*Część II*

# Technologia przetwarzania danych

# 3

# Problematyka oprogramowania

## *3.1. Struktura oprogramowania*

Oprogramowanie systemów APD składa się z:

a) programów sterujących (*control program*), określanych również jako system operacyjny [1],

b) programów przetwarzania (*processing programs*), które obejmują:

— programy tłumaczące (*language translators*) języki autokodów na język wewnętrzny maszyny,

— programy uniwersalne — usługowe (*service programs*) często określane jako standardowe; są to programy typu: sortowania, testowania, przenoszenia informacji między nośnikami (*utilities*), łączenia (*linkage editor*) czy bibliotekarza (*librarian*),

— programy użytkowe — zastosowań.

Programy sterujące maszyny mają na celu koordynację wszystkich funkcji maszyny, począwszy od operowania kluczami jednostki centralnej i wprowadzania programów, przez koordynację działania wszystkich urządzeń zewnętrznych maszyny aż do koordynacji w przetwarzaniu wieloprogramowym, satelitarnym itd. Takimi koordynatorami m. in. są programy wymienione poprzednio w pkt. 2.1.2.

Programy tłumaczące wiążą się ze stosowaniem języków programowania typu autokodów, wymagających przetłumaczenia ich na język wewnętrzny maszyny, w którym odbywa się przetwarzanie. Tłumaczenie to wykonuje komputer automatycznie za pomocą programów tłumaczących (translatorów). Do autokodów można zaliczyć takie języki programowania automatycznego, jak np.: SAKO, MARK, MAC, FORTRAN, ALGOL, COBOL, PL/I, MAL, RAPIDWRITE i inne.

---

[1] Warto podkreślić, że np. według terminologii stosowanej przez firmę IBM, system operacyjny składa się zarówno z programów sterujących, jak i programów przetwarzania. (Por. *IBM System 360 Operating System, Introduction*, File nr S360-20, Form C28-6534-1, s. 9).

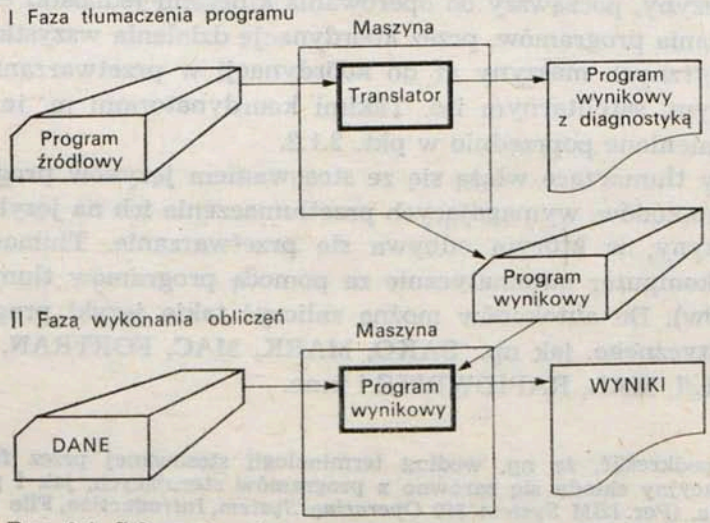
Programy uniwersalno-usługowe, wchodzące w skład biblioteki programów, stosuje się do przetwarzania typowych, często powtarzających się ciągów operacji. Budowa programów uniwersalnych umożliwia ich adaptowanie do różnych sytuacji. Składają się one z segmentów, które dobiera się w zależności od potrzeb. Wybrany segment wywoływany jest kartami sterującymi wypełnianymi przez programistę.

Programy użytkowe są specjalnie napisane dla tych zagadnień, których nie da się rozwiązać za pomocą programów uniwersalno-usługowych. Biblioteki programów uniwersalnych obejmują znaczną liczbę programów (średnio od 200 do 1000 tys. rozkazów). Umiejętność ich zastosowania jest niezwykle ważna przy projektowaniu technologii APD.

Odpowiednie stosowanie języków programowania i programów uniwersalno-usługowych skraca niekiedy kilkakrotnie czas uruchamiania systemów APD.

## 3.2. *Języki programowania*

Programowanie w języku maszyny jest pracochłonne i jest źródłem wielu błędów ze względu na bardzo dużą liczbę szczegółów, które programista w tym wypadku musi uwzględnić. W związku z tym bardzo poważnie rozwija się kierunek projektowania języków programowania coraz to bardziej automatyzujących proces programowania. Programista układając swój program w języku źródłowym (*source language*) angażuje sam komputer do przetłumaczenia tego programu na program wynikowy (*object language*), który kieruje właściwym przebiegiem przetwarzania. Do procesu tłumaczenia komputer wymaga odrębnego programu tłumaczącego — translatora. Na ryc. 3.1 przedstawiamy zależności między pro-




*Ryc. 3.1. Schemat systemu automatycznego programowania*

gramami: źródłowym, translatorem i wynikowym, który może być otrzymany na różnych nośnikach informacji, np.: tabulogramie, kartach lub taśmie dziurkowanej, wykorzystanych w następnej fazie przetwarzania do wykonania właściwych obliczeń.

Istnieje obecnie wiele sposobów systematyzowania języków programowania w zależności od przyjętych kryteriów klasyfikacji. Jeżeli np. za kryterium podziału przyjmiemy stopień zależności źródłowego języka programowania od języka maszyny, to wtedy możemy wyróżnić następujące języki programowania:

a) zależne od komputera (*computer dependent language*), np.: AUTOCODER (IBM 1400), MPL (ICT 1300), PLAN (1900), Mark (Elliott 803), MAT (Mińsk 22), SAS (ZAM 41), JAS (Odra 1204) i inne,

b) problemowe (*problem oriented language*), niezależnie od maszyny, zaprojektowane dla określonej klasy zagadnień, np. języki COBOL, PL/I, ALGOL, FORTRAN, SAKO [2], LISP, EOL [3], SOL i inne.

W innym, dość często przyjmowanym, podziale języków programowania przyjmuje się za kryterium stopień złożoności języka programowania. Zwykle wyróżniane są dwa takie stopnie:

1) języki programowania o niskim stopniu złożoności (*low level language*), a w szczególności języki montujące (*assembly language*), które zasadniczo tłumaczą jeden rozkaz programu źródłowego na jeden rozkaz programu wynikowego (tzn. 1 : 1); do tego typu języków zalicza się m. in. języki: SAS, JAS, MPL 1, Mark II, MAT II itp.,

2) języki programowania o wysokim stopniu złożoności (*high level language*), często określane jako języki proceduralne (*procedure oriented language*); do tego typu języków zaliczany jest przede wszystkim: COBOL, PL/I, FORTRAN, ALGOL.

Programy tłumaczące dla tych języków programowania charakteryzują się dużą złożonością, noszą nazwę translatorów albo kompajlerów (*compiler*).

Dość popularnym kryterium klasyfikującym języki programowania jest ich przeznaczenie użytkowe. Na przykład do:

1) przetwarzania danych stosuje się: COBOL, RAPIDWRITE, PL/I, AUTOCODER, PLAN, MAT i inne,

2) obliczeń numerycznych stosuje się: FORTRAN, ALGOL, PL/I, MAC, Mark, SAKO i inne,

---

[2] System Automatycznego Kodowania SAKO został opracowany w latach 1959/1960 przez zespół naukowy ówczesnego Zakładu Aparatów Matematycznych PAN i był szeroko stosowany na maszynach produkcji krajowej ZAM-2. Język SAKO zbliżony jest do języka FORTRAN II; ma on ponadto wiele cech dodatkowych. (Por. L. Łukaszewicz *SAKO, an Automatic Coding*, Pergamon Press, Annual Review in Automatic Programming Oxford 1961; L. Łukaszewicz, A. Mazurkiewicz, *System automatycznego kodowania SAKO*, PAN, Wrocław 1963).

[3] EOL jest językiem do przetwarzania symboli, przeznaczonym m. in. do pisania translatorów innych języków. Wersja EOL-2 funkcjonuje na maszynach krajowych ZAM 41, wersja EOL-3 na maszynach IBM 7094 oraz IBM/360. Por. L. Łukaszewicz, *Język do przetwarzania symboli EOL*, „Maszyny Matematyczne" 1969, nr 5.

3) przetwarzania symboli (napisów, pisania translatorów, wyszukiwania informacji) stosuje się: COMIT- LISP, IPL V, EOL i inne,

4) konwersacyjnego kontaktu człowiek—maszyna — tzw. języki bezpośredniego współdziałania (*interactive language*), np. QUICKTRAN,

5) obliczeń konstrukcyjnych — STRESS, NET i wiele innych.

Nie wdając się w opisy poszczególnych rodzajów języków programowania, których użytkowe poznanie wymaga przestudiowania oddzielnego podręcznika dla każdego języka, omówimy pewne wspólne tendencje występujące w budowie dotychczas użytkowanych języków. Do tendencji tych zaliczyć można m. in.:

a) mnemoniczne oznaczanie kodu operacji, np. CZYTAJ,

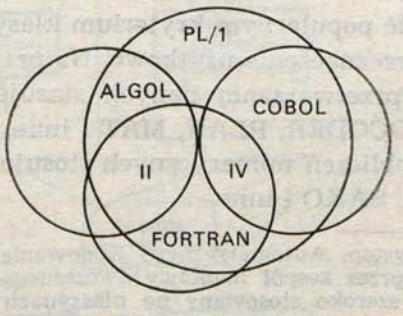
b) symboliczne oznaczanie adresów pamięci operacyjnej względem określonego położenia pamięci operacyjnej, albo za pomocą etykiet lub identyfikatorów, które np. mogą być alfanumerycznymi nazwami adresów pamięci operacyjnej, jak: ADAM, ALFA,

c) stosowanie zdań (*statement*) dla określania nazw podprogramów wejściowo-wyjściowych lub specjalnych, rozbudowanych rozkazów, np. dodawania czy odejmowania czteroadresowego; często zdania te potocznie określane są makrorozkazami,

d) stosowanie pseudorozkazów, które sterują procesem tłumaczenia i nie mają odpowiadających im rozkazów w programie wynikowym,

e) stosowanie podziału programu na określone typy segmentów, np. identyfikacji konfiguracji danych, procedury, który to podział jest w języku COBOL.

Z chwilą pojawienia się w sprzedaży maszyn IBM 360 w latach 1964/ /1965 zauważono wyraźną tendencję ujednolicania języków programowania zarówno dla obliczeń numerycznych, jak i przetwarzania danych. Wyrazem tej tendencji jest powstanie języka PL/I (gdzie I — oznacza nie-




*Ryc. 3.2. Wykorzystanie elementów budowy języków: ALGOL, FORTRAN II, IV, COBOL w budowie języka PL/I*

przetłumaczalne *one*, tzn. mniej więcej „język jedyny do wszystkiego"). Współzależność języków COBOL, ALGOL, FORTRAN i PL/I ilustruje ryc. 3.2. Język programowania PL/I, zaprojektowany głównie z myślą o maszynie IBM 360, jest syntezą dotychczasowych języków proceduralnych, chociaż nie jest z nimi wymienny w żadnym kierunku. Jak dotąd

tylko firma Scientific Data Systems opracowała translatory PL/I dla swoich maszyn. Wydaje się, że ze względu na wymienność z programami IBM, w tym kierunku następuje dalszy rozwój.

Prace nad językiem COBOL rozpoczęto w 1959 r. z inicjatywy administracji Stanów Zjednoczonych. Utworzono Komitet Projektowy, złożony z przedstawicieli czołowych firm produkujących maszyny oraz z niektórych użytkowników. Grupa złożona z przedstawicieli US Navy, US Air Force, US Bureau of Standards, IBM, Burroughs, Honeywell, RCA, Sylvania, Univac — opracowała w kwietniu 1960 r. pierwszą wersję COBOL o nazwie COBOL 60. Wersja ta okazała się mało elastyczna ze względu na trudności dostosowania do niektórych bardziej złożonych problemów. Do Komitetu dokooptowano jeszcze przedstawicieli Chase Manhattan Bank, Boeing, Du Pont, General Electric, General Motors, Lockheed, Standard Oil, US Steel Corp, Westinghouse. Grupa ta opracowała drugą wersję o nazwie COBOL 61. Następną wersją jest COBOL 65, który zawiera instrukcję sortowania oraz umożliwia programowanie przetwarzania wyrywkowego. Większość translatorów oparta jest na koncepcji COBOL 61. W dalszym ciągu opracowuje się translatory COBOL 65. Projektowanie translatorów opiera się na publikacji DOD (Departament Obrony USA), która zawiera pełną listę instrukcji. Projektanci wybierają instrukcje najbardziej przydatne. Dzięki temu pojawiło się kilka odmian języka COBOL, co w pewnym sensie jest ograniczeniem jego uniwersalności.

Na początku nie stosowano oczywiście powszechnie języka COBOL, jedynie firma RCA propagowała ten język. IBM opierał się głównie na języku AUTOCODER, NCR — na NEAT Compiler, Honeywell — na FACT. Dopiero pod koniec lat sześćdziesiątych, zgodnie z zarządzeniami administracji USA, która użytkuje kilka tysięcy maszyn, zabroniono dalszych zakupów maszyn do APD nie posiadających translatorów COBOL; stąd szersze zainteresowanie tym językiem. Wraz ze znacznym wzrostem prędkości maszyn odpada zarzut wolnego tłumaczenia programów napisanych w języku COBOL.

Obserwuje się też inną koncepcję rozwiązania wymienności programów. Jeśli mamy $m$ maszyn i $n$ języków, to należy opracować $mn$ translatorów, co oczywiście jest zadaniem bardzo trudnym i pracochłonnym. W związku z tym zaproponowano wprowadzenie uniwersalnego języka pośredniego, który nazwano UNCOL. Stosując powszechnie przyjęty język pośredni, dokonuje się dwukrotnej translacji: z $n$ autokodów (np. COBOL, FORTRAN itp.) na UNCOL, a następnie z UNCOL na języki wewnętrzne $m$ maszyn. Wtedy liczba translatorów mogłaby wynosić tylko $m+n$. Podstawową trudnością jest tu jednakże osiągnięcie porozumienia co do budowy języka UNCOL, podobnie jak trudno było uzyskać porozumienie np. w sprawie języka COBOL. Ponadto nie jest pewne, czy budowa UNCOL długo

utrzymałaby się na nie zmienionym poziomie wobec szybkiego rozwoju konstrukcji maszyn. Jak dotychczas koncepcja ta nie była praktycznie stosowana.

### *3.3. Język COBOL*

Przedstawimy tylko ogólną koncepcję budowy języka COBOL. Warto podkreślić, że jednym z powodów niezbyt szybkiego rozpowszechniania programowania w tym języku jest dość złożona i na pierwszy rzut oka mało czytelna forma większości podręczników programowania.

Programowanie w języku COBOL sprowadza się do ułożenia czterech rozdziałów programu;

1. Rozdział identyfikacji (*identification division*), w którym podaje się nazwę programu, autora programu, datę itp.

2. Rozdział struktury dotyczący konfiguracji i parametrów maszyny (*environment division*), w którym określa się:

— maszynę użytą do tłumaczenia i zastosowania do eksploatacji,

— strukturę zestawu maszynowego (pojemność pamięci operacyjnej, użyte urządzenia zewnętrzne itp),

— przyporządkowanie poszczególnych zbiorów (określonych w rozdziale danych) poszczególnym urządzeniom zewnętrznym.

3. Rozdział danych (*data division*), w którym opisuje się kartoteki, pozycje i zapisy, przyporządkowując im nazwy symboliczne (etykiety) oraz oznaczając długości i budowę tych danych:

— w zapisie numerycznym stosuje się notację, w której oznacza się pojedyncze pozycje liczby przez 9; liczbę o $n$ pozycjach zapisuje się jako 9 ($n$),

— w zapisie alfanumerycznym każdą literę oznacza się jako $A$, np. $A(12)$ oznacza słowo dwunastoliterowe,

— zapis alfanumeryczny oznacza się jako $X$, np. $X(3)$ oznacza słowo składające się z trzech znaków.

Opisu danych dokonuje się w trzech sekcjach: sekcji kartotek (*file section*), sekcji pamięci roboczej (*working storage section*), sekcji stałych (*constant section*).

4. Rozdział procedury (*procedure division*) zawiera listę rozkazów, które określają działania na danych. W języku COBOL występuje 13 rozkazów podstawowych i kilkanaście uzupełniających. Rozkazami podstawowymi można zakodować każdy program, jednak nie tak efektywnie jak przy użyciu wszystkich rozkazów. Rozkazy uzupełniające podwyższają efektywność programu i dają pewne dodatkowe ułatwienie. Do rozkazów podstawowych zalicza się:

— rozkazy arytmetyczne: ADD (dodawaj), SUBTRACT (odejmuj), MULTIPLAY (mnóż), DIVIDE (dziel),

— rozkazy logiczne IF (jeśli, porównania),
— rozkazy przesyłania MOVE (przesuń),
— rozkazy pomocnicze: GO TO (idź tu — skok bezwarunkowy), PERFORM (skok do podprogramu ze śladem), STOP,
— rozkazy wejściowo-wyjściowe: OPEN (otwórz kartotekę), CLOSE (zamknij kartotekę), READ (czytaj), WRITE (zapisz).

Do rozkazów uzupełniających zalicza się m. in.: ACCEPT (pobierz), DISPLAY (wypisz), COMPUTE (oblicz), TIMES (wykonaj tyle razy)[4], UNTIL (wykonaj aż do spełnienia warunku), VARYING (wykonaj cyklicznie zmieniając argumenty), EXAMIN (sprawdź), DEFINE (określ), INCLUDE (włącz).

Jak mówiliśmy (por. pkt 3.2.), listy rozkazów poszczególnych wersji języka COBOL mogą się różnić między sobą. Niektóre wersje odróżniają się nazwami, np. duży i mały COBOL.

W języku COBOL zdania występują w następującej postaci:

ADD PŁACA—BRUTTO TO DODATKI

gdzie „ADD TO" (dodaj do) jest rozkazem, natomiast „PŁACA—BRUTTO" i „DODATKI" jest nazwą symboliczną (etykietą). W rozwiniętych systemach COBOL nazwa może mieć do 30 znaków długości, przy czym w nazwie musi wystąpić co najmniej jedna litera i to na początku etykiety, a ponadto w nazwie nie może być odstępów (spacji).

Nazwy definiowane w rozdziale danych, a także nazwy plików (np. kartotek), których dane przetwarzane są na urządzeniach zewnętrznych, są przyporządkowywane tym urządzeniom w rozdziale struktury.

Każde zdanie w języku COBOL składa się z jednego lub kilku elementów języka, ułożonych w sensowny sposób, zrozumiały dla translatora. Translator podczas tłumaczenia posługuje się regułami, które umożliwiają mu zidentyfikowanie końca jednego elementu i początku następnego. Najogólniejszą regułą, którą powinien zapamiętać programista posługujący się tym językiem, jest oddzielanie sąsiednich elementów jednym lub kilkoma odstępami.

Trzy pierwsze rozdziały mają charakter deklaracji. Natomiast właściwe programowanie odbywa się w rozdziale procedury. Liczba rozkazów jest ograniczona, programowanie jest dość proste. Z kolei nazwy argumentów rozkazów są nazwami występującymi w danym systemie przetwarzania danych; wskutek tego programowanie bardziej uwzględnia wymagania związane z problemem, a mniej — z maszyną.

---

[4] UNTIL, VARYING i TIMES występują łącznie z innymi rozkazami, np. PERFORM ... UNTIL, PERFORM ... VARYING, PERFORM ... TIMES.

## 3.4. Język APL

A Programming Language został opracowany i opublikowany w 1962 r. przez K. E. Iversona, a następnie dopracowany przez A. D. Falkoffa, L. M. Breeda i R. D. Moora. APL jest przede wszystkim stosowany w obliczeniach konwersacyjnych. Jego największą zaletą jest prostota i uniwersalność.

Typowa konwersacja użytkownika końcówki z komputerem może być następująca:

| komputer | użytkownik końcówki |
|---|---|
| 9 | 3,4+5,6 |
| | 3,4×7 |
| 23,8 | |
| | X←3 |
| | Y←5 |
| | X×Y |
| 15 | |
| | X+Y |
| 8 | |
| | (X+Y)×(X−Y) |
| −16 | |

Działania na wektorach są wykonywane równie łatwo.

```
X←2 3 5 7 11
Y←2 0 2 0 1
X+Y
4 3 7 7 12
X×Y
4 0 10 0 11
X×Y (potęgowanie)
4 1 25 1 11
2 × X
4 6 10 14 22
```

W podobny sposób równie łatwo prowadzi się operacje na znakach alfanumerycznych. Autor APL — K. Iverson wyjaśnił, że głównym motywem uruchomienia tego języka była chęć udoskonalenia zapisu algorytmów stosowanych w typowych zagadnieniach uniwersyteckich. W ślad za Uniwersytetem Harvardzkim stosującym APL, z którego wywodzi się K. Iverson, podążyły inne uczelnie, a następnie firma IBM i ośrodki usługowe obliczeń abonenckich. Większość komputerów wykorzystywanych w obliczeniach numerycznych dysponuje translatorami języka APL. Coraz szersze zastosowanie znajduje APL w: projektowaniu oprogramowania, układów logicznych komputerów, programowanym nauczaniu, wyszukiwaniu informacji naukowo-techniczno-ekonomicznych, a nawet w przetwarzaniu danych.

### 3.5. *Wybór języka programowania*

Od wybranego języka programowania systemu zależy:
— metodyka szkolenia projektantów,
— ilość popełnianych błędów w programach,
— czas uruchamiania programów,
— czas przetwarzania programów,
— czas przetwarzania eksploatacyjnego.

W autokodach skraca się programowanie — o czym już mówiliśmy — ale wydłuża czas przetwarzania eksploatacyjnego średnio o 8 do 10%. Dla programów często powtarzanych i przy częstym wykorzystaniu drogich maszyn powinien być przeprowadzony odpowiedni rachunek ekonomiczny.

Powszechnie popełnianym błędem w szkoleniu programistów jest rozpoczynanie wykładów od nauczania programowania w języku wewnętrznym maszyny. Duże nagromadzenie szczegółów, spora trudność w przyswajaniu materiału powodują, że nauczanie w drugiej kolejności programowania w językach rozwiniętych, np. COBOL, MAL, RAPIDWRITE, BEST, nie dochodzi do skutku lub jest prowadzone pobieżnie, bez położenia należytego nacisku na prawidłowe i skuteczne przeszkolenie.

Nic więc dziwnego, że, w konsekwencji, oprogramowania systemu dokonuje się wolno i z dużymi trudnościami. Oczywiście zalecane jest zaznajomienie z programowaniem w języku wewnętrznym (kodzie) maszyny; może to być ewentualnie wykorzystane do tych wypadków, kiedy rozwiązanie autokodowe nie jest optymalne. Wówczas stosuje się włączanie tzw. wstawek do programu w kodzie maszyny. Inne zastosowanie kodu maszyny dotyczy układania programów standardowych, które na ogół wykonywane są przez producenta maszyny.

Obecnie przeprowadzimy porównania wydajności techniki programowania w języku PGF (*Programme Gestion de Fichier*) (adresy symboliczne i makrorozkazy dla urządzeń zewnętrznych) z językiem COBOL przy zastosowaniu maszyn GAMMA 30 [5].

Eksperyment ten został przeprowadzony przy udziale programistów producenta (firmy BULL) w języku PGF i przy udziale programistów użytkownika (Direction de la Compatililité Publique) w autokodzie COBOL. Programiści układający program w języku PGF mieli bardzo dobre przygotowanie i praktykę programowania w tym języku. Natomiast programiści układający program w języku COBOL przystąpili do swojego zadania bezpośrednio po kursie, bez żadnej praktyki w programowaniu (z wyjątkiem jednej osoby). Jako przedmiot eksperymentu obrano zagadnienie rozliczania płac, od dawna opracowywane za pomocą takich maszyn, jak Seria 300 MCT, IBM 1401.

---

[5] RCA, 301, ICT 1500, podobna do IBM 1410.

Programowanie w autokodzie PGF objęło 25 przebiegów i dało rezultaty, które przedstawiamy w tablicy 3.1. Programowanie w COBOL objęło tylko 16 przebiegów, nie obejmując organizacji systemu, która została przedstawiona tej grupie w postaci sformalizowanej. Ponadto dużą

Tablica 3.1

*Pracochłonność faz programowania i uruchamiania*

| Wyszczególnienie | Liczba godzin | Średnio na przebieg |
|---|---|---|
| Organizacja systemu | 1 500 | 60 |
| Programowanie | 900 | 34 |
| Uruchamianie | 1 000 | 40 |
| Razem programowanie i uruchamianie | 1 900 | 74 |
| Czas maszyny | 75 | 3 |
| Liczba rozkazów autokodowych | 8 000 | 320 |
| Pracochłonność 1 rozkazu autokodu | ok. 0,25 godz. | |
| — programowanie i uruchamianie | ok. 0,25 godz. | |
| — projektowanie | ok. 0,45 godz. | |

część czasu zajęło doszkalanie programistów, którzy nie mieli właściwie praktyki. Rezultaty przedstawiamy w tablicy 3.2. Odliczając straty na ułożenie pierwszych czterech programów, wynikające z braku praktyki programistów, można przyjąć, że czas programowania jednego przebiegu powinien wynosić 12 godzin.

Tablica 3.2

*Pracochłonność faz uruchamiania i programowania w przeliczeniu na 1 przebieg*

| Wyszczególnienie | Liczba godzin | Średnio na przebieg | Liczba przebiegów, dla których liczono czas |
|---|---|---|---|
| Programowanie | 250 | 16 | 15 |
| Tłumaczenie programów uruchamiania | 33 | 2h 10′ | 15 |
| | 5h 30′ | 0h 30′ | 12 |
| Razem uruchamianie | 30h | 2h 30′ | |
| Liczba rozkazów w autokodzie COBOL | | | |
| w tym: | 3 656 | 228 | 16 |
| — rozdział danych | 1 510 | 94 | |
| — rozdział procedury | 2 146 | 134 | |

Z porównania wynika, że pracochłonność programowania jednego przebiegu w języku COBOL jest prawie 3-krotnie mniejsza od pracochłonności programowania w autokodzie PGF (34 : 12).

Strukturę poszczególnych programów podajemy w tablicy 3.3.

W autokodzie PGF więcej niż 50% wszystkich programów zawiera ponad 300 rozkazów, podczas gdy w języku COBOL prawie 85% programów zawiera mniej niż 300 rozkazów.

Tablica 3.3

*Długość programów w PGF i COBOL*

| Rozkazy lub pseudorozkazy | Liczba programów | |
|---|---|---|
| | PGF | COBOL |
| poniżej 100 | — | 2 |
| 100—200 | 4 | 8 |
| 200—300 | 5 | 4 |
| 300—500 | 1 | |
| 500—700 | 4 | |
| powyżej 700 | | |

Analizę struktury programów pod względem prędkości zredagowania podajemy w tablicy 3.4. Z zestawienia tego wynika, że 50% wszystkich programów było układanych z prędkością około 10 rozkazów/godz., podczas gdy w COBOL, poza pierwszymi programami szybkość redagowania była większa.

Tablica 3.4

*Wydajność programowania w PGF i COBOL*

| Rozkazów/godzinę | Liczba programów | |
|---|---|---|
| | PGF | COBOL |
| mniej niż 10 | 8 | 4[a] |
| 10—20 | 2 | 4 |
| 20—30 | 4 | 4 |
| 30—40 | 2 | 2 |

[a] Dotyczy pierwszych czterech programów.

Podobny charakter ma analiza (por. tablicę 3.5), z której wynika, w ciągu ilu dni poszczególne programy były zaprogramowane. Z tablicy 3.5 wynika, że większość programów pisanych w języku COBOL była

Tablica 3.5

*Okres programowania w PGF i COBOL*

| Liczba dni | Liczba programów | |
|---|---|---|
| | PGF | COBOL |
| 1 | 4 | 7 |
| 2 | 2 | 3 |
| 3 | 3 | 2 |
| 4 | | 1 |
| 5 | | 1 |
| 6 | 1 | 1 |
| 7 | | |
| 8 | 2 | |
| 9 | 1 | |
| 10 | 1 | |
| 10 | 3 | |

redagowana w okresach krótszych niż tydzień, podczas gdy dla autokodu PGF większość programów wymagała dłuższego czasu niż tydzień.

Przy rozpatrywaniu zagadnienia łatwości uruchamiania programów należy przeanalizować: a) obciążenie programistów, b) obciążenie maszyny, c) liczbę tłumaczeń programu.

Programowanie w języku COBOL zmniejsza liczbę błędów w redagowaniu. Uruchamianie jednego programu w PGF wymagało średnio 40 godzin pracy programisty, a 10 godzin dla przeciętnego programu w języku COBOL. Średnio jeden program w języku COBOL wymagał 3 do 4 tłumaczeń oraz 3 testów, podczas gdy liczba przebiegów maszyny dla tłumaczenia i testowania programów w PGF wynosiła około 18. Obciążenie maszyny podczas uruchamiania programu ułożonego w PGF wynosiło średnio 3 godziny, a 2,5 godzin dla programu w języku COBOL. Przy bardziej szczegółowej analizie tego czasu stwierdza się, że samo uruchamianie w pierwszym wypadku wymagało 1,75 godziny, a 0,5 godziny w drugim.

Czas tłumaczenia programów w języku COBOL w stosunku do programów w PGF jest dłuższy średnio o 20%. Jednakże uzyskuje się skrócenie rzędu 75% na uruchamianiu programu, czyli tej fazy, która wymaga największego udziału pracy ludzkiej.

Wypełnienie pamięci operacyjnej dla programów w języku COBOL jest średnio o 30% większe niż dla programów w PGF. Analizę wydajności programów pisanych w tych dwóch językach przeprowadzono na podstawie pomiaru czasów przetwarzania tych samych danych w tej samej liczbie. Ogólnie biorąc, czasy przetwarzania są ekwiwalentne, co przedstawiamy w tablicy 3.6.

Tablica 3.6

*Czas eksploatacji programów w językach PGF i COBOL*

| Numer programu | Czas w sekundach | |
|---|---|---|
| | PGF | COBOL |
| 55 | 160 | 185 |
| 60 | 145 | 125 |
| 100 | 80 | 95 |
| 110 | 100 | 142 |
| 120 | 308 | 320 |
| 130 | 47 | 44 |
| 201 | 45 | 47 |
| Razem | 885 | 968 |

Program ułożony w języku COBOL był przeciętnie o 10% wolniejszy od programu ułożonego w PGF, ale przy rozpatrywaniu poszczególnych programów okazuje się, że dominacja jednego lub drugiego języka jest zmienna.

Reasumując porównanie można stwierdzić, że programowanie w języku COBOL jest krótsze prawie 3-krotnie, uruchamianie zaś jest również krótsze i to 4-krotnie. Obciążenie EMC w obu wypadkach jest porównywalne. Natomiast wypełnienie PAO w języku COBOL jest o 30% większe, jak również czas przetwarzania może być o 10% dłuższy.

Przy wyborze języka programowania istotna jest wymienność programów między różnymi maszynami, w wypadku awarii.

Istnieje przekonanie, że programy w zakresie przetwarzania danych mogą być układane wyłącznie w językach wyspecjalizowanych. Okazuje się jednak, że wiele programów z tej dziedziny zastosowań można redagować w językach przystosowań do obliczeń numerycznych, o czym świadczy stosunkowo szerokie stosowanie języka FORTRAN IV.

W Zakładzie Elektronicznej Techniki Obliczeniowej (ZOWAR Warszawa) wykonano wiele programów na maszynie typu ICT 1300 (przy współpracy z taśmami magnetycznymi) w języku MAC, wybitnie przystosowanym do obliczeń numerycznych. W wypadkach gdy problem przetwarzania danych alfanumerycznych nie jest podstawowy, może być polecane wykorzystanie prostych autokodów.

### *3.6. Programy tłumaczące*

Translatory są bardzo złożonymi i długimi programami, np. translator COBOL liczy średnio kilkadziesiąt tysięcy rozkazów w zależności od rodzaju maszyny. Przy ocenie translatora bierze się pod uwagę minimalną konfigurację zestawu komputera (na którym można realizować przetwarzanie w danym języku programowania), czas tłumaczenia oraz rozwiązania, a także system diagnostyczny oceniający prawidłowość ułożenia programu.

Przedstawimy ogólną charakterystykę translatorów COBOL i ich głównych producentów rynku europejskiego.

ICL dysponuje dwoma kompilatorami: COMPACT COBOL i COBOL. COMPACT COBOL (wersja uproszczona) został uruchomiony dla maszyn ICL 1900 z czterema jednostkami taśm magnetycznych i z pamięcią operacyjną o pojemności 8192 słów. Uproszczona wersja COMPACT zawiera minimum listy zatwierdzonej przez Departament Obrony Stanów Zjednoczonych (mniej więcej połowę) oraz rozkaz ENTER (wprowadź do programu w COBOL rozkazy, np. w autokoderze), który ułatwia pisanie wstawek w niższych językach i rozkaz USAGE (zastosuj), który umożliwia przetwarzanie danych szterlingowych oraz w zapisie dwójkowym. Pełna wersja języka COBOL dostępna będzie dla 16 384 słów 24-bitowych pamięci operacyjnej i czterech jednostek taśm magnetycznych.

Honeywell dysponuje czterema wersjami COBOL — B, D, H, I. Wersja B wymaga 8192 znaków pamięci operacyjnej, trzech jednostek taśm