedza Powszechna, W-wa 1965

3. PL/I Language Specyfication. IBM 1966

4. Paszkowski S. Język ALGOL 60. PWN, 1965

O.

1. Oswald H. Translation by XACT. Datamation 1/67 s.37-38

2. Opler A. Fourth generation software. Datamation 1/67 s. 22-24

3. Opler A. The receding future. Datamation 9/67, s.31-32

R.

1. Raymond F.H. L'automatique des informations. Masson et C.Editeurs, Paris, 1957

2. Ruderman M.E. Pappalardo A.N. The hospital computer comes of age. Computers and Automation. 6/70 s.29-32

3. Rubey R.J. A comparative evaluation of PL/I. Datamation 12/68 s. 21-38

4. Revised Report on the Algorithmic Language ALGOL 60. IFIP, 1962 pod red. P.Naura

5. Radin G. Rogoway H.P. NPL - nev programming Language. Comm. of ACM v.8.1. 1965

6. Rottmann H.H. Programmieren leicht gemacht mit RPG IBM 360/20. Carl Hauser Verlag, München, 1968

7. Reynolds C.H. Software development and its costs. Computers and Automation 2/67 s. 18-21

8. Rishel W.J. Incremental compilers. Datamation 1/70 s. 129-136

9. Ryznar Z. Sterowanie procesami technologicznymi za pomocą elektronicznych maszyn cyfrowych. Organizacja-Samorząd-Zarządzanie 4/67 s. 204-208

10. Ryznar Z. Komputery w społeczeństwie XXI wieku. Problemy 8/69 s.483-486

11. Ryznar Z. Zarządzanie w czasie rzeczywistym. Organizacja-Samorząd-Zarządzanie 9/66 s. 516-519

12. Richards R.K. Digital computer components and circuits. D. van Nostrand Co, 1958

S.

1. Sammet E.J. Programming languages: History and fundamentals Prentice Hall In c., 1969

2. Sammet E.J. Fundamental concepts of programming languages. Computers and Automation 2/67 .s.30-31,34-35

3. Shirley D. Comparing COBOL's. Data and Control Systems 1/67

4. Szuprowicz B.O. The time-sharing users: who are they? Datamation 8/69

T.

1. Tatarkiewicz Władysław Historia filozofii. t.III. PWN 1958

2. Targowski A. Automatyzacja przetwarzania danych. PWE 1970

3. Trachtenbrot B.A. Algoritmy i maszinnoje reszenije zadacz. Moskwa 1960

4. Teague R.G., Brady A.H. COAX: a preprocessor for COBOL
Datamation 7/69

V.

1. Vocabulary of Information Processing.
IFIP-ICC. North Holland. 1968

W.

1. Wilkes M.V. Computers then and now.
Journal of the Association for Computing
Machinery. 1/68, s. 2-7

2. Warmus M. GIER-ALGOL. PWN, W-wa 1966

3. Wilkes M.V. Time-sharing computer systems.
Mc Donald, London, 1970

X.

1. Datamation 7/69 s. 69, 79
2. Computer Weekly 14.1.71
3. Datamation 4/69 s.199
4. Datamation 6/67 s.31
5. Data Processing 9-10/67 s.17-20
6. Computers and Automation 12/68 s.23
7. Nowości ETO 2/69
8. Data and Control Systems 1/67
9. Sowriemiennoje programmirowanije.
Zbiór art. Tłum. z ang. Moskwa 1966
10. Computers and Automation 5/70 s.77
11. Communic. of ACM vo. 8.no 1,5/65
12. Datamation 2/69 s.11
13. Introduction to IBM data processing systems. IBM 1969

14. COBOL ICL System 4-50 Manual

15. COBOL ICL series 1900 Manual

16. Maszyny Matematyczne 2/67

17. The next generation from 50 viewpoints. Datamation 1/67 s. 31-34

18. Burroughs chosen to build ILLIAC IV. Computers and Automation 4/67 s.51

19. Datamation 7/69 s.203

20. Maszyny Matematyczne 5/68 s. 21-22

21. Elektronische Datenverarbeitung 2/70

SKOROWIDZ NAZW, NAZWISK I POJĘĆ