8 1984

Informatyka

Szkoły Mikroprocesorowe
Informatyczna edukacja
Spirala Ulama

WYDAWNICTWO NOT SIGMA

8 1984

Rok XIX

1984

Organ Komitetu Informatyki MNSzWiT oraz Komitetu Naukowo-Technicznego NOT ds. Informatyki

Materiałów nie zamówionych redakcja nie zwraca

Redakcja: 00-041 Warszawa, ul. Jasna 14/16, pok. 243 i 244, tel. 27-71-40 lub 26-82-61 w. 184

Zakł. Graf. „Tamka". Zam. 2181. Obj. 4,0 ark. druk. Nakład 4200 egz. T-29.

INDEKS 36124

Cena egzemplarza zł 75,—
Prenumerata roczna zł 900,—

W NUMERZE:

Strona

FORTH — język i system programowania (3)
Wojciech Trojniar 1

Roboty (3). Proste języki programowania
Cezary Zieliński 6

Programowanie w języku PL/M (2)
Jerzy Dańda, Zbigniew Poznański 9

mikroKLAN 13
— O jabłku opowieść (2)
— Przenoszenie informacji z komputerów dużych na osobiste
— Jak zainstalować FORTH (1)
— Bank Danych — CSK

SAMOTESTY 21
— I/B. Indeksowanie danych i przeszukiwanie zbiorów

Z KRAJU 22
— Szkoły Mikroprocesorowe *(Jacek Żebrowski, Jerzy Dańda, Andrzej J. Piotrowski)*

ZE ŚWIATA 24
— Informatycy zatrudnieni (Francja)
— Nauczanie wspomagane komputerem (USA)
— Sposób na adepta (Wielka Brytania)
— Chronić dzieło programisty!
— Urządzenie do automatycznej analizy fotografii nieba
— Komputer HP 71B
— BCS
— Komputerowe okno na świat

TERMINOLOGIA 31
— Dokumentacja oprogramowania (2)

POGLĄDY okł.
— Kupić — nie kupić

CZWARTA OKŁADKA — *Rafał Pietrak, Jakub Tatarkiewicz*

W NAJBLIŻSZYCH NUMERACH:

- Roman Żelazny o narzędziach inżynierii oprogramowania
- Andrzej Szałas o systemach ekspertowych
- Ewa Gutman o bazie danych obsługiwanej przez komputer pośredniczący
- Władysław Udrycki o języku CHILL
- Jerzy Szyller o mikroprocesorach lat osiemdziesiątych
- Józef de Mezer o sterowaniu alfaskopem
- Wacław Iszkowski o języku ojczystym informatyków

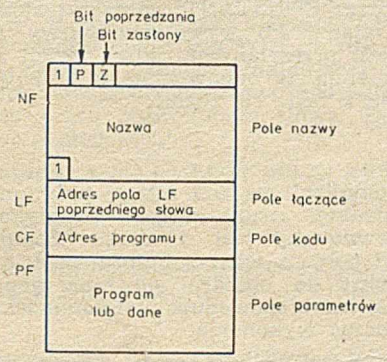
WOJCIECH TROJNIAR
Zakład Systemów Automatyki Kompleksowej
Polska Akademia Nauk
Gliwice

FORTH—język i system programowania (3)

W poprzednich częściach artykułu opisano model FORTHA, strukturę systemu, w którym wykonywane są programy, oraz znaczenie niektórych słów używanych do programowania w tym języku. Poniżej omówiono sposób implementacji kodu wynikowego, sposób kompilacji programów oraz przykładowy program w języku FORTH.

IMPLEMENTACJA FORTHA

Kod wynikowy programu w języku FORTH ma pewną szczególną postać, która decyduje o jego właściwościach. Każde słowo znajdujące się w słowniku zajmuje pewien spójny obszar pamięci. Kod wynikowy słowa składa się z czterech pól (rys. 1). Pola **NF** i **LF** tworzą nagłówek składający się z nazwy oraz łącznika do pozostałych słów w słowniku. Na pozostałych dwóch polach znajduje się kod programu i danych. Pole kodu (**CF**) zawiera bezpośredni adres programu, który zostanie wykonany po wywołaniu słowa. Tak więc pole kodu nadaje interpretację polu parametrów (**PF**).



Rys. 1. Struktura kodu słowa

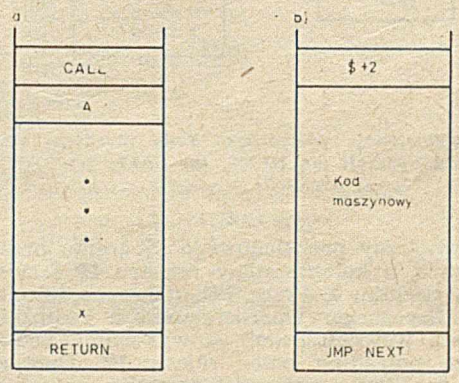
W większości przypadków programy słów mają postać kodu nizanego. W znanych implementacjach języka FORTH najczęściej używa się pośredniego kodu nizanego. Program w tym kodzie składa się z sekwencji wywołań uprzednio zdefiniowanych procedur. W kodzie nizanym określone są trzy operacje:

CALL — wywołanie podprogramu w kodzie nizanym
RETURN — powrót z podprogramu w tym kodzie
NEXT — zainicjowanie wykonania programu i przesunięcie wskaźnika interpretacji do adresu kolejnego wywołania.

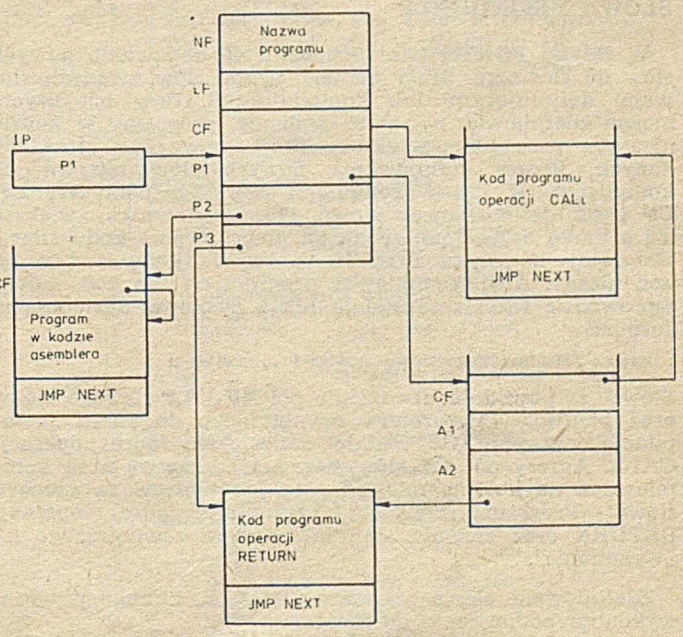
Interpreter kodu zawiera wskaźnik interpretacji **IP** oraz stos. Operacja CALL pozostawia na stosie aktualny stan wskaźnika interpretacji. Podczas operacji RETURN następuje pobranie wartości znajdującej się na szczycie stosu i przesłanie jej do rejestru **IP**. Podstawowym kodem nizanym jest kod natychmiastowy. Program w tym kodzie jest sekwencją instrukcji asemblerowych:

```
CALL P1
CALL P2
...
CALL Pn
RET
```

Wskaźnikiem interpretacji jest tu licznik operacji. Operacja NEXT polega na zwiększeniu licznika operacji w chwili wykonania instrukcji CALL. Procedura wywołana przez instrukcję CALL może być programem w kodzie nizanym lub programem w kodzie maszynowym kończącym się instrukcją RET.



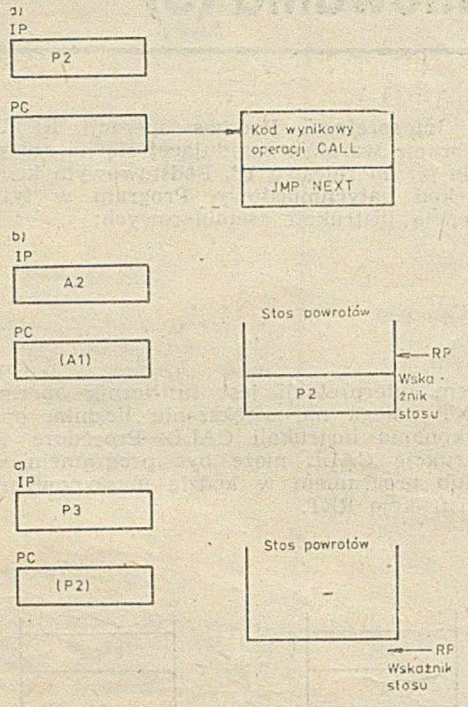
Rys. 2. Postać programu — a) kod nizany, b) kod maszynowy



Rys. 3. Kod wewnętrzny przykładowego programu

Mgr inż. WOJCIECH TROJNIAR ukończył studia na Wydziale Automatyki Politechniki Śląskiej. Od 1976 roku pracuje w Zakładzie Systemów Automatyki Kompleksowej PAN w Gliwicach, gdzie zajmuje się oprogramowaniem aplikacyjnych systemów mikrokomputerowych.

Strukturę programu w pośrednim kodzie nizanym przedstawiono na rysunku 2. Kod programu rozpoczyna się od adresu początku operacji CALL. W następnych komórkach umieszczona jest sekwencja adresów pól **CF** słów, które są kolejno wywoływane w czasie wykonywania programu. Kod jest zakończony adresem operacji RETURN. Rejestr **IP** zawiera adres pola **CF** słowa, które ma być wykonane. Stos powrotny jest używany do przechowywania śladów programów.



Rys. 4. Implementacja pośredniego kodu nizanego; zmiany stanu po wykonaniu operacji (a), NEXT, (b) CALL, (c) RETURN

Kod wewnętrzny przykładowego programu przedstawiono na rysunku 3, a kolejne stany rejestru **IP** i stosu powrotów — na rysunku 4. Przez **PC** oznaczono licznik operacji procesora fizycznego. Operacja NEXT realizuje funkcję „interpretera wewnętrznego" — wykonanie kodu nizanego wskazanego pośrednio przez rejestr **IP**. Operacja CALL powoduje przesłanie na stos powrotów aktualnego stanu rejestru **IP**, skąd jest on następnie pobierany w trakcie operacji RETURN.

SŁOWA DEFINIUJĄCE

W języku FORTH istnieje możliwość dodawania nowych słów do słownika. Służy do tego grupa słów zwanych słowami definiującymi lub kompilatorami. Przy ich użyciu można zdefiniować np. stałe, zmienne, programy w kodzie nizanym oraz inne struktury programowe lub struktury danych. Proces definiowania nowych słów nazywa się kompilacją. W czasie kompilacji powstają pola **NF**, **LF**, **CF** i **PF** definiowanego słowa. Obszar słownika zwiększa się o liczbę bajtów potrzebną na przechowanie kodu słowa. Programy w języku FORTH są zwykle kompilowane na kod nizany. Aby skompilować program na ten kod, należy wprowadzić tekst z terminala lub z dysku w następującej formie:

```
: nazwa-definiowanego-słowa nazwa-1 ... nazwa-n ;
```

Słowo „:" powoduje utworzenie pól **NF** i **LF** nowego słowa oraz przejście interpretera zewnętrznego do stanu „kompilacja". W polu **CF** zostaje umieszczony adres operacji CALL. Adresy pól **CF** słów **nazwa-1**, ..., **nazwa-n** są kompilowane na kolejne miejsca pola parametrów tworzonego słowa. Program słowa „;" kompiluje adres operacji RETURN oraz zmienia stan interpretera zewnętrznego na „wykonanie".

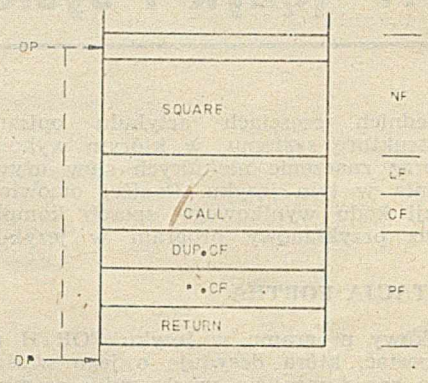
Zdefiniowane słowo o nazwie SQUARE, którego program wykonuje podnoszenie do sześcianu.
stosu:

```
: SQUARE DUP * ;
```

Zmianę zawartości słownika po definicji powyższego słowa przedstawiono na rysunku 5. Napisy **DUP.CF** i ***.CF** oznaczają adresy pól **CF** odpowiednich słów. Nazwy SQUARE można teraz używać do definiowania kolejnych słów, np. program:

```
: CUBE DUP SQUARE * ;
```

wykonuje podnoszenie do sześciu.



Rys. 5. Zmiana zawartości słownika po kompilacji słowa SQUARE

Jeżeli zostanie zdefiniowane słowo o nazwie już istniejącej w słowniku, to na ekranie terminala pojawi się napis:

nazwa-słowa NOT UNIQUE

Dane 16-bitowe można definiować, jak wiemy, używając słów CONSTANT i VARIABLE. Po wywołaniu słowa zdefiniowanego przez CONSTANT, na szczyt stosu przesyłana jest wartość stałej. Po wywołaniu nazwy zmiennej, na szczyt stosu jest przesyłany jej adres. Jeżeli zatem **X** jest stałą, a **Y** zmienną, to odpowiednie instrukcje przypisania **X=5*X** i **X=5*Y** będą się różnić:

```
X 5 * X !
Y 5 * X !
```

W czasie kompilacji na kod nizany programy niektórych słów np. instrukcji strukturalnych, są wykonywane gdy interpreter zewnętrzny jest w stanie „kompilacja". Program słowa ." pobiera tekst z bufora wejściowego i kompiluje ten tekst do obszaru słownika poprzedzając go adresem słowa, które spowoduje przesłanie tekstu na ekran terminala podczas wykonania.

Użytkownik może tworzyć własne programy kompilacji przy użyciu słowa definiującego „:" oraz słów o programach wykonywanych w czasie kompilacji. W stanie „kompilacja" interpreter zewnętrzny rozpoznaje te słowa przez sprawdzenie stanu bitu poprzedzania (ang. precedence) w polu nazwy. Jeżeli bit jest ustawiony, to program odpowiedniego słowa jest wykonywany, a nie kompilowany. Bit można ustawić po zakończeniu definicji słowa, przez wykonanie programu słowa IMMEDIATE.

Poniżej podano kilka przykładów definicji słów (teksty napisane przez maszynę są podkreślone).

```
: SAY-SOMETHING ." COMPILED" ; IMMEDIATE OK
: PROGRAM SAY-SOMETHING ; COMPILED OK
: COM-PROGRAM [COMPILE] SAY-SOMETHING ; OK
COM-PROGRAM COMPILED OK
:PRINT-COM [ COM-PROGRAM ] ." PROGRAM " ; COMPILED OK
```

Program słowa SAY-SOMETHING jest wykonywany w czasie kompilacji. W definicji słowa COM-PROGRAM słowo **[COMPILE]** poprzedza słowo SAY-SOMETHING, co powoduje kompilację słowa SAY-SOMETHING. W definicji słowa PRINT-COM słowo COM-PROGRAM jest poprzedzone słowem „[", które powoduje przejście interpretera zewnętrznego do stanu „interpretacja". Słowo „]" oznacza powrót do stanu „kompilacja".

Skompilowany program można usunąć z pamięci operacyjnej przy użyciu słowa FORGET. W czasie kompilacji pewnych niestandardowych struktur programowych często używa się apostrofu (słowa ') służącego do znajdowania adresu pola parametrów innego słowa, napisanego bezpośrednio po nim (w buforze wejściowym terminala lub dysku).

DEFINIOWANIE SŁÓW DEFINIUJĄCYCH

Najciekawszą właściwością FORTHA jest konstrukcja umożliwiająca definiowanie nowych słów definiujących, dzięki czemu można definiować kompilatory struktur danych lub struktur programowych:

```
: nazwa-słowa-definiującego <BUILDS program-kompilacji
                             DOES> program-wykonawczy ;
```

Po wywołaniu słowa, zdefiniowanego w powyższy sposób następuje utworzenie — przy użyciu słowa **<BUILDS** — pól **NF** i **LF** (**0 CONSTANT**) nowego słowa. Program kompilacji najczęściej tworzy strukturę słowa. Wykonanie programu zdefiniowanego słowa rozpoczyna się od umieszczenia na szczycie stosu adresu pola parametrów **PF** tego słowa, po czym następuje wykonanie właściwego programu, umieszczonego między słowami **DOES>** i „;".

Słowo VARIABLE zdefiniowane przy użyciu tej konstrukcji ma postać:

```
: VARIABLE <BUILDS , DOES> ;
```

Po wywołaniu słowa VARIABLE, wartość znajdująca się na szczycie stosu zostanie umieszczona w polu parametrów definiowanego słowa (zapis wartości początkowej definiowanej zmiennej). Program:

```
5 VARIABLE FIVE
```

Spowoduje umieszczenie wartości **5** w polu **PF** słowa FIVE. Po wywołaniu słowa FIVE, na szczycie stosu zostanie umieszczony adres pola **PF** (adres zmiennej), po czym zakończy się wykonanie słowa, ponieważ między słowami **DOES>** i „;" nie ma żadnego programu.

Utworzymy słowo VECTOR definiując tablicę jednowymiarową o zakresie indeksów **0..I-1**, przy czym wartość **I** będzie znajdować się na szczycie stosu przed definicją tablicy. Wartości początkowe elementów tablicy powinny wynosić **0** w całym zakresie indeksów. Po wywołaniu słowa zdefiniowanego przez VECTOR na stos zostanie przesłany adres komórki pamięci, znajdującej się wewnątrz tablicy i odpowiadającej indeksowi znajdującemu się na szczycie stosu przed wywołaniem słowa. Definicja słowa VECTOR jest następująca:

```
: VECTOR <BUILDS 0 DO 0 , LOOP DOES> SWAP DUP + + ;
```

Wywołanie:

```
5 VECTOR TABLICA
```

spowoduje utworzenie tablicy jednowymiarowej o zakresie indeksów 0..4, a program:

```
3 TABLICA
```

spowoduje przesłanie na szczyt stosu adresu odpowiadającego indeksowi 3 słowa TABLICA.

PRZYKŁADOWY PROGRAM — EDYTOR PUNKTOWY

Program (rys. 6a, 6b) służy do wprowadzania rysunków z klawiatury na ekran monitora. Użytkownik może przesuwać kursor (widoczny w postaci migającego punktu) po ekranie w ośmiu kierunkach. Ruchem kursora można sterować w jednym z trzech trybów pracy: wstawienie punktu, usunięcie punktu i przesuwanie.

Linie ekranu zawierają 640 punktów. Każdej linii odpowiada 80 bajtów ciągłego obszaru pamięci obrazu. Do przesuwania kursora przeznaczono osiem przycisków klawiatury numerycznej. Kierunek przesunięcia kursora po naciśnięciu danego klawisza odpowiada jego położeniu względem klawisza środkowego klawiatury numerycznej, w następujący sposób: 1 — kierunek **WS**, 2 — **S**, 3 — **ES**, 4 — **W**, 5 — bez zmian, 6 — **E** itd. Klawisze **D**, **R**, **M** ustawiają tryb pracy odpowiednio na: wstawianie, usuwanie, przesuwanie.

Edytor punktowy jest wywoływany po interpretacji słowa DOT-EDITOR. Kursor ustawia się w środkowe położenie ekranu, a rysunek jest wprowadzany przez sterowanie trybami pracy i przesuwanie kursora po ekranie. Zakończenie pracy następuje po naciśnięciu klawisza **CTRL-C**.

Tekst programu rozpoczyna się od kadru dyskowego o numerze **200**. Kody znaków są zdefiniowane przez stałe. Zmienna POINT służy do tymczasowego przechowywania bajtu wskazanego przez kursor. Zmienna SWITCH przechowuje kod trybu pracy. Podczas pracy edytora na

```
SRC # 200
  0 ( GED - PROGRAM WPROWADZANIA RYSUNKOW NA MONITOR )
  1 HEX          0140 CONSTANT TOPSCREEN   3E80 CONSTANT ENDSCREEN
  2 36 CONSTANT RIGHT          34 CONSTANT LEFT
  3 38 CONSTANT UP             32 CONSTANT DOWN
  4 31 CONSTANT LEFT-DOWN      37 CONSTANT LEFT-UP
  5 33 CONSTANT RIGHT-DOWN     39 CONSTANT RIGHT-UP
  6 44 CONSTANT DRAW           52 CONSTANT ERASE
  7 4D CONSTANT MOVE           0D CONSTANT HOME
  8 0 VARIABLE POINT           0 VARIABLE SWITCH
  9 --)

SRC # 201
  0 ( GED - PROGRAM WPROWADZANIA RYSUNKOW NA MONITOR )
  1 ( SPRAWDZENIE CZY ADRES BIEZACEGO KURSORA - RSA -)
  2 ( PRZEKRACZA ROZMIARY EKRANU                     )
  3
  4 ( SPRAWDZENIE DOLNEGO OGRANICZENIA   RSA -) RSA )
  5 : ROLLDOWN BUF ENDSCREEN - 1 ) IF ENDSCREEN - ENDIF ;
  6 ( SPRAWDZENIE GORNEGO OGRANICZENIA   RSA -) RSA )
  7 : ROLLUP DUP 0( IF ENDSCREEN + ENDIF ;
  8 ( POBRANIE Z KADRU BAJTU WSKAZANEGO PRZEZ KURSOR )
  9 ( MASK RSA -) MASK RSA MASK BAJT                 )
 10 : TAKE-BYTE OVER OVER TOPSCREEN + C@ ;
 11 ( USUNIECIE I USTANIENIE BITU POD KURSOREM       )
 12 ( MASK RSA -) MASK RSA                           )
 13 : RESET-POINT TAKE-BYTE SWAP OFF XOR AND OVER TOPSCREEN + C! ;
 14 : SET-POINT TAKE-BYTE OR OVER TOPSCREEN + C! ;
 15 --)

SRC # 202
  0 ( GED - PROGRAM WPROWADZANIA RYSUNKOW NA MONITOR )
  1                                                  )
  2 ( AKTUALIZACJA PUNKTU NA EKRANIE                 )
  3 ( JEZELI SWITCH MA WARTOSC 1, TO WSTAWIENIE,     )
  4 ( JEZELI = 2, TO USUNIECIE BITU MASK             )
  5 ( MASK RSA -) MASK MSA                           )
  6 : ACT DUP TOPSCREEN + C@ POINT C! SWITCH @ DUP IF
  7   1 - IF SET-POINT 0 ELSE RESET-POINT 0 ENDIF ENDIF DROP ;
  8 ( OBLICZANIE CZASU WYSWIETLANIA KURSORA           )
  9 : DELAY 80 0 DO ?TERMINAL IF LEAVE ENDIF LOOP ;
 10 --)

SRC # 203
  0 ( GED - PROGRAM WPROWADZANIA RYSUNKOW NA MONITOR )
  1                                                  )
  2 ( PRZESUNIECIE KURSORA                           )
  3 ( MASK RSA -) MASK MSA                           )
  4 : CRIGHT SWAP DUP 7+ OFF AND SWAP 0= IF DROP 1+ 1 ENDIF SWAP
  5   ROLLDOWN ACT ;
  6 : CLEFT SWAP 2 / DUP 0= IF DROP 1 - ROLLUP 80 ENDIF
  7   SWAP ACT ;
  8 : CDOWN 50 + ROLLDOWN ACT ;
  9 : CUP 50 - ROLLUP ACT ;
 10 : CLEFT-DOWN 50 + ROLLDOWN CLEFT ;
 11 : CLEFT-UP 50 - ROLLUP CLEFT ;
 12 : CRIGHT-DOWN CRIGHT CDOWN ;
 13 : CRIGHT-UP CRIGHT CUP ;
 14 --)
```

Rys. 6a

```
SRC # 204
  0 ( GED - PROGRAM WPROWADZANIA RYSUNKOW NA MONITOR )
  1                                                  )
  2 ( USTAWIENIE TRYBU PRZESUWANIA KURSORA           )
  3 : SET-MOVE  0 SWITCH ! ;
  4 : SET-ERASE 1 SWITCH ! ;
  5 : SET-DRAW  2 SWITCH ! ;
  6
  7 ( POWROT Z NIESKONCZONEJ PETLI                   )
  8 : EXIT    R> R>  DROP DROP ;
  9 --)

SRC # 205
  0 ( GED - PROGRAM WPROWADZANIA RYSUNKOW NA MONITOR )
  1                                                  )
  2 ( KOMPILACJA STRUKTURY DANYCH ZLOZONEJ Z REKORDOW )
  3 ( POSTACI: ZNAK, ADRES PROCEDURY OBSLUGI, ..., 0 )
  4 : KEY-STRUCTURE
  5     [ RIGHT      C, ] CRIGHT       [ LEFT      C, ] CLEFT
  6     [ UP         C, ] CUP          [ DOWN      C, ] CDOWN
  7     [ RIGHT-DOWN C, ] CRIGHT-DOWN  [ RIGHT-UP  C, ] CRIGHT-UP
  8     [ LEFT-UP    C, ] CLEFT-UP     [ LEFT-DOWN C, ] CLEFT-DOWN
  9     [ DRAW       C, ] SET-DRAW     [ ERASE     C, ] SET-ERASE
 10     [ MOVE       C, ] SET-MOVE     [ HOME      C, ] EXIT
 11     [ 0          C, ] ;
 12 --)

SRC # 206
  0 ( GED - PROGRAM WPROWADZANIA RYSUNKOW NA MONITOR )
  1                                                  )
  2 ( INTERPRETACJA ZNAKU ZE SZCZYTU STOSU           )
  3 : INTERPRET-CHAR     ( MASK RSA CHAR -) MASK RSA )
  4   )R ' KEY-STRUCTURE   ( ZNAK NA STOSIE POBRANY  )
  5     BEGIN      ( ADRES STRUKTURY WANYCH NA STOS  )
  6       DUP CO DUP   ( POBRANIE I POBIELENIE ZNAKU )
  7       IF      ( JEZELI 0, TO KONIEC INTERPRETACJI )
  8        R = ( PORONANIE Z ZABRANYM ZNAKIEM STOSU POBRANY )
  9        IF 1+ 0 EXECUTE 0    ( WYKONANIE PROCEDURY )
 10        ELSE 2+ 1 ENDIF ( ZWIEKSZENIE ADRESU O 2 )
 11       ELSE SWAP DROP          ( USUNIECIE ZNAKU )
 12       ENDIF
 13     0= UNTIL  ( KONTYNUACJA PETLI AZ DO ROZPOZ- )
 14   R> DROP ;             ( NANIA ZNAKU LUB KODU 0 )
 15 --)

SRC # 207
  0 ( GED - PROGRAM WPROWADZANIA RYSUNKOW NA MONITOR )
  1                                                  )
  2 ( USTAWIENIE KURSORA NA SRODKU EKRANU            )
  3 ( PO CYKLU MIGNIECIA, SPRAWDZENIE STANU KLAWIA-  )
  4 ( TURY - INTERPRETACJA NACISNIETEGO KLAWISZA     )
  5 ( PROGRAM WYKONUJE SIE W NIESKONCZONEJ PETLI     )
  6 : DOT-EDITOR
  7         0 ENDSCREEN 2 / 28 +
  8         BEGIN
  9          DUP TOPSCREEN + C@ POINT ! RESET-POINT
 10          DELAY SET-POINT DELAY POINT @ OVER
 11          TOPSCREEN + C! ?TERMINAL
 12          IF KEY INTERPRET-CHAR ENDIF
 13         AGAIN DROP DROP ;
 14 ;S
```

Rys. 6b

szczycie stosu znajdują się kolejno: maska bitu (MASK) w bajcie wskazanym przez kursor oraz adres względny bajtu (RSA) wskazanego przez kursor. Słowa ROLLDOWN i ROLLUP w kadrze **201** służą do sprawdzenia, czy adres kursora nie znajduje się poza obszarem pamięci obrazu. Program słowa TAKE-BYTE pobiera z pamięci obrazu bajt wskazany przez kursor i przesyła go na stos wraz z maską bitu. Słowa RESET-POINT i SET-POINT służą do usunięcia lub wstawienia punktu na ekranie monitora w pozycji kursora. Na kadrze dyskowym **202** znajduje się słowo ACT przeznaczone do aktualizacji — zgodnie z bieżącym trybem pracy — stanu punktu wskazanego przez kursor. Słowo DELAY służy do opóźnienia pracy programu. Program tego słowa kończy wykonanie po naciśnięciu klawisza lub po zadanej liczbie pętli. Na kadrze dyskowym o numerze **203** zaprogramowano ruchy kursora w ośmiu kierunkach (słowa CRIGHT, CLEFT, CDOWN, CUP, CLEFT-DOWN, CLEFT-UP, CRIGHT-DOWN, CRIGHT-UP). Programy słów SET-MOVE, SET-ERASE, SET-DRAW z kadru dyskowego **204** ustawiają tryb pracy na przesuwanie, usuwanie lub wstawianie. Słowo EXIT powoduje wyjście z edytora punktowego. Słowo KEY-STRUCTURE z kadru dyskowego **205**, po kompilacji, zawiera strukturę danych, która w polu parametrów ma następującą postać: kod przycisku, adres procedury obsługi (2 bajty), ..., 0. Kompilacji każdego elementu struktury złożonego z kodu przycisku i adresu procedury dokonuje program w postaci:

[wyliczenie-kodu-klawisza C,] procedura-obsługi

Po przejściu interpretera do stanu „interpretacja", następuje przesłanie kodu przycisku na szczyt stosu. Słowo **C** służy do skompilowania bajtu znajdującego się na szczycie stosu. Z kolei interpreter zewnętrzny przechodzi do stanu „kompilacja", w którym następuje skompilowanie adresu procedury obsługi klawisza. Słowo INTERPRET-CHAR z kadru **206** służy do pobrania kodu znaku ze szczytu stosu i wykonania procedury obsługi, jeżeli kod tego znaku znajduje się w polu parametrów słowa KEY-STRUCTURE. Słowo DOT-EDITOR służy do wykonania zadań edytora punktowego.

W przykładzie zilustrowano, w jaki sposób przez kolejne definiowanie słów powstaje program w języku FORTH (będący ostatnim słowem ciągu definicji). Położono przy tym nacisk na czytelność programu, kosztem efektywności jego działania (nie ma on większego znaczenia praktycznego). Starano się tak dobrać nazwy, aby struktura programu była zrozumiała dla czytelnika, który nie chce wgłębiać się w znaczenie poszczególnych słów użytych w definicjach.

* * *

W trzech odcinkach cyklu przedstawiono model FORTHA, sposób jego implementacji oraz opis funkcjonalny języka. Do pełniejszego zrozumienia FORTHA należałoby jednak kontynuować tę tematykę, w sposób bardziej szczegółowy, np. w mikroKLANIE.

FORTH prawdopodobnie nie będzie językiem tak popularnym jak PASCAL czy FORTRAN, gdyż opanowanie go na ogół pochłania więcej czasu niż to jest potrzebne w przypadku innych języków. Trud poznania tego języka jest ceną nabycia wygodnego narzędzia.

Nowe słowa używane w tej części:

C, (b→) — przesłanie bajtu pod następny wolny adres słownika i zwiększenie wskaźnika słów o 1

, (n→) — przesłanie wartości ze szczytu stosu pod następny wolny adres słownika i zwiększenie wskaźnika słów o 2

[— wstrzymanie kompilacji; słowa występujące po **[** są wykonywane, nie kompilowane

] — wznowienie kompilacji

IMMEDIATE — ustawienie bitu poprzedzania w polu nazwy; zaznacza, że program ostatnio zdefiniowanego słowa, wywołanego w stanie „kompilacja", będzie wykonywany; użytkownik może kompilować słowa z ustawionym bitem poprzedzania, umieszczając przed nimi słowo **[COMPILE]**

[COMPILE] — używane w definicji powoduje kompilację słowa, które ma ustawiony bit poprzedzania

<BUILDS — utworzenie pola nazwy i pola łączącego słowa oraz wykonanie programu kompilacji

DOES> — zasygnalizowanie początku definicji programu wykonawczego; po wywołaniu zdefiniowanego słowa w sekwencji **z nnnn, DOES>** zmienia zawartość pola kodu i pierwszego parametru słowa **nnnn** w ten sposób, aby wskazywały program po słowie **DOES>**

EXECUTE (addr→) — wykonanie programu słowa, którego adres pola kodu znajduje się na stosie

;S (addr) — wykonanie operacji RETURN.

FORTH-ASSEMBLER dla mikroprocesora INTEL-8080

System FORTH umożliwia dołączenie procedur napisanych w języku asemblera do słowników systemu. Dokonuje się tego przy użyciu słownika o nazwie ASSEMBLER. Zbiór słów znajdujących się w tym słowniku pozwala na kompilację programu napisanego w języku FORTH-asemblera, tzn. na dołączenie do słownika nazw oraz odpowiadającego im kodu.

W słowniku ASSEMBLER istnieje pełny zbiór słów będących mnemonicznymi instrukcjami języka asemblera. Wywołanie słowa powoduje jego kompilację, tzn. umieszczenie pod adresem wskazanym przez wartość zmiennej DP kodu odpowiadającego tej nazwie i wykonanie programu związanego z dołączeniem wartości argumentów. Po skompilowaniu instrukcji, słownik zostanie powiększony o liczbę bajtów zależną od formatu argumentu.

Słowa odpowiadające mnemonicznym nazwom instrukcji definiuje się przy użyciu specjalnie do tego celu przeznaczonych słów definiujących. Metakompilator dla instrukcji jednobajtowych (NOP, HL itp.) ma następującą postać:

```
: 1MI <BUILDS C, DOES> C ↑ C, ;
0 1MI NOP
```

Słowo definiowane przez 1MI, w fazie kompilacji, powoduje przesłanie kodu definiowanej instrukcji do pierwszego bajtu pola parametrów. W przypadku instrukcji NOP, pole parametrów będzie zawierało kod 0. W fazie wykonania, po wywołaniu instrukcji NOP słowo C ↑ spowoduje umieszczenie kodu instrukcji na szczycie stosu, a słowo C, — dołączenie tego kodu do słownika.

Przy zapisie w programie instrukcji mającej argumenty, nazwa instrukcji musi być poprzedzona argumentami. Przykładowo; instrukcję:

MOV A,B

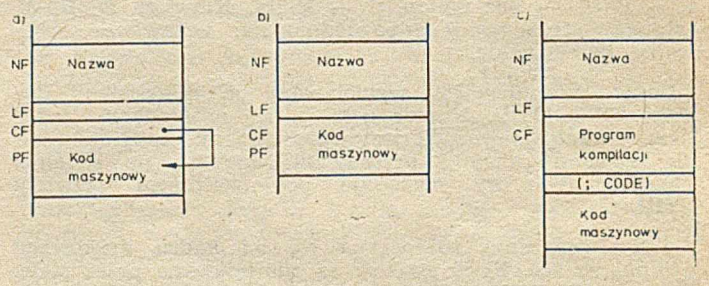
zapisuje się jako:

B A MOV

Definiowanie słowa, którego program jest napisany w języku asemblera, dokonuje się przez słowo CODE. W zdefiniowanym słowie, pole CF zawiera adres pola PF, tzn. pierwszej instrukcji kodu maszynowego. Definicja słowa XXX ma następującą postać:

CODE XXX tekst-programu-w-języku-asemblera C;

Program słowa CODE wykonuje się, gdy interpreter zewnętrzny jest w stanie „interpretacja". Po wywołaniu tego słowa, słownik CONTEXT jest zastępowany przez ASSEMBLER. Po zakończeniu definicji, słowo „C;" przełącza słownik CONTEXT na CURRENT. Kod asemblerowy zwykle kończy instrukcja NEXT JMP, która wykonuje skok do interpretera wewnętrznego.



Programy zdefiniowane przez słowa: a) CODE, b) LABEL, c) :...;CODE...

Asembler, podobnie jak kompilator interpretera zewnętrznego, tłumaczy program w jednym przebiegu. Przeniesienie sterowania programu „w przód" można zaprogramować wykorzystując instrukcje strukturalne (warunkowe lub pętle). Warunek w tych instrukcjach oznacza identyfikator bitu rejestru stanu mikroprocesora. Na oznaczenie warunków używa się następujących nazw:

Bit	Warunek
C	CS
Z	0=
S	0
P	PE

Wystąpienie słowa **NOT** po nazwie warunku powoduje jego zaprzeczenie. Przykładowy program pętli opóźniającej, zrealizowany przy użyciu dwóch rejestrów, ma następującą treść:

```
CODE BEGIN
     BEGIN
     C DCR
     0= UNTIL
     B DCR
     0= UNTIL C;
```

Słowa będące nazwami warunków definiuje się przy użyciu słowa **CONSTANT**. Stałą zapisywaną w pole parametrów jest kod instrukcji skoku warunkowego. Po wywołaniu takiego słowa kod instrukcji zostanie przesłany na szczyt stosu. Wywołanie instrukcji warunkowej spowoduje umieszczenie kodu skoku warunkowego w słowniku oraz rezerwację miejsca (IF i WHILE) lub wypełnienie go argumentem instrukcji (UNTIL).

Słowo definiujące **LABEL** ułatwia posługiwanie się nazwami symbolicznymi przy pisaniu instrukcji sterujących i ma następujący format:

```
LABEL YYY kod-programu-w-języku asemblera C;
```

gdzie **YYY** jest definiowaną nazwą. Adres zawarty w polu słowa zdefiniowanego przez **LABEL** wskazuje część wykonawczą słowa definiującego **VARIABLE**. W ten sposób, po wykonaniu słowa zdefiniowanego przez **LABEL**, na szczycie stosu zostanie umieszczony adres początku programu napisanego po tym słowie. Przykładowym zastosowaniem słowa **LABEL** jest definiowanie podprogramów. Podprogram mnożenia liczby 16-bitowej przez 10 (HL = 10*LICZBA) ma następującą postać:

```
LABEL          NUMBER
     H DAD L E MOV H D MOV
     H DAD H DAD D DAD RET C;
```

```
SRC # 99
  0 ( *** FIG-FORTH 8080 ASSEMBLER *** )
  1 HEX VOCABULARY ASSEMBLER IMMEDIATE
  2 ' ASSEMBLER CFA ' ;CODE 8 + !
  3 : CODE ?EXEC CREATE [COMPILE] ASSEMBLER !CSP ; IMMEDIATE
  4 : C; CURRENT @ CONTEXT ! ?EXEC ?CSP SMUDGE ; IMMEDIATE
  5 : LABEL ?EXEC 0 VARIABLE SMUDGE -2 ALLOT [COMPILE] ASSEMBLER
  6       !CSP ; IMMEDIATE
  7 : 8* DUP + DUP + DUP + ; ASSEMBLER DEFINITIONS
  8 4 CONSTANT H     5 CONSTANT L     7 CONSTANT A     6 CONSTANT PSW
  9 2 CONSTANT D     3 CONSTANT E     0 CONSTANT B     1 CONSTANT C
 10 6 CONSTANT M     6 CONSTANT SP                   145 CONSTANT NEXT
 11 : 1MI <BUILDS  C,  DOES>  C@  C, ;
 12 : 2MI <BUILDS  C,  DOES>  C@ + C, ;
 13 : 3MI <BUILDS  C,  DOES>  C@ SWAP 8* + C, ;
 14 : 4MI <BUILDS  C,  DOES>  C@ C, C, ;
 15 : 5MI <BUILDS  C,  DOES>  C@ C, , ;  -->

SRC # 100
  0 00 1MI NOP     76 1MI HLT     F3 1MI DI      FB 1MI EI
  1 07 1MI RLC     0F 1MI RC      17 1MI RAL     1F 1MI RAR
  2 E9 1MI PCHL    F9 1MI SPHL    E3 1MI XTHL    EB 1MI XCHG
  3 27 1MI DAA     2F 1MI CMA     37 1MI STC     3F 1MI CMC
  4 80 2MI ADD     88 2MI ADC     90 2MI SUB     98 2MI SBB
  5 A0 2MI ANA     A8 2MI XRA     B0 2MI ORA     B8 2MI CMP
  6 09 3MI DAD     C1 3MI POP     C5 3MI PUSH    02 3MI STAX
  7 0A 3MI LDAX    04 3MI INR     05 3MI DCR     03 3MI INX
  8 0B 3MI DCX     C7 3MI RST     D3 4MI OUT     DB 4MI IN
  9 C6 4MI ADI     CE 4MI ACI     D6 4MI SUI     DE 4MI SBI
 10 E6 4MI ANI     EE 4MI XRI     F6 4MI ORI     FF 4MI CPI
 11 22 5MI SHLD    2A 5MI LHLD    32 5MI STA     3A 5MI LDA
 12 C4 5MI CNZ     CC 5MI CZ      D4 5MI CNC     DC 5MI CC
 13 E4 5MI CPO     EC 5MI CPE     F4 5MI CP      FC 5MI CM
 14 CD 5MI CALL
 15 -->

SRC # 101
  0 C0 1MI RNZ     C8 1MI RZ      D0 1MI RNC     D8 1MI RC
  1 E0 1MI RPO     E8 1MI RPE     F0 1MI RP      F8 1MI RM
  2 C9 1MI RET     C3 5MI JMP     C2 CONSTANT 0=  D2 CONSTANT CS
  3 E2 CONSTANT PE  F2 CONSTANT 0<    : NOT 8 + ;
  4 : MOV 8* 40 + + C, ;  : MVI 8* 6 + C, C, ;  : LXI 8* 1+ C, , ;
  5 : ENDIF 2 ?PAIRS HERE SWAP ! ;    : THEN [COMPILE] ENDIF ;
  6 : IF C, HERE 0 , 2 ;
  7 : ELSE 2 ?PAIRS C3 IF ROT SWAP ENDIF 2 ;
  8 : BEGIN HERE 1 ;
  9 : UNTIL SWAP 1 ?PAIRS C, , ;  : AGAIN 1 ?PAIRS C3 C, , ;
 10 : WHILE IF 2+ ;
 11 : REPEAT >R >R AGAIN R> R> 2 - ENDIF ;
 12 FORTH DEFINITIONS DECIMAL ;S
 13
 14
 15
```

a jego wywołanie:

```
10 NUMBER CALL
```

Do standardowego słownika **FORTHA** należy także słowo **;CODE**, które powoduje zmianę słownika **CONTEXT** na **ASSEMBLER**. W czasie kompilacji słownika **ASSEMBLER** treść kodu słowa **;CODE** musi być uzupełniona o adres słowa **ASSEMBLER**. Postać kodu zdefiniowanego przez słowa **CODE**, **LABEL** i **;CODE** przedstawiono na rysunku.

Zasady prenumeraty

Zamówienia i przedpłaty na prenumeratę **INFORMATYKI** przyjmuje Zakład Kolportażu Wydawnictwa **NOT SIGMA**. Adres pocztowy: Wydawnictwo **NOT SIGMA** — Zakład Kolportażu, 00-950 Warszawa, skr. poczt. 1004. Konto bankowe: 1036-7490-139-11, III O/M NBP w Warszawie.

JEDNOSTKI GOSPODARKI USPOŁECZNIONEJ, INSTYTUCJE I ORGANIZACJE przesyłają zamówienia (w 1 egz.) zawierające: tytuł czasopisma, liczbę zamawianych egzemplarzy, okres prenumeraty i pełny adres zamawiającego z kodem pocztowym, oddział i nazwę banku z numerem konta bankowego zamawiającego oraz (ewentualnie) adres odbiorców, którzy na zlecenie i koszt zamawiającego mają egzemplarze otrzymywać.

Warunkiem realizacji zamówienia jest równoczesne dokonanie odpowiedniej wpłaty na ww. konto Wydawnictwa **NOT SIGMA**.

Za prenumeratę nie wystawiane są rachunki i nie potwierdzane salda. Prenumeratorzy zbiorowi proszeni są o podawanie na dowodach wpłat (przelewach) znaku kancelaryjnego zamówienia, którego dotyczy wpłata.

Dopisując na zamówieniu **PRENUMERATA STAŁA**, zamawiający (tylko prenumeratorzy zbiorowi!) nie będą musieli corocznie ponawiać zamówienia, a jedynie dokonywać przedpłaty według aktualnie obowiązujących cen. Wydawnictwo przekazywać będzie co roku potwierdzenie kontynuacji prenumeraty.

PRENUMERATORZY INDYWIDUALNI dokonują wpłaty przekazem NBP na ww. konto, pod powyższym adresem, podając na odwrocie odcinka dla adresata-posiadacza rachunku: tytuł czasopisma, liczbę zamawianych egzemplarzy oraz okres prenumeraty.

Do **PRENUMERATY ULGOWEJ** upoważnieni są członkowie stowarzyszeń naukowo-technicznych **NOT**, studenci, uczniowe szkół zawodowych. Warunkiem jej uzyskania jest poświadczenie blankietu przekazu NBP dla nabywcy indywidualnego (na odcinku dla adresata) przez właściwe stowarzyszenie **NOT**, wyższą uczelnię lub szkołę zawodową.

Zamówienia i wpłaty przyjmowane są na okresy kwartalne, półroczne i roczne w terminach:

- do 1 listopada — na I kwartał, I półrocze i cały rok następny
- do 28 lutego — na II, III i IV kwartał
- do 31 maja — na IV kwartał i II półrocze
- do 31 sierpnia — na IV kwartał.

Uwaga: Przy podawaniu kodu pocztowego i numeru konta bankowego obowiązuje bardzo czytelne pismo. Prenumerata nie wymaga specjalnego przekazu z czerwonym paskiem; wystarczy zwykły przekaz bankowy.

Prenumerata normalna: kwartalna — 225 zł, półroczna — 450 zł, roczna — 900 zł. Prenumerata ulgowa: kwartalna — 150 zł, półroczna — 300 zł, roczna — 600 zł. Prenumerata ze zleceniem wysyłki za granicę jest dwukrotnie droższa.

Dodatkowych informacji o prenumeracie udziela: Zakład Kolportażu, tel. 40-00-21 w. 293, 299 oraz 40-35-89. Egzemplarze archiwalne można nabywać w Klubie Prasy i Informacji Technicznej w Warszawie, ul. Mazowiecka 12, tel. 27-43-65. Zamówienia na egzemplarze archiwalne należy kierować pod adresem Zakładu Kolportażu.

CEZARY ZIELIŃSKI
Instytut Automatyki
Politechnika Warszawska

Roboty (3)

Proste języki programowania

Rozwój produkcji oraz badań nad robotami przemysłowymi stymulowany jest mnogością ich zastosowań. Większość tych automatów programowana jest przez wtykanie kołków w bębny lub matryce diodowe oraz odpowiednie ustawienie potencjometrów położenia zadanego lub wyłączników krańcowych. Kolejnym etapem rozwoju metod programowania jest „uczenie" trajektorii ruchu. Operator przeprowadza ramię robota poprzez pożądane położenia, które zapamiętywane są w postaci cyfrowej w pamięci układu sterującego, tak aby następnie mogły być wielokrotnie i samodzielnie odtwarzane.

Metody te są skuteczne jedynie w prostych przypadkach, tzn. gdy praca przeznaczona do wykonania daje się zapisać jako liniowa sekwencja czynności. Dlatego też zwiększono możliwości programatorów klawiszowych, wprowadzając instrukcje warunkowych skoków programowych oraz rozkazy synchronizacji z procesem (głównie komendy oczekiwania na pewne zdarzenie zewnętrzne lub na upłynięcie określonego czasu). Nadal jednak nie było możliwości użycia podprogramów, czyli zwartego zakodowania identycznej sekwencji czynności powtarzanej w różnych punktach przestrzeni roboczej. Zapisanie algorytmów przetwarzania informacji, uzyskanych z receptorów (czujników), też było poza zasięgiem wyżej opisanych metod programowania.

Ubogi repertuar oferowanych możliwości nie był jedyną niedogodnością dotychczasowych metod. Najpoważniejszym problemem było to, że w trakcie programowania robót nie wykonywał żadnych użytecznych czynności, a więc nie zarabiał na siebie. Nawet drobna — wydawałoby się — poprawka w programie, związana ze zmianą jednego położenia manipulatora, wiązała się z unieruchomieniem linii produkcyjnej i żmudnym przeprogramowaniem robota. Jedynym wyjściem było przygotowanie programu na urządzeniu nie związanym bezpośrednio z produkcją — najlepiej na komputerze. Zmiana asortymentu produkcji wiązałaby się z przygotowaniem programów poza robotem a wstrzymanie produkcji następowałoby tylko na czas ładowania ich do pamięci układów sterujących. W ten sposób zrodziła się potrzeba symbolicznego zapisu czynności manipulatora, a więc opracowania języków programowania robotów.

Istnieją dwa równoległe podejścia do tego zagadnienia. Pierwsze to wykorzystanie uniwersalnego języka programowania typu FORTRAN, PASCAL czy ADA oraz dodanie specjalizowanych procedur sterujących częścią mechaniczną. Drugie, częściej stosowane, to opracowanie języka specjalizowanego, przeznaczonego do sterowania manipulatorem oraz przetwarzania informacji pochodzących od „zmysłów" robota.

Zaletą pierwszego podejścia jest wykorzystanie możliwości języków uniwersalnych oraz mniejszy koszt realizacji przedsięwzięcia. Zaleta drugiego jest lepsze dostosowanie do potrzeb użytkownika oraz brak zbędnych konstrukcji językowych, nie wykorzystywanych w programowaniu robota, a wymagających większej wiedzy programisty.

Specjalizowane języki programowania robotów można podzielić w zależności od poziomu ogólności opisu zadania.

Na poziomie najniższym znajdują się języki zorientowane na przemieszczanie poszczególnych fragmentów łańcucha kinematycznego[1]) **robota** (ang. joint level). Języki te zmuszają programistę do wyrażenia zadania jako sekwencji instrukcji sterujących poszczególnymi napędami. Wymaga to dość dużej wprawy, gdyż niezbędne jest wyczucie ruchu końca łańcucha kinematycznego, który jest efektem złożenia instrukcji przemieszczających poszczególne człony ramienia.

Języki kolejnego poziomu uwalniają użytkowników od tej niedogodności. Są one zorientowane na przemieszczanie końcówki manipulatora w przestrzeni kartezjańskiej (ang. manipulator level). W tym przypadku specyfikacja zadania polega na podaniu punktów przestrzeni kartezjańskiej, przez które należy kolejno przeprowadzać koniec ramienia robota. Aczkolwiek jest to duże udogodnienie, w porównaniu z językami poprzedniej grupy, to nadal programista bardziej zajmuje się opisem ruchów manipulatora niż opisem czynności, które powinny być wykonane.

Języki zorientowane na przemieszczenia obiektów (ang. object level) **zakładają, że układ sterujący (lub kompilator języka) ma już pewne informacje o obiektach znajdujących się w otoczeniu robota.** Użytkownik podaje więc w programie, które z przedmiotów i w jaki sposób mają być przemieszczone, aby zadanie zostało wykonane. System sterujący robotem spowoduje — na podstawie wiedzy o obiektach i relacjach zachodzących między nimi — odpowiednie przemieszczenie łańcucha kinematycznego.

Zamiast wyszczególniania wszystkich czynności wykonywanych na obiektach należałoby jednak dążyć do ograniczenia problemu jedynie do ogólnego opisu zadania, umożliwiającego automatyczne wygenerowanie odpowiedniego planu działania, a następnie jego realizację. Takie aspiracje mają języki zorientowane na zadanie (ang. task level). Są to języki sztucznej inteligencji bądź języki bezpośrednio z nich się wywodzące. Mimo że prace nad nimi trwają najdłużej to są one najmniej rozwinięte.

PRZYKŁADY JĘZYKÓW

Ponad dwudziestoletni rozwój robotyki doprowadził do powstania wielu języków. Podane poniżej przekłady pozwalają lepiej scharakteryzować istotne cechy poszczególnych grup tych języków. Przegląd rozpoczniemy od języków najprostszych.

*

Reprezentantem grupy języków zorientowanych na przemieszczanie poszczególnych członów ramienia jest rozwiązanie zaproponowane przez prof. Spura z Politechniki Berlińskiej. Instrukcje tego języka zostały podzielone na trzy zbiory.

W pierwszym znajdują się instrukcje i dyrektywy powodujące przemieszczanie napędów. Dyrektywy decydują, czy w następujących po nich instrukcjach ruchu współrzędne mają być traktowane jako bezwzględne (ABS) czy względne (REL). Ponadto określają prędkość, z jaką ruch ma być realizowany:

F < liczba wyrażająca prędkość ruchu>

Same instrukcje ruchu mają postać:

<jednoliterowy identyfikator stopnia swobody> <liczba>

Przykładowo — **C10** oznacza, że fragment ramienia o nazwie **C** należy przemieścić o dziesięć jednostek lub do położenia **10**, w zależności od rodzaju dyrektywy (REL, ABS) umieszczonej przed tą instrukcją.

[1]) Łańcuchem kinematycznym nazywa się spójny zespół członów połączonych w pary kinematyczne (dwa człony połączone ruchowo)

Drugi zbiór zawiera instrukcje wejścia-wyjścia, a więc zbierające informacje od urządzeń produkcyjnych bądź własnych czujników oraz wysyłające sygnały sterujące współpracującymi maszynami.

W skład trzeciego wchodzą instrukcje sterujące wykonaniem programu. Wzorzec tych rozkazów został zaczerpnięty z FORTRANU, zatem programista ma do dyspozycji:
— skok bezwarunkowy
— skok do podprogramu
— warunkową realizację instrukcji
— możliwość założenia pętli programowej (pętla DO).

*

Przedstawicielem tej grupy jest język SIGLA, opracowany przez firmę OLIVETTI do sterowania robotem SIGMA, robotem zwanym kartezjańskim [2]).

Język SIGLA jest dostarczany użytkownikowi z zestawem instrukcji dostosowanych do jego potrzeb. W ten sposób obszar pamięci przeznaczony na przechowywanie interpretera został zminimalizowany. W zestawie tym mogą znajdować się następujące rozkazy: wstępne pozycjonowanie napędów, ruch ramienia, testowanie wejścia binarnego oraz ustawianie wyjścia binarnego, wstrzymanie wykonania programu, skok warunkowy oraz bezwarunkowy. Ponadto mogą być jeszcze dodane tak złożone instrukcje, jak np. ruch ramienia połączony z wywieraniem odpowiedniej siły nacisku.

Cały system został tak zaprojektowany, aby w przypadku sterowania wieloma robotami i urządzeniami współpracującymi przeznaczone dla nich instrukcje były umieszczane w oddzielnych plikach. Użytkownik może żądać równoległego wykonania instrukcji z wielu plików. Na przykład — przekazanie systemowi sterującemu komendy w postaci:

AU/n1, n2,.../m1, m2,.../.../p1, p2,...

spowoduje równoległe wykonanie zawartości plików **n1, m1,...,p1**, a następnie **n2, m2,...,p2** itd. Synchronizacja pomiędzy poszczególnymi zadaniami następuje poprzez wspólne obszary danych. W ten sposób programista może żądać współbieżnego wykonania aż 64 zadań.

*

Kolejny język opracowany przez firmę OLIVETTI dla robotów SUPERSIGMA został bardziej upodobniony do powszechnie używanych języków programowania. Nazwano go MAL (Multipurpose Assembly Language). Zrezygnowano w nim z asemblerowej struktury rozkazów języka SIGLA, na korzyść syntaktyki zaczerpniętej z BASICA. Wprowadzono tu wiele odmian instrukcji sterujących ruchem manipulatora. Przykładowo:

MOVE XL = 10, YL = 20

powoduje przemieszczenie lewego ramienia robota kartezjańskiego SUPERSIGMA do położenia **X = 10, Y = 20**, natomiast **INCR XL = 10, YL = 20** przemieszcza ramię o 10 jednostek wzdłuż osi **X** oraz 20 wzdłuż **Y**. Dodanie litery **W** po kodzie operacji powoduje, że powyższe instrukcje wstrzymują dalsze wykonanie programu do chwili zakończenia ruchu.

Instrukcje sterujące wykonaniem programu są typowe. W ich skład wchodzą: warunkowe wykonanie instrukcji (IF), skok bezwarunkowy (GOTO), skok (CALL) do oraz powrót z podprogramu (RETURN) i pętla programowa (NEXT).

Ponadto można w programie określić wiele zadań (TASK). Będą one wykonane równolegle. Synchronizację między nimi zapewniają instrukcje podstawienia (SET) oraz oczekiwania na spełnienie warunku (WAIT).

Język zawiera także instrukcje aktywujące (ACT) i deaktywujące (DEACT) urządzenia współpracujące oraz instrukcje wejścia-wyjścia (READ-PRINT).

Ze względu na specyficzną konstrukcję tego robota, sterowanie poszczególnymi napędami jest równoważne przemieszczeniom we współrzędnych kartezjańskich. Jednakże ze względu na brak wbudowanego w system przelicznika współrzędnych, język został zakwalifikowany do grupy zorientowanej na przemieszczanie członów łańcucha kinematycznego. Uwaga ta dotyczy również języka SIGLA.

*

Jednym z pierwszych języków przeznaczonych do sterowania manipulatorami był WAVE. Został on opracowany w Stanford Artificial Intelligence Laboratory w latach 1970—1975. Jest on w pełni zorientowany na przemieszczanie końca manipulatora w przestrzeni kartezjańskiej. Cały system sterujący został zaimplementowany na komputerach PDP-10 oraz PDP-6 i jest przeznaczony dla manipulatora o sześciu stopniach swobody.

Podstawowym typem danych zdefiniowanym w tym języku jest układ współrzędnych. Operuje się jedynie na prawoskrętnych układach prostokątnych **XYZ**. Jeden z nich jest wyróżniony jako bazowy, bezwzględny układ odniesienia. Zazwyczaj jest on związany z podstawą robota. Z każdym obiektem oraz każdym charakterystycznym punktem przestrzeni roboczej programista może związać lokalny układ współrzędnych. Położenie (przesunięcie oraz orientację przestrzenną) nowego układu względem układu odniesienia określa się instrukcją TRANS (ang. transform). Na przykład instrukcja:

TRANS UK 25, 15, 10, 0, 90, 0

spowoduje nadanie wartości zmiennej typu: układ współrzędnych. Dla tej instrukcji przesunięcia będą miały wartość **X = 25, Y = 15, Z = 10**, natomiast orientacja będzie określona jako obrót o 0° wokół osi **Z**, 90° wokół osi **X** oraz 0° wokół osi **Y**. Dla robota instrukcja ta stanowi informację o tym gdzie znajduje się obiekt oraz w jaki sposób należy go chwycić, gdy będzie to konieczne.

W języku tym wprowadzono także inne typy danych. Są to wektory oraz liczniki pętli. Instrukcja:

VECT <nazwa> <współrzędna x>, <współrzędna y>, <współrzędna z>

nadaje wartości współrzędnym wektora, którego identyfikator znajduje się w jej treści. Natomiast rozkaz:

ASSIGN <nazwa>, <wartość początkowa>

inicjuje licznik pętli.

Prócz instrukcji opisujących punkty charakterystyczne środowiska, język zawiera także rozkazy powodujące zmianę stanu manipulatora. Są nimi: OPEN, CLOSE, CENTER oraz MOVE i CHANGE.

OPEN <liczba> powoduje rozwarcie palców chwytaka na zadaną odległość. Gdy rozwarcie palców staje się większe od wyspecyfikowanego, to chwytak jest zamykany.

Instrukcja **CLOSE <liczba>** powoduje zamykanie chwytaka aż do napotkania oporu. Jeśli odległość między palcami będzie mniejsza od podanej w instrukcji, w chwili napotkania siły przeciwdziałającej ich zbliżaniu, to system wyda komunikat o błędzie.

Najprostszym sposobem implementacji tego rozkazu jest skorzystanie z informacji otrzymywanej z dwóch czujników dotyku zamontowanych na palcach.

Najbardziej wyrafinowaną instrukcją dotyczącą chwytania obiektów jest komenda **CENTER <siła>**. Powoduje ona ściśnięcie obiektu z zadaną siłą, ale bez jego poruszenia. Układ sterujący zamyka chwytak śledząc stan czujników dotyku. W momencie gdy jeden z nich da sygnał informujący o zetknięciu któregokolwiek z palców z obiektem, rozpoczyna się przemieszczanie całego ramienia z taką prędkością, aby palec przylegający do obiektu pozostał nieruchomy w przestrzeni. W chwili zetknięcia się drugiego z palców z obiektem ruch ramienia jest przerywany, natomiast zamykanie chwytaka jest kontynuowane do osiągnięcia odpowiedniej siły nacisku. W ten sposób można uchwycić obiekt bez przesuwania go w płaszczyźnie chwytu.

Podstawowym rozkazem powodującym ruch ramienia jest:

[2]) Robot kartezjański — manipulator, którego fragmenty łańcucha kinematycznego przemieszczają się wzdłuż osi kartezjańskiego układu odniesienia

MOVE <nazwa układu współrzędnych>

Powoduje on przemieszczenie końcówki manipulatora z położenia aktualnego do wyspecyfikowanego. Ruch odbywa się poprzez dwa punkty pośrednie. Ich położenia wyliczane są automatycznie przez system. Pierwszy punkt znajduje się nad pozycją początkową, drugi natomiast nad celem. Punkty pośrednie zostały wprowadzone po to, by system wyznaczając trajektorię ruchu nie przeprowadził jej poprzez ławę roboczą. Niemniej programista sam musi się zatroszczyć, by robot nie uległ kolizji z innymi obiektami w jego otoczeniu.

Inną instrukcją napędzającą ramię jest CHANGE. Ma ona pięć argumentów oddzielonych przecinkami: nazwa wektora wyznaczającego kierunek przesunięcia, liczba określająca zakres ruchu, nazwa wektora stanowiącego oś obrotu chwytaka, kąt obrotu, czas przeznaczony na wykonanie przemieszczenia (jeśli podany czas będzie równy zero, to system wyliczy czas ruchu opierając się na wewnętrznym modelu dynamiki robota).

Ponadto wprowadzono dyrektywę STOP. Jej argumentami są dwa wektory. Pierwszy określa siłę, a drugi moment siły, po napotkaniu których należy zatrzymać ramię. Przykładowo:

```
VECT V 0,0,-20
STOP V , NIL
MOVE UK
```

spowoduje ruch manipulatora w kierunku punktu **UK**, przy czym po napotkaniu oporu o podanej sile i zerowym momencie siły, ramię zostanie zatrzymane.

Ponieważ w języku istnieją instrukcje, które w przypadku nienormalnego zakończenia wydają komunikat o błędzie (np. CLOSE) — umożliwiono dodatkowe korzystanie z tej informacji. SKIPE oraz SKIPN są rozkazami powodującym opuszczenie lub wykonanie kolejnej instrukcji, w zależności od pojawienia się lub braku komunikatu o błędzie. Ponadto wprowadzono skok bezwarunkowy **JUMP <etykieta>.**

Przewidziano również dyrektywy ułatwiające wkładanie wałków w otwory. W momencie gdy bolec zacznie zagłębiać się w otwór, układ „traci" stopnie swobody. Faktycznie pozostają tylko dwa z nich — przesunięcie do wnętrza otworu oraz obrót wokół jego osi. Programista może wskazać te stopnie swobody dyrektywami FREE oraz SPIN. Argumentami pierwszej są osie, wzdłuż których można dokonywać przemieszczeń, natomiast drugiej — osie, wokół których można obracać obiekt. Ponadto istnieje dyrektywa wprowadzająca nadgarstek w drgania. Jest nią **WOBBLE <amplituda drgań>.** Stosuje się ją w przypadku przewidywanego zakleszczenia bolca w otworze.

Jak widać, język WAVE zawiera zarówno instrukcje uniwersalne, możliwe do zastosowania przy wykonywaniu każdego zadania, jak i specjalizowane, przeznaczone do realizacji konkretnych celów. Taka niejednorodność spowodowana jest chęcią ułatwienia oprogramowania zadań często spotykanych w praktyce, przy jednoczesnym zachowaniu uniwersalizmu, umożliwiającego — niekiedy w dość skomplikowany sposób — realizację przedsięwzięć rzadko spotykanych lub nawet trudnych do przewidzenia.

*

Przykładem języka zawierającego zarówno instrukcje uniwersalne, jak i specjalizowane jest także język opracowany przez L. Nemes'a z Węgierskiej Akademii Nauk. Ma on obok rozkazów powodujących przesunięcia poszczególnych członów ramienia robota również takie, które dokonują przemieszczeń końcówki manipulatora w przestrzeni kartezjańskiej. Oprócz bogatego repertuaru instrukcji „ruchowych", język stwarza możliwość definiowania układów współrzędnych związanych z różnymi miejscami w przestrzeni roboczej. Dzięki temu można odwoływać się do dowolnej pozycji w przestrzeni przez podanie nazwy lokalnego układu współrzednych oraz współrzędnych punktu, wyrażonych w tym układzie.

Rozkazy tego języka można zaszeregować do jednej z czterech kategorii w zależności od rodzaju parametrów.

W grupie pierwszej znajdują się instrukcje bezparametrowe; są nimi: GRIP, DROP oraz SEIZE. GRIP powoduje zamykanie, a DROP — otwieranie chwytaka. SEIZE zamyka chwytak, tak modyfikując położenie nadgarstka ramienia, aby nie przesunąć chwytanego obiektu.

Do drugiej kategorii zaliczane są instrukcje sterujące poszczególnymi napędami ramienia (LIFT, TURN, TILT, ROTATE). Parametrami ich są: położenie lub przyrost. W przypadku konieczności jednoczesnego poruszania kilku członów stosuje się znak **I.** Ilustruje to następujący przykład:

```
TURN   10  I
LIFT   30  I
ROTATE  5
```

Sekwencja ta spowoduje równoczesny obrót kolumny, podnoszenie ramienia i obrót nadgarstka. W przypadku braku znaku **I**, wszystkie trzy instrukcje zostałyby wykonane sekwencyjnie.

W skład kategorii trzeciej wchodzą rozkazy bardziej złożone. Przykładowo instrukcja **MOVE (R1, P1)** powoduje przemieszczenie chwytaka do punktu **P1** w układzie odniesienia **R1**, natomiast **PATH (l, n)** przesuwa chwytak wzdłuż zadanej trajektorii z zachowaniem jego orientacji.

Grupa czwarta zawiera rozkazy specjalizowane. Między innymi: **TRANSFER (<nazwa obiektu>; P, R)** — powodują pobranie, z podajnika, obiektu o zadanej nazwie oraz przeniesienie go do punktu **P** układu współrzędnych **R.** Innym ciekawym rozkazem jest **FETCH (<nazwa obiektu>; P, R),** który pobiera wskazany obiekt z miejsca gdzie został uprzednio pozostawiony i przenosi go do punktu **P** we współrzędnych **R.**

* * *

W następnym numerze INFORMATYKI opiszę języki wysokiego poziomu przeznaczone do programowania robotów. Będzie to ostatnia część naszego cyklu.

LITERATURA

[1] Gini G., Gini M., Gini R., Giuse D.: Introducing software systems in industrial robots. Proceedings of the 9th International Symposium on Industrial Robots, March 1979

[2] Gini G., Gini M.: ADA: A language for robot programming? Computers in Industry, Vol. 3, No. 4, 1982

[3] Nemes L.: A simple robot Language for microcomputer based robot controllers. Proceedings of the 7th International Symposium on Industrial Robots, October 1977

[4] Paul R.: WAVE. A model based language for manipulator control. The Industrial Robot. March 1977

[5] Salmon M.: SIGLA — The Olivetti Sigma Robot Programming Language. Proceedings of the 8th International Symposium on Industrial Robots, 1978

[6] Spur G., Auer B. H., Sinning H.: CNC — Control for industrial robots with a modular programming language. Proceedings of the 7th International Symposium on Industrial Robots, October 1977.

KSIĄŻKI NADESŁANE

WYDAWNICTWA NAUKOWO-TECHNICZNE

Błażewicz J., Cellary W., Słowiński R., Węglarz J.: Badania operacyjne dla informatyków. 338 str., nakład 5000 egz., cena 200 zł, Warszawa, 1983.

Książka przedstawia wybrane pojęcia i metody badań operacyjnych (m.in. złożoność obliczeniową problemów decyzyjnych, przepływy w sieciach, programowanie liniowe i całkowitoliczbowe, programowanie dynamiczne, systemy masowej obsługi). Opisane są także zagadnienia informatyczne, w których badania operacyjne znalazły szerokie zastosowanie (np. probabilistyczne problemy szeregowania zadań i rozdziału zasobów, optymalizacja zarządzania pamięcią wirtualną, dobór liczb kopii segmentów pamięci programów w wieloprocesorowym systemie sterującym). Selekcja materiału dokonana była przede wszystkim z punktu widzenia potrzeb projektantów systemów operacyjnych.

JERZY DAŃDA
ZBIGNIEW POZNAŃSKI
Instytut Technologii Elektronowej NPCP
Warszawa

Programowanie w języku PL/M (2)

W drugiej części artykułu omawiamy bardziej złożone konstrukcje języka, wyjaśniając ich użycie w przykładach — analogicznie do konwencji przyjętej w części pierwszej.

PRZYKŁAD 3 — SYMULACJA RUCHU OBIEKTÓW

Symulowany jest ruch obiektów we współrzędnych biegunowych. W każdym takcie, do określonego miejsca pamięci mikrokomputera, przesyłana jest informacja o azymucie losowo wybranego obiektu.

```
INFORMATION$SYSTEM: DO;

DECLARE DCL LITERALLY 'DECLARE', TRUE LITERALLY '0FFH';
DCL (I,K) BYTE,(BAZA$LOSOWANIA,ADRES$W$HL) ADDRESS;
DCL (LICZBA$LOS BASED BAZA$LOSOWANIA, ZAWARTOSC$PAMIECI BASED
ADRES$W$HL) BYTE;
DCL PARA$REJ$HL STRUCTURE(H$REJ BYTE, L$REJ BYTE);
DCL PARA$REJ$BC STRUCTURE(B$REJ BYTE, C$REJ BYTE);
DCL OBIEKT(5) STRUCTURE(R BYTE, FI ADDRESS,DR BYTE, DFI ADDRESS)
                        INITIAL(5,1,1,2,
                                9,3,4,1,
                                11,25,2,5,
                                24,15,3,3,
                                19,27,4,5);
DCL TABLICA$ADRESOW(10) BYTE DATA(20H,20H,20H,22H,20H,24H,20H,
26H,20H,29H);

BAZA$LOSOWANIA = 1250H;
DO WHILE TRUE;
  DO I = 0 TO 4;
    OBIEKT(I).R = OBIEKT(I).R + OBIEKT(I).DR;
    OBIEKT(I).FI = OBIEKT(I).FI + OBIEKT(I).DFI;
  END;
  K = LICZBA$LOS MOD 5;
  BAZA$LOSOWANIA = BAZA$LOSOWANIA + 1;
  PARA$REJ$HL.H$REJ = TABLICA$ADRESOW(2*K);
  PARA$REJ$HL.L$REJ = TABLICA$ADRESOW(2*K+1);

  PARA$REJ$BC.B$REJ = HIGH(OBIEKT(K).FI);
  PARA$REJ$BC.C$REJ = LOW(OBIEKT(K).FI);

  ADRES$W$HL = SHL(PARA$REJ$HL.H$REJ,8) + PARA$REJ$HL.L$REJ;
  ZAWARTOSC$PAMIECI = PARA$REJ$BC.B$REJ;
  ADRES$W$HL = ADRES$W$HL + 1;
  ZAWARTOSC$PAMIECI = PARA$REJ$BC.C$REJ;
END;

END INFORMATION$SYSTEM;
```

Wydruk 1. Przykład modułu 3

Program realizujący odpowiedni algorytm stanowi moduł główny (wydruk 1). Zadeklarowano w nim dwa synonimy DCL i TRUE dla słów DECLARE i OFFH. Następne deklaracje obejmują zmienne BAZA$LOSOWANIA i ADRES$W$HL. Pierwsza z nich stanowi adres, od którego odczytywane będą liczby losowe, druga zaś określa adres znajdujący się w parze rejestrów HL. Obie te zmienne stanowią bazy dla zmiennych bazowanych LICZBA$LOS i ZAWARTOSC$PAMIECI (typu BYTE), które umożliwiają dostęp do komórek pamięci o adresach określonych przez bazy. W kolejnych deklaracjach zdefiniowano struktury PARAREJHL i PARAREJBC, reprezentujące pary rejestrów HL i BC.

Do opisu obiektów, których ruch jest symulowany, wykorzystano tablicę struktur o nazwie OBIEKT. Atrybutami każdego elementu tej tablicy są współrzędne biegunowe położenia **R** i **FI** oraz przyrosty tych współrzędnych, odpowiednio — **DR** i **DFI.** Wykorzystano także możliwość inicjowania wymienionych atrybutów przy użyciu deklaracji INITIAL. Ponieważ zadeklarowano pięcioelementową tablicę struktur, a każdy z elementów tej tablicy ma cztery atrybuty, podano dwadzieścia wartości początkowych.

Ostatnią deklarację stanowi dziesięcioelementowa tablica o nazwie TABLICA$ADRESOW i elementach typu BYTE. Zawarto w niej adresy, pod które należy przesyłać wartości azymutów kolejnych obiektów, przy czym **k**-temu obiektowi przyporządkowano adres określony przez dwa elementy tablicy **2k** i **2k+1.** Ponieważ adresy są stałe, w celu nadania elementom tablicy TABLICA$ADRESOW wartości początkowych wykorzystano deklarację DATA. Przeciwnie niż w przypadku deklaracji INITIAL, wartości te nie mogą być zmieniane podczas wykonywania programu. W podobny sposób można inicjować zmienne, struktury i tablice struktur.

Program rozpoczyna się od przypisania zmiennej BAZA$LOSOWANIA wartości 1250H. Jest to adres początkowy dla prostego generatora liczb pseudolosowych, który wykorzystuje zawartość przypadkowo wypełnionej pamięci (kolejnych komórek od adresu 1250H). Zasadnicza część programu jest zawarta w bloku DO WHILE, stanowiącym nieskończoną pętlę. Początek bloku DO WHILE realizuje algorytm ruchu obiektów przy użyciu instrukcji iteracyjnej DO..TO, która dla kolejnych wartości indeksu **I** oblicza nowe wartości atrybutów **R** i **FI** w kolejnych strukturach tablicy OBIEKT. Do atrybutów struktur używa się dostępu zdalnego, tzn. z nazwą struktury i kropką.

Po obliczeniu nowych współrzędnych dla wszystkich obiektów program przechodzi do losowania numeru obiektu, którego azymut będzie przesyłany do mikrokomputera. Zmienna bazowana LICZBA$LOS ma wartość równą zawartości miejsca pamięci o adresie BAZA$LOSOWANIA. Przy użyciu operatora modulo MOD, dostarczającego reszty z dzielenia dwóch liczb całkowitych, normuje się wartość tej zmiennej do przedziału [0,4], przypisując wynik zmiennej **K.** Następnie zwiększa się wartość bazy o 1, aby w następnym takcie zmienna LICZBA$LOS przyjęła inną wartość. Kolejne instrukcje wprowadzają do rejestrów określonych przez zmienne H$REJ i L$REJ (w strukturze PARAREJHL) bardziej i mniej znaczącą część adresu z tablicy TABLICA$ADRESOW, odpowiadającego wylosowanemu numerowi obiektu. W podobny sposób do rejestrów R$REJ i C$REJ (w strukturze PARAREJBC) wprowadza się obydwa bajty wartości azymutu FI wylosowanego obiektu, wykorzystując funkcje HIGH i LOW. Wartością pierwszej z nich jest bardziej znaczący bajt zmiennej FI typu ADDRESS, zaś drugiej — mniej znaczący bajt tej zmiennej.

Następnie z pary rejestrów H$REJ i L$REJ odczytywany jest adres, pod który należy przesłać zawartości rejestrów B$REJ i C$REJ. Funkcja SHL przesuwa wszystkie bity rozszerzonego H$REJ o osiem pozycji w lewo. ADRES WHL jest bazą dla zmiennej bazowanej ZAWARTOSC$PAMIECI, zatem kolejna instrukcja przypisuje komórce pamięci (zmiennej ZAWARTOSC$PAMIECI) bajt danych znajdujący się w rejestrze C$REJ. Po zwiększeniu adresu (zmienna ADRESWHL) o 1, wykonywane jest analogiczne przypisanie zawartości rejestru B$REJ zmiennej ZAWARTOSC$PAMIECI. W ten sposób do dwóch kolejnych miejsc pamięci wprowadzona zostaje wartość azymutu, a program przechodzi do obliczenia nowych współrzędnych obiektów.

Struktury

Tablice umożliwiają łączenie w grupy elementów tego samego typu (BYTE lub ADDRESS), natomiast dzięki strukturom, możliwe jest łączenie w grupy elementów różnych typów. Każda struktura ma swoją nazwę, która stanowi jednocześnie referencję do atrybutów danej struktury. W deklaracji struktury należy podać również nazwy i typy wszystkich jej atrybutów. Atrybutem struktury może być także tablica.

W przykładzie 3 wykorzystano strukturę do opisu pary rejestrów. Jej atrybutami są dwie zmienne tego samego typu — BYTE. Rozwiązaniem mogłoby być także zastosowanie tablicy. Wybór struktury podyktowany był przede wszystkim dążeniem do uzyskania lepszej czytelności programu.

Tablice struktur są bardzo wygodną konstrukcją wtedy, gdy zachodzi konieczność opisu wielu obiektów przy użyciu tej samej struktury. W tym przykładzie w taki właśnie sposób przedstawiono opis obiektów występujących w symulacji, tj.:

DECLARE OBIEKT (5) STRUCTURE (R BYTE, FI ADDRESS, DR BYTE, DFI ADDRESS);

Przykładowo — dostęp do atrybutu DR trzeciej struktury w powyższej tablicy struktur jest następujący:

OBIEKT (2). DR

Podobnie jak w przypadku zwykłych tablic, elementy tablicy struktur są numerowane od zera. Ważną dziedziną zastosowania tablic struktur jest rachunek macierzowy. W języku PL/M nie zdefiniowano tablic dwuwymiarowych, zastępują je jednak tablice struktur z tablicą jako atrybutem, np.:

DECLARE WIERSZ (20) STRUCTURE (KOLUMNA (20) ADDRESS);

Powyższa deklaracja reprezentuje tablicę dwuwymiarową 20×20, a jej **(i, j)**-ty element ma postać

WIERSZ (I). KOLUMNA (J)

```
PODWOJNA$SUMA: DO;

DECLARE N LITERALLY '20';
DECLARE W(N) STRUCTURE( A(N) ADDRESS,X BYTE,Y BYTE);
DECLARE (I,J) BYTE,Q ADDRESS INITIAL(0);

   DO I = 0 TO N-1;
   DO J = 0 TO N-1;
      Q = Q + W(I).A(J) * W(I).X * W(J).Y;
   END;
   END;

END PODWOJNA$SUMA;
```

Wydruk 2. Obliczanie wartości wyrażenia $\sum_{i=0}^{19}\sum_{j=0}^{19} a_{ij}x_iy_i$

Na wydruku 2 przedstawiono program realizujący obliczenie wyrażenia w postaci:

$$\sum_{i=0}^{19}\sum_{j=0}^{19} a_{ij}x_iy_i$$

gdzie **x** i **y** są wektorami jednowymiarowymi, zaś **a** jest dwuwymiarową tablicą współczynników.

Zmienne bazowane

W języku PL/M istnieje możliwość pośredniego odwoływania się do zmiennych. Wykorzystując zmienną bazowaną, jedną nazwą można odwoływać się do różnych danych w zależności od wartości tzw. bazy, stanowiącej adres w pamięci. Kompilator nie przydziela zmiennej bazowanej żadnego adresu, ponieważ adres ten jest uzależniony od wartości bazy. Deklaracja zmiennej bazowanej zawsze musi być poprzedzona deklaracją bazy (por. przykład 1). Słowo BASED musi występować za nazwą zmiennej bazowanej i poprzedzać nazwę bazy. Wartością zmiennej bazowanej jest zawartość komórki pamięci (BYTE lub ADDRESS w zależności od typu zmiennej) o adresie określonym przez bazę.

PRZYKŁAD 4 — WYSZUKIWANIE DOKUMENTÓW

W banku danych znajdują się dokumenty posiadające dwa atrybuty: nazwę i czterocyfrowy numer. Wyszukuje się dokumenty o nazwach różnych od zera, które po konwersji na kod ASCII wyprowadza się do zadanego bufora.

Program rozwiązujący powyższy problem przedstawiono na wydruku 3. W porównaniu z poprzednimi zawiera on niewielką liczbę instrukcji, natomiast część deklaracyjna jest bardziej rozbudowana. Uzyskuje się to dzięki wprowadzeniu procedur. W pierwszej części zdefiniowano synonimy LICZBA$DOKUMENTOW i DLUGOSC$BUFORA, tablicę BUFF stanowiącą bufor, do którego wprowadza się numery dokumentów (każdy numer wypełnia cztery elementy tablicy), oraz — tablicę struktur dokumentów o nazwie DOKUMENT z dwoma atrybutami: NAZWA i NUMER. Kolejne deklaracje dotyczą procedur CYFRA i KONWERSJA.

```
BANK$DOKUMENTOW: DO;

DECLARE DCL LITERALLY 'DECLARE',LICZBA$DOKUMENTOW LITERALLY '50',
DLUGOSC$BUFORA LITERALLY '200';
DCL BUFF(DLUGOSC$BUFORA) BYTE, P BYTE;
DCL Q ADDRESS INITIAL(0);
DCL DOKUMENT(LICZBA$DOKUMENTOW) STRUCTURE(NAZWA BYTE, NUMER ADDRESS);

CYFRA: PROCEDURE(I) BYTE;
DCL I BYTE;
DO CASE I;
 RETURN '0';
 RETURN '1';
 RETURN '2';
 RETURN '3';
 RETURN '4';
 RETURN '5';
 RETURN '6';
 RETURN '7';
 RETURN '8';
 RETURN '9';
END;
END CYFRA;

KONWERSJA: PROCEDURE(ADRES$DOKUMENTU, ADRES$BUFORA);
DCL (ADRES$DOKUMENTU,ADRES$BUFORA,ZMIENNA$POMOCNICZA) ADDRESS, J BYTE;
DCL BUFOR BASED ADRES$BUFORA(4) BYTE;
DCL BOXX BASED ADRES$DOKUMENTU STRUCTURE(NAZWA BYTE,NUMER ADDRESS);

ZMIENNA$POMOCNICZA = BOXX.NUMER;
DO J = 0 TO 3;
 BUFOR(3-J) = CYFRA( ZMIENNA$POMOCNICZA MOD 10 );
 ZMIENNA$POMOCNICZA = ZMIENNA$POMOCNICZA / 10;
END;

END KONWERSJA;
/* **** PROGRAM GLOWNY **** */
DO P = 0 TO LICZBA$DOKUMENTOW - 1;
 IF DOKUMENT(P).NAZWA <> 0 THEN
 DO;
  CALL KONWERSJA(.DOKUMENT(P),.BUFF + 4*Q);
  Q = Q + 1;
 END;
END;

END BANK$DOKUMENTOW;
```

Wydruk 3. Przykład modułu 4

Procedura CYFRA jest procedurą funkcyjną, dotychczas nazywaną funkcją, a jej wartością jest cyfra w kodzie ASCII, odpowiadająca wartości parametru aktualnego. Zasadniczą jej treść stanowi instrukcja DO CASE. Wywołanie funkcji z parametrem aktualnym powoduje, że liczba ta (np. 5) zostaje przypisana parametrowi formalnemu **I**, który z kolei określa numer instrukcji do wykonania w bloku DO CASE. W tym przypadku będzie to piąta instrukcja, a zatem wartość powrotna funkcji będzie wynosiła '5'.

Procedura o nazwie KONWERSJA ma dwa parametry formalne, ADRES$DOKUMENTU i ADRES$BUFORA. Obie te zmienne stanowią bazy dla struktury bazowanej BOXX i tablicy bazowanej o nazwie BUFOR. Działanie procedury polega na konwersji atrybutu NUMER struktury DOKUMENT z postaci dwójkowej na kod ASCII. Dostęp do tego atrybutu uzyskuje się przez zmienną bazowaną BOXX, przy czym jej bazę stanowi adres struktury będący parametrem aktualnym. Algorytm konwersji jest prosty i polega na kolejnym poddawaniu wartości atrybutu NUMER operacjom: modulo 10 i dzielenie przez 10. Po konwersji na kod ASCII przy użyciu funkcji CYFRA, kolejne cyfry są wstawiane do bufora BUFOR poczynając od ostatniej pozycji.

Po deklaracjach procedur rozpoczyna się program główny, tj. ciąg instrukcji modułu. W pętli iteracyjnej DO..TO sprawdza się warunek równości nazwy każdego dokumentu z zerem. W przypadku różnicy wywoływana jest procedura KONWERSJA.

Pierwszym parametrem aktualnym tej procedury jest adres kolejnego elementu tablicy struktur DOKUMENT. Odwoływanie się do adresów zmiennych, tablic i struktur jest doknywane przez podanie kropki przed nazwą. Przykładowo — wartością odwołania **DOKUMENT(P)** jest adres początku **p**-tej struktury w tablicy struktur DOKUMENT

Drugim parametrem aktualnym jest adres elementu tablicy BUFF, od którego wpisana ma być czterocyfrow.

wartość numeru **Q**-tego wyszukanego dokumentu. Tak określone parametry aktualne stanowią bazy dla tablicy i struktury bazowanej, umożliwiające dostęp do odpowiednich elementów tablicy i atrybutów struktury. W ten sposób analizowane są wszystkie dokumenty znajdujące się w banku danych (tablicy struktur). Program oblicza dodatkowo liczbę wybranych dokumentów wykorzystując do tego celu zmienną **Q**.

Procedury i funkcje

Deklaracja procedury ma następującą postać:

```
<nazwa>: PROCEDURE (<lista parametrów formalnych>);
         <treść procedury>;
         END <nazwa>;
```

Treść procedury obejmuje deklaracje i instrukcje. W treści procedury należy zadeklarować jej wszystkie parametry formalne (o ile występują). Deklaracja procedury w programie musi pojawić się przed jakimkolwiek odwołaniem się do niej. Realizacja ciągu instrukcji procedury następuje po jej wywołaniu instrukcją CALL. W szczególności może to być pierwsza instrukcja w programie. Po wykonaniu treści procedury następuje przejście do pierwszej instrukcji występującej za instrukcją wywołującą.

Parametrami formalnymi procedury nie mogą być elementy tablic i struktur. Parametry formalne należy deklarować w treści procedury. Parametry aktualne mogą być wyrażeniami. W przypadku procedury KONWERSJA, parametrowi formalnemu ADRES$DOKUMENTU przypisywana jest, w chwili wywołania procedury, wartość adresu **p**-tej struktury, tj. **.DOKUMENT(P).** Parametr ADRES$BUFORA przyjmuje wtedy wartość równą wartości wyrażenia **.BUFF+4*Q.** Liczba parametrów aktualnych musi być równa liczbie parametrów formalnych procedury.

W języku PL/M istnieje możliwość deklarowania procedur z jednoczesnym podaniem ich typu (BYTE lub ADDRESS). Tego rodzaju procedury noszą nazwę funkcji. Deklaracja funkcji ma następującą postać:

```
<nazwa>: PROCEDURE (<lista parametrów formalnych>) <typ>;
         <treść funkcji>;
         RETURN <wyrażenie>;
         END <nazwa>;
```

W celu wywołania funkcji wystarczy podać jej nazwę wraz z parametrami aktualnymi. Funkcja musi zawierać w swej treści co najmniej jedną instrukcję RETURN, która powoduje automatyczne przypisanie nazwie funkcji wartości wyrażenia występującego po słowie RETURN. A zatem funkcja może być argumentem w dowolnym wyrażeniu, w którym wystąpiła jej nazwa. Zasady przekazywania parametrów w funkcjach są analogiczne jak w przypadku procedur.

Tablice i struktury bazowane

Tablice i struktury bazowane stanowią rozszerzenie koncepcji zmiennych bazowanych. Jeżeli zmienna bazowana umożliwia dostęp do jednego miejsca pamięci o adresie równym bazie, to w przypadku tablic i struktur dla jednej wartości bazy możliwy jest dostęp do wielu kolejnych miejsc pamięci. Podobnie jak dla zmiennym bazowanych, dla tablic i struktur bazowanych nie rezerwuje się pamięci. Deklaracja każdej tablicy i struktury bazowanej musi być poprzedzona deklaracją bazy.

```
PRZYKLAD: DO;

DECLARE DCL LITERALLY 'DECLARE';

MOVE: PROCEDURE(ADR1,ADR2,N);
DCL (ADR1,ADR2,N,I) ADDRESS;
DCL (TAB1 BASED ADR1,TAB2 BASED ADR2)(1) BYTE;
   DO I = 0 TO N-1;
     TAB2(I) = TAB1(I);
   END;
END MOVE;

CALL MOVE(2000H,3200H,100H);

END PRZYKLAD;
```

Wydruk 4. Procedura MOVE

Na wydruku 4 przedstawiono procedurę MOVE, która przenosi **N** bajtów danych, poczynając od adresu ADR1, w obszar zaczynający się od adresu ADR2. Zakłada się, że oba obszary pamięci nie zachodzą na siebie. W programie wykorzystano dwie tablice bazowane o bazach równych ADR1 i ADR2, dzięki czemu uzyskano dostęp do kolejnych komórek zadanych obszarów pamięci, które wynoszą 2000H-20FFH i 3200H-32HFH. Przez sparametryzowanie procedury MOVE możliwe jest przesuwanie dowolnych bloków pamięci w różne obszary pamięci mikrokomputera.

PRZYKŁAD 5 — OBSŁUGA PRZERWAŃ

Obsługa przerwania polega w tym przykładzie na obliczeniu wartości wielomianu trzeciego stopnia dla odczytanej z czujnika wartości wejściowej. Obsługę realizuje procedura o nazwie PRZERWANIE z atrybutem INTERRUPT 2 (wydruk 5). Z chwilą zgłoszenia przerwania z poziomu drugiego, jeżeli przerwania są dozwolone, aktualnie wykonywana instrukcja (faktycznie rozkaz mikroprocesora) zostaje dokończona, a następnie wywoływana jest procedura obsługi, tj. PRZERWANIE. Procedura ta oblicza wartość zmiennej WYJŚCIE w oparciu o funkcje: WIELOMIAN i ODCZYTZCZUJNIKA. Druga z tych funkcji dostarcza wartości danej, odczytanej z drugiego portu mikroprocesora.

```
OBSLUGA$PRZERWANIA: DO;
DECLARE BI LITERALLY '2';
DECLAR  A(4) ADDRESS, WYJSCIE ADDRESS;
ODCZYT$Z$CZUJNIKA: PROCEDURE BYTE REENTRANT;
                   RETURN INPUT(BI);
END ODCZYT$Z$CZUJNIKA;

WIELOMIAN: PROCEDURE(X) ADDRESS REENTRANT;
DECLARE (X,I) BYTE, Y ADDRESS;
Y = 0;
DO I = 0 TO 3;
 Y = A(I) + X*Y;
END;
RETURN Y;
END WIELOMIAN;

PRZERWANIE: PROCEDURE INTERRUPT 2;
            WYJSCIE = WIELOMIAN(ODCZYT$Z$CZUJNIKA);
END PRZERWANIE;

END OBSLUGA$PRZERWANIA;
```

Wydruk 5. Przykład modułu 5

W przypadku, gdy procedury obsługi przerwań kilku poziomów wykorzystują tę samą procedurę, dla której zmiennych zarezerwowano stałe komórki pamięci, to istnieje niebezpieczeństwo, że zawartość tych komórek może ulec zniszczeniu. W celu uniknięcia tej niedogodności wprowadzono atrybut REENTRANT. Zapewnia on, że wartości zmiennych lokalnych procedur wywoływanych podczas obsługi przerwań są przechowywane na stosie, dzięki czemu nie są niszczone przez inne programy. Z tego powodu funkcje WIELOMIAN i ODCZYTZCZUJNIKA mają atrybuty REENTRANT. Ogólnie, procedurę z tym atrybutem wykorzystuje się, jeśli jest ona zawieszana przez przerwanie, którego obsługa także ją wywołuje, lub jeśli jest ona rekursywna.

Zabranianie bądź zezwalanie przerwań realizują instrukcje DISABLE i ENABLE. Każda procedura obsługi przerwań zabrania innych przerwań (system wykonuje jako pierwszą instrukcję DISABLE). Chcąc zatem realizować, np. priorytetowy system przerwań, należy w procedurach obsługi zezwolić na przerwanie instrukcją ENABLE. Po wyjściu z każdej procedury obsługi przerwania, niezależnie od tego, czy zewala ona bądź zabrania przerwania, przerwania zostają dozwolone. Procedury z atrybutem INTERRUPT nie mogą być funkcjami oraz nie mogą mieć żadnych parametrów formalnych. Ponadto, numer poziomu przerwania określony w deklaracji procedury po słowie INTERRUPT może pojawić się w programie tylko jeden raz.

ZASADY ŁĄCZENIA MODUŁÓW

Ważnym czynnikiem, mającym duże znaczenie w programowaniu jest modularyzacja programów. Z tego względu wszsytkie przykładowe programy stanowiły kompletne moduły, w których użyto podstawowych konstrukcji języka PL/M. Praktyka wykazuje, że podstawową rolę w tworzeniu dużych programów odgrywają procedury oraz że każ-

dej procedurze powinien odpowiadać oddzielny moduł. Procedura zadeklarowana w danym module, podobnie jak zmienne, tablice lub struktury, jest w nim lokalna, tzn. jest dostępna tylko w tym module. Do rozszerzenia zakresu działania procedury na inne moduły służą atrybuty PUBLIC i EXTERNAL. Procedura zadeklarowana w module z atrybutem PUBLIC jest dostępna także w innych modułach pod warunkiem, że w każdym z nich zostanie zadeklarowana z atrybutem EXTERNAL. Wynika stąd, że jedna procedura może być zadeklarowana z atrybutem PUBLIC tylko jeden raz, natomiast z atrybutem EXTERNAL — wielokrotnie, w zależności od liczby modułów, w których jest wywoływana. Deklaracja procedury z obydwoma atrybutami w jednym module nie ma sensu. Powyższe uwagi dotyczą także zmiennych, tablic i struktur.

Deklaracja procedury z atrybutem PUBLIC stanowi klasyczną deklarację procedury, przy czym w nagłówku procedury podaje się dodatkowo słowo PUBLIC, np:

```
KONWERSJA: PROCEDURE(ADRES$DOKUMENTU, ADRES$BUFORA) PUBLIC;
           <treść procedury>;
           END KONWERSJA;
```

Deklaracja tej samej procedury z atrybutem EXTERNAL (w innym module) ma postać:

```
KONWERSJA: PROCEDURE (A, B) EXTERNAL;
           DECLARE (A, B) ADDRESS;
           END; .
```

Treść procedury z atrybutem EXTERNAL zawiera tylko deklarację parametrów formalnych, przy czym wymagana jest jedynie zgodność typów parametrów, natomiast ich nazwy mogą być różne. W przypadku zmiennych i tablic, atrybuty PUBLIC i EXTERNAL wymienia się po deklaracji typu, zaś w przypadku struktur — po specyfikacji atrybutów.

Zasadę roszerzania zakresu działania zmiennych, tablic struktur i procedur zilustrowano na wydruku 6. Tablica TAB, zmienna ZMIENNA, struktura X oraz procedura

```
MODUL1: DO; /* MODUL GLOWNY */

DECLARE EXTR LITERALLY
'EXTERNAL';
DECLARE ZMIENNA BYTE PUBLIC;
DECLARE TAB(10) ADDRESS EXTR;
DECLARE X STRUCTURE(B ADDRESS,
A BYTE) PUBLIC;

KONW: PROCEDURE(X,Y) EXTR;
DECLARE (X,Y) ADDRESS;
END;

  /* PROGRAM GLOWNY */
CALL KONW(.X.B,.TAB);

END MODUL1;
```

```
MODUL2: DO;

DECLARE EXTR LITERALLY
'EXTERNAL';
DECLARE ZMIENNA BYTE EXTR;
DECLARE TAB(10) ADDRESS PUBLIC;
DECLARE X STRUCTURE(B ADDRESS
A BYTE) EXTR;

KONW: PROCEDURE(A$D,A$B) PUBLIC;
      TRESC PROCEDURY
END KONW;
CYFRA: PROCEDURE(I) BYTE;
      TRESC FUNKCJI
END CYFRA;

END MODUL2;
```

Wydruk 6. Zasada użycia atrybutów PUBLIC i EXTERNAL

KONW są dostępne w obu modułach, natomiast funkcja CYFRA jest lokalna w module 2 i może być dostępna tylko w nim. Chcąc rozszerzyć zakres jej działania należałoby dodać w nagłówku jej deklaracji słowo PUBLIC.

Atrybuty PUBLIC i EXTERNAL stanowią łączniki między poszczególnymi modułami, umożliwiają m.in. komunikację i wymianę informacji między nimi. Łączenie modułów w jeden moduł, po niezależnej kompilacji każdego z nich, dokonuje się przy użyciu programu o nazwie LINK, wchodzącego w skład programów systemu operacyjnego. Moduł otrzymany po połączeniu nie zajmuje więcej pamięci niż odpowiadający mu moduł powstały wskutek kompilacji jednego dużego programu.

* * *

Z przedstawionego opisu języka PL/M wynika, że stanowi on bardzo dobre narzędzie dla projektanta-programisty na początkowych etapach procesu projektowania systemu mikroprocesorowego. Różnorodność konstrukcji językowych i modularność programów umożliwia stosunkowo łatwy opis algorytmów i otoczenia, jak również szybkie uruchamianie programów. Na poziomie tego języka powinny być także łatwo eliminowane błędy projektowe.

Miejsce grafiki komputerowej

Czwartą stronę okładki będziemy odtąd przeznaczać przede wszystkim na grafikę komputerową. Nie mamy autorów, którzy zapełnialiby stale to miejsce. Proponujemy więc wszystkim zainteresowanym udział w konkursie na najlepsze czarno-białe realizacje. Nagrodą będzie publikacja. I oczywiście honorarium. (Red.)

Naczelna Organizacja Techniczna

podjęła starania na rzecz prezentacji udziału inżynierów i techników w procesie tworzenia gospodarki narodowej w okresie czterdziestolecia PRL, ze szczególnym uwzględnieniem pierwszych lat odbudowy kraju po II wojnie światowej.

NOT liczy na zgłoszenie się tych inżynierów i techników, którzy w latach 1944—1949 przystąpili do odbudowy różnych dziedzin gospodarki (m.in. jako pełnomocnicy PKWN, uczestnicy grup operacyjnych, pracownicy tworzonych wówczas struktur organizacyjnych życia gospodarczego w Polsce).

Zgłoszenia należy kierować pod adresem Naczelnej Organizacji Technicznej — 00-950 Warszawa, ul. Czackiego 3/5, z dopiskiem: 40-lecie PRL. W zgłoszeniu należy podać imię, nazwisko i adres, a także krótkie omówienie swojej działalności w tamtym okresie.

Druga część opisu mikrokomputera APPLE poświęcona została oprogramowaniu. Tym razem jednak rzecz nie w opisie fantastycznych gier komputerowych czy programów graficznych — pozwalających architektowi zaprojektować dom od dachu po fundamenty z uwzględnieniem kształtu krzeseł w jadalni. Aby realizacja wspomnianych programów była możliwa w mikrokomputerach, stosowany jest program zarządzający o nazwie MONITOR (lub SYSTEM OPERACYJNY, gdy obsługuje również współpracę z dyskami). W jego skład wchodzi szereg procedur, od stopnia wyrafinowania których zależy prostota (a często i możliwości) programów użytkowych.

Procedury realizujące zbliżone funkcje można obecnie spotkać w oprogramowaniu zasadniczym większości „szanujących się" mikrokomputerów. Tym, którzy kompletują program MONITOR dla komputera własnej konstrukcji, radzimy skorzystać z dobrze sprawdzonego wzorca.

apple

O jabłku opowieść (2)

Oprogramowanie

Opisując mikrokomputer APPLE II warto wspomnieć o dwóch wersjach oprogramowania podstawowego mikroprocesora 6502. Wersją pierwszą, nieco starszą jest SMR (System Monitor ROM). W wersji tej dostępny jest język INTEGERBASIC (interpreter języka BASIC z ograniczeniem do liczb całkowitych) oraz MINI-ASSEMBLER 6502. Drugą wersją jest system AMR (Autostart Monitor ROM). Obok znacznie bogatszego oprogramowania podstawowego, językiem wysokiego poziomu jest tu APPLESOFT.

A oto podstawowa struktura oprogramowania umieszczonego w pamięci ROM dla AMR. Zarządcą i zarazem niewolnikiem systemu jest program SYSTEM MONITOR. Kontroluje on realizację wszystkich programów, tłumaczy niektóre na język wewnętrzny mikroprocesora 6502, ale też wszystkie one przekazują mu do wykonania różne polecenia.

SYSTEM MONITOR AMR zawiera między innymi mini DISASSEMBLER (mini — ponieważ nie potrafi przetwarzać adresów symbolicznych). Mini DISASSEMBLER uwzględnia 56 typów rozkazów i 13 trybów adresowania. Przekazywana informacja składa się z trzech części: adresu, kodu (w systemie szesnastkowym) oraz rozkazu. SYSTEM MONITOR składa się z szeregu procedur. Oto najważniejsze z nich:

Podprogram COUT (ang. Character OUTput) — wyprowadza znaki na urządzenia zewnętrzne w czterech postaciach: normal, inverse (negatyw), flash (migająca), esc-function (znak edytorski). Znaki te kodowane są w akumulatorze. Podprogram ten uwzględnia jedynie duże litery, ma pełną kontrolę nad kursorem, jego aktualnym położeniem na ekranie, organizuje tzw. okna, czyli liczbę kolumn i wierszy. Maksymalnie dopuszcza 40 kolumn w 24 wierszach. Program ten uruchamia w określonych warunkach głośnik, który generuje sygnał akustyczny o częstotliwości 1 KHz co 1/10 sekundy.

Podprogram COUT1 — wyprowadza znaki z akumulatora na ekran monitora. Może być on wywoływany przez COUT.

Podprogram SETINV (ang. SET INVerse) — uruchamiany bezpośrednio przez COUT. Przekształca obraz ekranu na jego negatyw.

Podprogram SETNORM (ang. SET NORMal) — powoduje powrót do pozytywu.

Podprogram CROUT (ang. Carriage Return OUTput) — współpracuje z urządzeniami peryferyjnymi, przesyłając do nich informację o funkcji RETURN kończącej dany rozkaz, czy linię.

Podprogram CROUT1 — współpracuje z podprogramem COUT dokonując odpowiednich korekt i wymazując niepotrzebne informacje z ekranu i pamięci.

Podprogram PRBYTE (ang. PRint BYTE) — kontroluje urządzenia wyjściowe; przesyła do nich zawartość akumulatora.

Podprogram BELL — generuje sygnał akustyczny poprzez urządzenie wejściowe.

Brak wyobraźni

Znajomy młody informatyk, do niedawna z irytacją reagujący na przejawy zniechęcenia wśród kolegów i demencji przełożonych, oświadczył nagle stanowczo, że cała praca w tej informatyce jest bezsensowna. Że być informatykiem — głupio; i że od dzisiaj liczą się dla niego tylko pieniądze. Pisać do INFORMATYKI nie będzie, dopóki gdzie indziej zapłacą więcej. Na argument, że mikroinformatyka zmienia sytuację, że nie całkiem zależna jest od resortowych „informatycznych" bankructw, odpowiadał z przekonaniem, że „mikroinformatyka pójdzie tą samą drogą". To jest — donikąd, niszczona skutecznie przez różne „całościowe" wizje i wytyczne.

Dzisiaj jest to już dość częsta postawa, wyrażająca z a s a d ę, będącą symptomem informatycznej aury. Postawa, która znajduje sobie wiele przekonywających uzasadnień, potrafiąca znaleźć poparcie w wielu liczbach, statystykach, prognozach. I choć może jest to na swój sposób postawa rozumna, to przecież jednocześnie — nie do przyjęcia.

Rezygnacja z aktywności może mieć dziś miejsce (i ma!) w każdej niemal sferze życia społecznego. Niszczy lub degeneruje społeczne funkcje, do jakich — w następstwie świadomego wyboru czy nie — jesteśmy powołani. Poddanie się apatii powoduje, że ta nasza nowa rola (przeczekiwacza) obezwładnia nas samych, wpycha w ślepy zaułek życia rodzinnego, towarzyskiego... Rodzi cynizm i zgorknienie.

Konsekwencja własnych poczynań w życiu publicznym wymaga dzisiaj sporego hartu, samodyscypliny, chroniącej przed mirażem tanich zasług i jałowych pochwał. I zapewne musi to być bardziej „na przekór" niż „w zgodzie" z otoczeniem. Ale tak czy inaczej ważniejszy jest wybór „za czymś" niż „przeciw czemuś". Owo „za" rozstrzyga o sensie własnych poczynań.

To nie agitacja. Chcę tylko zwrócić uwagę na możliwość daną przez — jeszcze horrendalnie drogą, ale już dostępną — mikroinformatykę. Że ze świadomością celu (także społecznego), a zyskawszy dogodne narzędzie, można — drążąc powoli — ruszać bryłę z posad...

Trzeba hartu, pewności swego. Trzeba umiejętności. I konsekwencji. Dopiero pogodzenie się ze słabością oznacza porażkę.

Ów młody informatyk znalazł się tam (także w sensie duchowym), gdzie dają więcej pieniędzy, gdzie może wyślą za granicę, dadzą awans. Jeśli nawet kosztem zawodowej i społecznej degradacji, to przecież — korzyści większe.

Dobrze przynajmniej, że nie udaje satysfakcji.

ZBIGNIEW GLUZA

Podprogram BELL1 — generuje sygnał akustyczny 1 kHz z głośnika komputera co 1/10 sekundy.

Podprogram RDKEY (ang. ReaD a KEY) — reaguje na wprowadzanie pojedynczych znaków z klawiatury; uwidocznia mały migający kwadracik, określający aktualną pozycję kursora kontrolowanego przez podprogram COUT. Podczas gdy RDKEY czeka na naciśnięcie klawisza, dodaje co 1/10 sekundy jedynkę do określonej pary komórek pamięci. Gdy kod znaku zostanie wprowadzony, wartości z obu komórek są dodawane tworząc liczbę z zakresu od 1 do 65 535. Ponieważ wynik jest zdarzeniem losowym, właściwość tę wykorzystuje się do tworzenia **generatora liczb losowych.**

Podprogram RDCHAR (ang. ReaD CHARacter) — współpracuje z RDKEY kontrolując jego działanie.

Podprogram KEYIN — odczytuje znaki z klawiatury. Po naciśnięciu klawisza, kursor na moment znika, a kod znaku zostaje przesłany do akumulatora.

Podprogram GETLIN (ang. GET a LINe) — pełni podobne funkcje do podprogramu RDKEY. W przeciwieństwie do niego, reaguje jednak nie na kod pojedynczego znaku, lecz na instrukcję RETURN. Specjalnym kodem informuje, w jakim języku aktualnie pracuje komputer. GETLIN współpracuje z podprogramami RDKEY oraz COUT i COUT1.

RDKEY przesyła kody z klawiatury do bufora w pamięci, skąd następnie zostaje on przepisany na ekran monitora. Wprowadzenie RETURN powoduje, że RDKEY wywołuje program GETLIN, a ten — ponownie RDKEY itd. Całością operacji steruje program COUT, odpowiednio sterujący kursorem i wyświetlaniem wprowadzonych znaków. GETLIN dopuszcza długość linii do 255 znaków. W przypadku przekroczenia tej granicy GETLIN kasuje całą linię, uruchamiając podprogram BELL1, generujący dźwięk o wysokości 1 kHz, po czym wraca do programu pierwotnego.

Podprogram WAIT — przerywa realizację programu na czas zależny od zawartości akumulatora. Zatrzymuje on wykonanie programu od 13 μs do 1/6 s.

Kolejne istotne podprogramy, charakterystyczne jednak dla specyficznej grafiki APPLE'a to: **SETCOL, NEXTCOL, PLOT, HLINE, VLINE, SCRN-SUBROUTINES.**

Organizacja SYSTEMU MONITOR AMR umożliwia znaczną jego rozbudowę przez dołączenie dodatkowych pamięci PROM i EPROM. Można zatem tworzyć własne podprogramy tego systemu.

Oprogramowanie dla APPLE'A zazwyczaj dostarczane jest na dyskietkach (5 1/4″ lub znacznie rzadziej 8″). Organizacja dyskietek 5 1/4″ polega na podziale dysku na 35 ścieżek (ang. tracks). Trzy z nich (najbadziej zewnętrzne) zajmuje na ogół sytem operacyjny DOS, który steruje współpracą z napędami dyskietek. Ścieżka środkowa rezerwowana jest na spis plików dostępnych na dysku. Dla użytkownika pozostaje zatem trzydzieści jeden wolnych ścieżek.

Każda ścieżka składa się z 16 bądź 13 sektorów, a jeden sektor umożliwia operowanie blokami o pojemności 256 bajtów. Użytkownik ma więc do dyspozycji 126 976 bajtów na każdej dyskietce. Bardzo pożyteczną procedurą (będącą częścią systemu operacyjnego) jest podprogram **RWTS** (ang. Read or Write a Track and Sector). Umożliwia on dostęp do dowolnego sektora dowolnej ścieżki.

```
 5  REM  SITO ERATOSTENESA
 6  REM  LICZBY PIERWSZE
10  HOME :INPUT "ILOSC LICZB PIERWSZYCH N=";N
14  PRINT CHR$ (12)
15  PRINT :GET A$:IF A$= "3"THEN PR# 3
20  DIM X(N):DIM Y(N)
30  X(0)= 3:Y(0)= 3:K= 1:V= 0
33  PRINT :PRINT :PRINT "2";" ";"3";
40  FOR I= 1TO N
44  : X(I)= X(I- 1)+ 2
50  : FOR L= 0TO K
55  : : UNLESS (X(I)/ Y(L)= INT (X(I)/ Y(L)))
58  : : : IF (Y(L)> SQR (X(I)))
60  : : : : V= V+ 1:L= V:K= V+ 1
66  : : : : Y(L)= X(I)
70  : : : : PRINT " ";Y(L);
77  : : : : GOTO 99
80  : : : :FIN
88  : : :NEXT L
90  : :FIN
99  :NEXT I
```

Na wydruku zaprezentowano próbkę programowania w języku APLUS. Program realizuje algorytm wyszukujący liczby pierwsze, znany pod nazwą „Sito Eratostenesa". Zamieszczone obok liczby otrzymano właśnie w wyniku realizacji programu. Strukturalne możliwości programowe w języku APLUS pozwalają na wyjątkowo przejrzyste przełożenie algorytmu na „język komputerowy". Nie znane są nam jednak implementacje tego języka dla komputerów innych niż produkowane przez firmę APPLE. Dlatego też nie podzielamy zgłoszonej nam przez autora tekstu opinii o potrzebie intensywnej popularyzacji tej właśnie odmiany języka BASIC. Prosimy jednak Czytelników o ewentualną korektę naszych przekonań. (Red.)

Na potrzeby APPLE'a stworzono wiele makroasemblerów, dostarczanych jako oprogramowanie dodatkowe na dyskach elastycznych. **MERLIN ASSEMBLER.** Makroasembler ten, napisany specjalnie na potrzeby APPLE'a, daje możliwość rozbudowy systemu redagowania tekstów, może współpracować z płytkami CARD-80 i pamięci 16 KB.

MICROSOFT'S ASSEMBLY LANGUAGE DEVELOPMENT SYSTEM. System ten składa się z trzech programów ASSEMBLER:
— ASSEMBLER 8080
— ASSEMBLER Z80
— 6502-kod.
Programy te można używać niezależnie bądź razem.

ALD SYSTEM II. Makroasembler dla APPLE'a napisany przez Paula Lutusa.

EDITOR/ASSEMBLER, Asembler ten ma bogate możliwości redagowania plików.

LISA ASSEMBLER, także autorstwa Paula Lutusa. Asembler ten umożliwia wykorzystanie całej gamy możliwości graficznych APPLE'a przez dołączanie procedur bibliotecznych. Stanowił on punkt wyjściowy dla propozycji graficznych w mikrokomputerze LISA.

```
]RUN
ILOSC LICZB PIERWSZYCH N=1000

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 7
9 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163
167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251
257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349
353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443
449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557
563 569 571 577 587 593 599 601 607 613 617 619 631 641 643 647
653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757
761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863
877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983
991 997 1009 1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069
1087 1091 1093 1097 1103 1109 1117 1123 1129 1151 1153 1163 1171
1181 1187 1193 1201 1213
```

Najistotniejsze i największe z programów użytkowych, to cztery „mega-programy":

WORDSTAR to program do redagowania tekstów. W połączeniu z płytką CARD-80 daje możliwość formatowania tekstu bezpośrednio na ekranie (ang. ON-SCREEN-FORMATING). Program ten ma bogate możliwości redakcyjne, a nawet jest w stanie kontrolować swoje działanie, np. sprawdza, czy podział sylabiczny jest sensowny, podaje też użytkownikowi konkretne propozycje. Wymaga zastosowania systemu CP/M.

VISICALC pracuje w tabeli 63 kolumn oraz 254 wierszy. W każdym z 63 × 254 pól może być zakodowana albo litera, albo wartość arytmetyczna. Poszczególne pola mogą być ze sobą powiązane przez matematyczne formuły. Jeżeli w tabeli zmienia się dana wartość, wszystkie pozostałe zależne od niej wartości również odpowiednio się zmieniają. Jest to punkt wyjścia do konstruowania skomplikowanych modeli fizycznych (np. teoria pola), bez potrzeby układania skomplikowanych programów, czy też znakomite narzędzie dla ekonomisty. Program wymaga stosowania systemu operacyjnego CP/M.

MULTIPLAN ma wszystkie zalety programu VISICALC oraz dodatkowo możliwość konstruowania pól chronionych a także sortowania w zależności od posiadanej cechy. Program ten wymaga stosowania systemu operacyjnego CP/M.

PIE WRITER to program do redagowania tekstów w 80 kolumnach. Formatowanie tekstu możliwe jest w wariantach pisma blokowego oraz ozdobnego.

PIOTR TYMOCHOWICZ

Większość debiutujących użytkowników mikrokomputerów staje wobec problemu zdobycia oprogramowania. Jedną z dróg może być przenoszenie programów z dużych komputerów. Wprawdzie programy te (często napisane w FORTRANIE) nie działają zbyt efektywnie na komputerach wykorzystujących mikroprocesory 8-bitowe, lecz w warunkach domowych — mogą „liczyć się" choćby całą noc. Podejście takie jest na pewno bardziej rozsądne niż pisanie samemu wszystkiego od początku. Autorzy poniższego tekstu zamiast żmudnego przepisywania wydruków zrealizowali — kosztem pracy koncepcyjnej — rozwiązanie na miarę krajowych możliwości. W rezultacie przenieśli kilkadziesiąt procedur numerycznych z minikomputera PDP-11/45 na mikrokomputer TRS 80.

Warto jeszcze wspomnieć, że w ten sposób można przenosić programy wynikowe generowane przez kompilatory skrośne dużych maszyn cyfrowych.

Przenoszenie informacji z komputerów dużych na osobiste

Problem komunikacji komputera osobistego z dużą maszyną cyfrową rozwiązywany jest na ogół poprzez użycie modemu dołączonego do zwykłego aparatu telefonicznego. Tyle że w naszych warunkach jest to rozwiązanie nierealne i trzeba, niestety szukać innych dróg. Problem polega na tym, że duże komputery używają do przenoszenia informacji taśm magnetycznych lub co najwyżej dysków elastycznych 8". Nośniki te nie stanowią standardowego wyposażenia (ze względu na cenę) komputera domowego. Na szczęście wiele minikomputerów wyposażonych jest w pamięci kasetowe PK-1, które zapisują informacje na zwykłych kasetach magnetofonowych. Poniżej omówimy sposób odczytu informacji z takiej kasety.

Kodowanie fazowe i sprzęt potrzebny do odczytu

Pamięć kasetowa PK-1 zapisuje informacje z gęstością 32 bity na milimetr (na jednej ścieżce), z szybkością 12,7 cm/s. Zapis informacji doknywany jest metodą przemagnesowania nośnika magnetycznego taśmy przez głowicę zapisu. Wykorzystywana jest mteoda kodowania fazowego, tzw. PE (ang. Phase Encoded), polegająca na tym, że:

— bit „1" określony jest przez zmianę kierunku strumienia magnetycznego w kierunku zgodnym z polaryzacją magnetyczną przerwy międzyblokowej; zaś bit „0" w kierunku przeciwnym do kierunku tej polaryzacji

— dodatkowe zmiany kierunku strumienia magnetycznego doknywane są w środku pomiedzy zmianami strumienia zdefiniowanymi powyżej; zapis i odczyt informacji kodowanej w ten sposób pokazany został na rysunku 1.

Dane zorganizowane są w bloki, które oddzielone są przerwami międzyblokowymi. Przerwa międzyblokowa jest skasowanym odcinkiem taśmy, którego długość zawiera się pomiędzy 17,8 a 250 mm (typ. 20,3 mm). Blok danych składa się z:

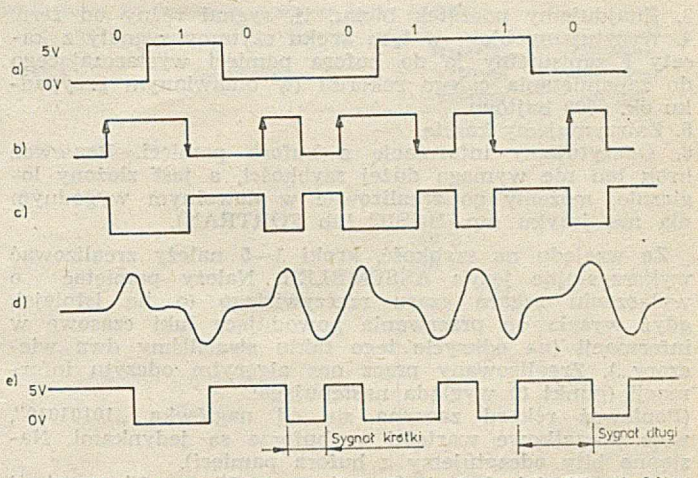
— nagłówka, tzn. ciągu bitów „10101010"

— ciągu bajtów danych; w zasadzie bajt reprezentowany jest przez 8 bitów danych + bit parzystości (nie zawsze występuje); pierwszy bit jest najmniej znaczący, ostatni jest bit parzystości

— dwóch bajtów cyklicznej kontroli nadmiarowej CRC (ang. Cyclic Redundancy Check), ale można się nimi nie przejmować, bo i tak nie będziemy ich dekodować

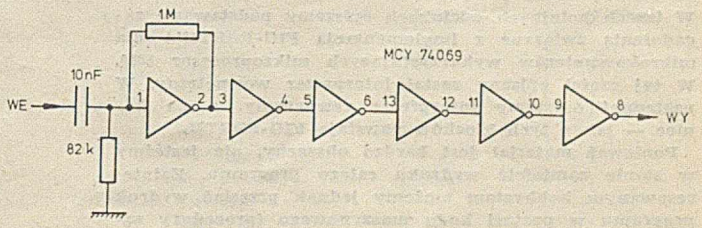
— zakończenia — tego samego ciągu bitów, co w nagłówku.

Dla uproszczenia odczytu zaleca się zapisywanie rekordów o tej samej długości, np. 128 bitów. Rzecz jasna, podany opis jest poważnie uproszczony. Pełny standard zapisu przedstawia norma: ISO/TC97/SC11 Project 6.



Rys. 1. Zapis i odczyt informacji kodowanej metodą PE: a) bity zapisanej informacji, b) informacja zakodowana, c) stan magnetyczny taśmy; d) sygnał z magnetofonu; e) uformowany sygnał odczytu

Tak zapisaną informację można odczytywać za pomocą zwykłego magnetofonu kasetowego. Na wyjściu „głośnik dodatkowy" uzyskujemy sygnał jak na rysunku 1d, z tym że należy uwzględnić różnice szybkości magnetofonu (4,75 cm/s) i stacji PK-1. W pierwszym rzędzie trzeba uzyskać uformowany sygnał odczytu (rys. 1e). Niektóre mikrokomputery, np. COMMODORE-64, VIC-20. NEPTUN-184, kodują informacje zapisywane na kasecie w analogiczny sposób; tak więc możemy czytać kasetę bezpośrednio na magnetofonie komputera. Dla innych mikrokomputerów np. ZX-81, SPECTRUM, TRS-80, należy sygnał uformować i odczytać poprzez port.



Rys. 2. Schemat opisywanego układu

Użyty przez nas układ podany został na rysunku 2. Różni się on od układów używanych w sprzęcie profesjonalnym (jest o wiele prostszy). Wprawdzie daje większą liczbę błędów odczytu (1 błąd na 10000 bajtów, bo w domu włącza się zamrażarka...), ale dla omawianych celów wystarcza w zupełności. W ten sposób uzyskujemy na wejściu komputera ciąg impulsów o poziomach 0 lub 1 i o różnych długościach, które następnie dekodujemy programowo. Jest to przykład zastąpienia odpowiednim oprogramowaniem rozwiązań sprzętowych, często dużo droższych.

Algorytm odczytu

Nie możemy niestety podać konkretnego programu odczytującego informację z kasety, gdyż zależy on od rodzaju komputera i jego oprogramowania (w naszym przypadku TRS-80 + dysk + translator FORTRANU firmy MICROSOFT). Opiszemy natomiast nasze podejście do odczytania kaset PK-1. Podstawowym założeniem było ograniczenie programowania w języku ASSMEBLER.

Odczyt jednego rekordu z kasety można podzielić na następujące kroki:

1. Odczekujemy aż wczytany bit będzie jedynką
2. Znajdujemy przerwę międzyblokową, tj. długi odcinek taśmy zawierający same zera (np. 200 zer).

3. Znajdujemy początek bloku, tj. sygnał różny od zera.
4. Wczytujemy blok; w tym kroku czytamy sygnały z kasety i zapisujemy je do bufora pamięci wystarczającego do zapamiętania całego rekordu (w omawianym przypadku ok. 9000 bajtów).
5. Zatrzymujemy kasetę.
6. Odczytujemy informację z bufora pamięci. Ponieważ krok ten nie wymaga dużej szybkości, a jest złożony logicznie, możemy go zrealizować w dowolnym wygodnym dla nas języku (np. BASIC lub FORTRAN).

Ze względu na szybkość, kroki 1—5 należy zrealizować wykorzystując język ASSEMBLER. Należy pamiętać o wyłączeniu zegara czasu rzeczywistego (o ile istnieje), gdyż wysyła on przerwania powodujące luki czasowe w informacji (na odkrycie tego faktu straciliśmy dwa wieczory...). Zrealizowany przez nas algorytm odczytu informacji (punkt 6) wygląda następująco:
(Ponieważ rekord zaczyna się od nagłówka „10101010", więc początkowe wartości w buforze są jedynkami. Następne bity odczytujemy z bufora pamięci).
a) przeszukaj bufor aż do zmiany z „1" na „0"; przyjmij za wartość wczytaną zero
b) przeskocz ustaloną liczbę miejsc w buforze (wartość ta zależy od szybkości procesora oraz algorytmu wczytywania sygnału z kasety i należy ją dobrać eksperymentalnie; w naszym przypadku wynosiła ona 6): odczytana wartość odpowiada bitowi informacji
c) czytaj znaki z bufora aż do zmiany odczytywnej wartości
d) idź do punktu b.
W ten sposób wczytujemy tyle bitów, ile powinien zawierać rekord.

W naszym przypadku rekordy nie zawierały bitów kontroli parzystości i poprawność odczytu kontrolowaliśmy poprzez dwukrotne odczytanie kasety, a następnie porównanie wyników. Dekodowanie CRC nie było konieczne.

W opisanej metodzie wystarcza minimalna ilość sprzętu, natomiast wymaga ona sporego wysiłku programisty. Zanim uzyskaliśmy końcowe wyniki, napisaliśmy serię programów testowych, rysujących na ekranie monitora odczytane przebiegi sygnału, badające rozrzut długości sygnałów długich i krótkich, co pozwoliło na dobranie odpowiednich stałych czasowych w programach. Wysiłek sowicie się opłacił, gdyż uzyskaliśmy w ten sposób bogate źródło programów, których nigdy byśmy sami nie przepisali.

Na koniec dodajmy, że bez pomocy pp. Jacka Staszelisa i Marka Kałużnego z CAMK-u przedstawiona metoda nie mogłaby być zrealizowana.

ALEKSY E. BARTNIK
MACIEJ GÓRSKI

Poniższa publikacja stała się możliwa dzięki grupie zwolenników języka FORTH, którzy rozpowszechnili metodę zainstalowania języka „FIG-FORTH" (nieco odmiennej wersji standardu FORTH 77), obligując jedynie do zamieszczania powyższej notatki.

W trzech kolejnych odcinkach opiszemy podstawowe zagadnienia związane z implementacją FIG-FORTH'A dla mikrokomputerów wykorzystujących mikroprocesor 8080. W tej części opisany został interpreter wewnętrzny. W następnej opiszemy interpreter zewnętrzny, a na koniec — jak z tych klocków powstaje FIG-FORTH.

Ponieważ materiał jest bardzo obszerny, nie jesteśmy w stanie zamieścić wydruku całego programu. Zainteresowanym hobbystom możemy jednak przesłać wydruk programu w postaci kodu maszynowego (procedury zależne od sprzętu w postaci źródłowej — co ułatwi adaptację programu). W tym celu prosimy o przesłanie do redakcji:
— dużej zaadresowanej koperty ze znaczkiem
— rogu okładki z napisem mikroKLAN 8
— krótkiej informacji o posiadanym sprzęcie mikrokomputerowym

Kod maszynowy będzie przesyłany w wersji wymagającej ciągłego obszaru pamięci RAM od adresu 100_H do 2000_H.

Jak zainstalować FORTH (1)

Interpreter wewnętrzny i stosy języka FORTH

Dla języka FORTH charakterystycznych jest pięć elementów: słownik, stosy (danych i powrotów), interpretery (wewnętrzny i zewnętrzny), asembler oraz pamięć wirtualna. Żaden z tych elementów nie jest unikalną własnością języka FORTH; jednak ich współistnienie powoduje wzajemne potęgowanie działania, tworząc system programowy o niespodziewanej mocy i elastyczności. Omówimy stosy (danych i powrotów) oraz interpreter wewnętrzny.

FORTH korzysta z hierarchicznego systemu procedur. Na samym dole hierarchii znajdują się procedury napisane w kodzie maszynowym. Procedury stojące wyżej nie zawierają kodu maszynowego, lecz składają się z wywołań procedur podrzędnych. Liczba poziomów jest teoretycznie nieograniczona. Dowolny program napisany w języku FORTH jest zamieniany (kompilowany) w procedurę znajdującą się na szczycie pewnej hierarchicznej piramidy wywołań innych procedur. Wywołania procedur muszą się wiązać z zapamiętywaniem adresów powrotu. Ze względu na wielopoziomowość hierarchii, najbardziej naturalnym sposobem organizacji jest stos, nazywany stosem powrotów.

Oprócz stosu powrotów istnieje drugi, zupełnie odrębny — stos danych. Jest on miejscem przechowywania argumentów i wyników operacji wykonywanych przez procedury.

W przypadku wykorzystania mikroprocesora 8080, jeden ze stosów można obsługiwać poprzez użycie rozkazów CALL i RET bądź PUSH i POP. Drugi stos trzeba obsługiwać poprzez bardziej rozbudowane procedury programowe. Ostateczną postać interpretera (wersja e) przedstawimy jako wynik kolejnych kroków optymalizujących rozwiązanie.

W wersji **a** procedura wyższego poziomu składa się z kolejnych wywołań procedur podrzędnych (niekoniecznie najniższego poziomu). Użycie rozkazu CALL w procedurze nadrzędnej i RET w procedurach podrzędnych świadczy o sprzętowej obsłudze stosu powrotów. Wadą rozwiązania jest nieekonomiczne wykorzystanie pamięci; duża część (jedna trzecia) procedury nadrzędnej — kody rozkazów CALL — nie niesie istotnej informacji.

W wersji **b** z procedury wyższego poziomu wyeliminowano kody rozkazów CALL; zostały tylko adresy wywoływanych procedur podrzędnych. Takiego kodu nie może wykonywać procesor; trzeba było dodać interpreter **NEXT**. Używa on zmiennej **ip** (wskaźnik interpretacji — ang. interpretive pointer), wskazującej adresy kolejnych procedur podrzędnych i zwiększane przed każdym wywołaniem procedury o liczbę bajtów zajmowanych przez adres. Wyrażenie **[ip]** oznacza wartość umieszczoną w pamięci pod adresem wskazywanym przez **ip**. Takie rozwiązanie oszczędza pamięć i zwalnia stos sprzętowy. Nie organizuje ono jednak stosu programowego, wskutek czego hierarchiczna struktura procedur ma tylko dwa poziomy: poziom procedur nadrzędnych (będących sekwencjami adresów) i poziom procedur podrzędnych, napisanych w kodzie maszynowym.

Wersja **c** realizuje programowo stos powrotów. Jeśli kolejny element listy tworzącej procedurę nadrzędną jest adresem procedury najniższego poziomu, to interpretacja

przebiega analogicznie jak w poprzednim przypadku. Jeśli jednak element listy jest adresem procedury będącej także listą, to następuje skok do rozkazu **jmp DOCOL**, umieszczonego na początku tej listy, i wykonanie procedury **DOCOL**. Procedura ta realizuje programową obsługę stosu powrotów, umieszczając bieżącą wartość wskaźnika interpretacji w miejscu pamięci wskazywanym przez zmienną **rsp** (wskaźnik stosu powrotów — ang. return stack pointer). Odzyskanie ze stosu powrotów wskaźnika interpretacji następuje po obsłudze ostatniego adresu listy przez wykonanie procedury SEMIS. Taki sposób organizacji listy nosi nazwę bezpośredniego kodu nizanego (ang. direct threaded code).

Wersja **d** jest realizacją pośredniego kodu nizanego (ang. indirect threaded code). Wyeliminowano tu kod instrukcji **jmp**, pozostawiając tylko adres procedury **DOCOL**. Podobnie w procedurach najniższego poziomu przed pierwszą instrukcją maszynową umieszczono adres wskazujący tę instrukcję. Interpreter NEXT musi teraz zapewnić przekazanie sterowania nie na początek procedury, lecz pod adres umieszczony na początku procedury. Wskaźnik interpretacji **ip** wskazuje więc kolejny adres z listy, zaś zawartość spod tego adresu, po umieszczeniu w rejestrze **w**, wskazuje na początkowy adres umieszczony w procedurze podrzędnej, czyli adres procedury DOCOL albo adres pierwszego rozkazu procedury najniższego poziomu.

Interpreter NEXT może też działać z post-inkrementacją (wersja **e**). Nastąpiła tu zamiana kolejności zapisania rejestru **w** i zwiększenia wskaźnika **ip**. Przy wykonywaniu procedury podrzędnej, wskaźnik interpretacji przechowywany na stosie wskazuje już następną procedurę podrzędną.

```
;   NEXT - INTERPRETER ADRESOW FORTHA
;          (WERSJA Z POSTINKREMENTACJA)
;
;       REJESTRY:
;       FORTH       8080
;       -----       ----
;        IP         BC
;        W          DE
;        SP         SP
;
ORG     0143H
DPUSH:  PUSH        D       ;***STOS <- DE***
HPUSH:  PUSH        H       ;***STOS <- HL***
NEXT:   LDAX        B       ;***DE <- (BC) +1 BC <- BC+2
                            ; JMP DE***
        INX         B
        MOV         L,A
        LDAX        B       ; HL <- (BC)
        INX         B       ; BC <- BC+2
        MOV         H,A
        MOV         E,M
        INX         H
        MOV         D,M     ; DE <- (HL), HL <- HL+1
        XCHG                ; DE <-> HL
        PCHL                ; JMP HL
;
ORG     0128H
RPP:    DW          0       ; WSKAZNIK STOSU POWROTOW
;
ORG     0447H
SEMIS:  DW          $+2     ; ***BC <- RSTOS***
        LHLD        RPP
        MOV         C,M
        INX         H
        MOV         B,M
        INX         H       ; BC <- (RPP)
        SHLD        RPP     ; RPP <- RPP+2
        JMP         NEXT
;
ORG     0611H
DOCOL:  LHLD        RPP     ; ***RSTOS <- BC, DE <- DE+1
                            ;      BC <- DE+1***
        DCX         H
        MOV         M,B
        DCX         H
        MOV         M,C     ; (RPP) <- BC
        SHLD        RPP     ; RPP <- RPP-2
        INX         D       ; DE <- DE+1
        MOV         C,E     ; BC <- DE
        MOV         B,D
        JMP         NEXT
```

Wydruk 1. Realizacja interpretera wewnętrznego języka FORTH dla 8080

Na wydruku 1 przedstawiono sposób realizacji interpretera wewnętrznego (NEXT, SEMIS i DOCOL) dla mikroprocesora 8080.

Optymalizacja interpretera wewnętrznego języka FORTH

Wersja	Interpreter	Procedura wyższego poziomu	Procedura najniższego poziomu
a	procesor	. . . call A call B call C . . .	A: . . . . return
b	NEXT: ip←ip+2 jmp [ip]	. . . A B C	A: . . . jmp NEXT
c	NEXT: ip←ip+2 jmp [ip] SEMIS: ip←[rsp] rsp←rsp+2 jmp NEXT DOCOL: rsp←rsp−2 [rsp]←ip ip←[ip]+2 jmp NEXT	jmp DOCOL A B C . . . SEMIS	A: . . . jmp NEXT
d	NEXT: ip←ip+2 w←[ip] jmp [w] SEMIS: $+2 ip←[rsp] rsp←rsp+2 jmp NEXT DOCOL: rsp←rsp−2 [rsp]←ip ip←w+2 jmp NEXT	DOCOL A B C . . . SEMIS	A: $+2 . . . jmp NEXT
e	NEXT: w←[ip] ip←ip+2 jmp[w] SEMIS: $+2 ip←[rsp] rsp←rsp+2 jmp NEXT DOCOL: rsp←rsp−2 [rsp]←ip ip←w+2 jmp NEXT	DOCOL A B C . . . SEMIS	A: $+2 . . . jmp NEXT

Komunikację operatora z systemem (poprzez konsolę i drukarkę) organizują dwie procedury najniższego poziomu — KEY i PEMIT (wydruk 2). Wywołują one procedury CIN, COUT i POUT — wymieniające znaki między rejestrami procesora a konsolą i drukarką. Etykietom CIN, COUT i POUT należy przyporządkować adresy wynikające z konkretnej realizacji wykorzystywanego systemu mikroprocesorowego.

```
CIN     EQU     OF803H  ; ADR. PROC. WCZYT. ZNAK DO A
                        ; Z KONSOLI
COUT    EQU     OF809H  ; ADR. PROC. WYSYL. ZNAK Z C
                        ; NA KONSOLE
POUT    EQU     OF80FH  ; ADR. PROC. WYSYL. ZNAK Z C
                        ; NA DRUKARKE
;
ORG     1643H
EPRINT: DW      0       ; ZMIENNA STERUJACA DRUKARKA
                        ; 0=WYLACZONA 1=WLACZONA
;
ORG     02F2H
KEY:    DW      $+2     ; ***CZYTAJ ZNAK Z KONSOLI NA
                        ;     STOS. JEZELI CTRL P,
                        ;     ZMIEN STAN DRUKARKI***
        JMP     PKEY
ORG     1686H
PKEY:   CALL    CIN     ; WCZYTAJ DO A ZNAK Z KONSOLI
        CPI     10H     ; GDY ZNAK = CTRL P
        MOV     E,A     ; E <- ZNAK
        JNZ     PKEY1
        LXI     H,EPRINT;
        MVI     E,20H   ; E <- SPACJA
        MOV     A,M
        XRI     1
        MOV     M,A     ; ZMIEN STAN DRUKARKI
PKEY1:  MOV     L,E     ;
        MVI     H,0     ; HL <- 0 I ZNAK
        JMP     HPUSH   ; ZNAK NA STOS, POTEM NEXT
;
ORG     169EH
PEMIT:  DW      $+2     ; ***WYSLIJ ZNAK ZE STOSU NA
                        ;     KONSOLE***
        POP     H       ; L <- KOD ZNAKU ZE STOSU
        PUSH    B       ; CHRON IP
        MOV     C,L
        CALL    CPOUT   ; WYSLIJ NA KONS. ZNAK Z C
        POP     B       ; ODZYSKAJ IP
        JMP     NEXT
ORG     1667H
CPOUT:  CALL    COUT    ; ***WYSLIJ ZNAK Z C NA KONSOLE
                        ;     I BYC MOZE NA DRUKARKE***
        XCHG
        LXI     H,EPRINT
        MOV     A,M     ; GDY (EPRINT) <> 0
        ORA     A
        JZ      CPOU1
        MOV     C,E
        CALL    POUT    ; WYSLIJ ZNAK Z C NA DRK.
CPOU1:  RET
```

Wydruk 2. Realizacja procedur KEY i PEMIT

Na wydruku 3 przedstawiono następnych pięć procedur najniższego poziomu.

```
ORG 0156H
LIT:    DW      $+2     ; ***STOS <- (BC), BC <- BC+2***
        LDAX    B
        INX     B
        MOV     L,A
        LDAX    B
        INX     B       ; HL <-<- (BC)
        MOV     H,A     ; BC <- BC+2
        JMP     HPUSH   ; STOS <- HL
;
ORG     017AH
BRAN:   DW      $+2     ; ***BC <- BC + (BC)***
BRAN1:  MOV     H,B
        MOV     L,C     ; HL <- BC
        MOV     E,M
        INX     H
        MOV     D,M
        DCX     H       ; DE <- (HL)
        DAD     D       ; HL <- HL+DE
        MOV     C,L
        MOV     B,H     ; BC <- HL
        JMP     NEXT
;
ORG     04C9H
PLUS:   DW      $+2     ; ***STOS + STOS -> STOS***
        POP     D
        POP     H
        DAD     D
        JMP     HPUSH
;
ORG     0549H
SWAP:   DW      $+2     ; ***ZAMIEN MIEJSCAMI DWA
                        ;     OSTATNIE NA STOSIE***
        POP     H
        XTHL
        JMP     HPUSH
;
ORG     0556H
DUP:    DW      $+2     ; ***POWIEL OSTATNI NA STOSIE***
        POP     H
        PUSH    H
        JMP     HPUSH
```

Wydruk 3. Realizacja niektórych słów języka FORTH jako procedur na najniższym poziomie

KEY — czeka na wciśnięcie klawisza, po czym umieszcza na stosie 16-bitową liczbę, której bardziej znaczący bajt jest zerem, zaś mniej znaczący bajt — kodem wciśniętego klawisza. W przypadku jednoczesnego wciśnięcia klawiszy CTRL i P zmienia stan podłączenia drukarki.

PEMIT — zdejmuje ze stosu 16-bitową liczbę i wysyła mniej znaczący bajt (znak ASCII) na konsolę (i ewentualnie na drukarkę, jeśli jest przydzielona).

LIT — umieszcza na stosie 2-bajtową liczbę wskazywaną przez wskaźnik interpretacji, czyli znajdującą się na liście bezpośrednio za odwołaniem do LIT, po czym zwiększa wskaźnik interpretacji o 2, aby wskazywał kolejny adres na liście.

BRAN (BRANCH) — do wskaźnika interpretacji dodaje 2-bajtową liczbę (dodatnią lub ujemną) wskazywaną przez niego, czyli znajdującą się na liście bezpośrednio za odwołaniem do BRANCH; tym samym realizuje skok.

PLUS — zdejmuje ze stosu dwie 16-bitowe liczby, dodaje je arytmetycznie i wynik umieszcza na stosie.

SWAP — zamienia miejscami dwie 16-bitowe liczby na szczycie stosu.

DUP — powiela na stosie liczbę znajdującą się na jego szczycie.

```
ORG     00060H
CRLF:                   ; ***WYSLIJ CR I LF***
        DW      DOCOL   ; STOS PUSTY
        DW      LIT
        DW      0000DH  ; STOS: 000D
        DW      PEMIT   ; WYSLIJ 0D - STOS PUSTY
        DW      LIT
        DW      0000AH  ; STOS: 000A
        DW      PEMIT   ; WYSLIJ 0A - STOS PUSTY
        DW      SEMIS
;
LOOP:                   ; ***PETLA NIESKONCZONA***
                        ; WYGLAD STOSU
        DW      CRLF    ; -
        DW      KEY     ; KODN
        DW      DUP     ; KODN KODN
        DW      LIT
        DW      1       ; KODN KODN 0001
        DW      PLUS    ; KODN KODN+1
        DW      SWAP    ; KODN+1 KODN
        DW      DUP     ; KODN+1 KODN KODN
        DW      LIT
        DW      -1      ; KODN+1 KODN KODN FFFF
        DW      PLUS    ; KODN+1 KODN KODN-1
        DW      CRLF
        DW      PEMIT   ; KODN+1 KODN
        DW      CRLF
        DW      PEMIT   ; KODN+1
        DW      CRLF
        DW      PEMIT   ; -
        DW      BRAN    ;   SKOCZ NA POCZATEK PETLI
        DW      -36     ;   CZYLI ZMNIEJSZ IP O 36D
;
ORG 00050H
INIT:   LXI     SP,3B00H        ; INICJUJ STOS DANYCH
        LXI     H,3BA0H
        SHLD    RPP             ; INICJUJ STOS POWROTOW
        LXI     B,LOOP          ; INICJUJ WSK. INTERPR.
        JMP     NEXT            ; ROZPOCZNIJ INTERPRET.
```

Wydruk 4. Program sprawdzający

Proponujemy sprawdzenie działania interpretera wewnętrznego poprzez uruchomienie programu LOOP. Jest on napisany jako procedura wyższego poziomu (lista adresów) i działa w nieskończonej pętli. W czasie jednego przebiegu pętli program czeka na wciśnięcie klawisza z konsoli, po czym wysyła na konsolę w trzech kolejnych linijkach trzy znaki ASCII, których kody różnią się o 1, zaś środkowy jest znakiem wciśniętym na klawiaturze. Program wywołuje zarówno procedury najniższego poziomu, jak i procedurę wyższego poziomu (listę) — CRLF.

Cały FORTH (przedstawiono tylko jego fragment) potrzebuje pamięci o adresach od **0100H** do **3FFFH**, czyli ok. 16 KB. Przeprowadzając relokację programu można go umieścić i w innym obszarze pamięci. Wykonanie należy rozpocząć poprzez skok do procedury inicjującej INIT.

ZBIGNIEW POJMAŃSKI
PIE — Warszawa

Toniemy w powodzi informacji. Z odsieczą przychodzi jednak nowoczesna technika. Wystarczy mikrokomputer i zręcznie zastosowana baza danych, aby główny księgowy nie musiał odchorowywać (lub co gorsza odsiadywać) każdego bilansu, aby magazynier naprawdę wiedział co posiada, a szary petent nie przeklinał biurokraty, który za każdym razem żąda dziesiątek zaświadczeń — bo nie jest w stanie odszukać dokumentów sprzed miesiąca. Prawdę mówiąc, nie potrafię sobie wyobrazić firmy, w której nie istniałaby potrzeba zastosowania tego typu oprogramowania! I to nie dlatego, żeby pani Iksińska mogła zbijać bąki — gdy komputer za nią pracuje, lecz by móc wykorzystać posiadane informacje w tak efektywny sposób, że nawet pięć zaharowanych pań Iksińskich nie byłoby w stanie tego przygotować. To może trudne do uwierzenia, ale opisywane narzędzie jest dostępne u nas w kraju — za złotówki!

BANK DANYCH-CSK

Bank Danych — CSK to system zarządzenia relacyjną bazą danych opracowany przez Computer Studio Kajkowski. Może on zostać wykorzystany dla mikrokomputerów zbudowanych na bazie mikroprocesorów 8-bitowych 8080 i Z80 lub 16-bitowego 8086. W oparciu o Bank Danych — CSK możliwe jest tworzenie różnych systemów użytkowych, takich jak:
— systemy finansowo-księgowe
— systemy płacowe
— systemy osobowe, bibliograficzne itp.
— systemy magazynowe
— systemy biurowe.

Wymagania sprzętowe

Mikrokomputer, na którym implementowany jest Bank Danych — CSK musi być wyposażony w system operacyjny CP/M (dla mikrokomputera 16-bitowego — CP/M-86) lub system z nim kompatybilny, a ponadto — dysponować przynajmniej 48 KB pamięci operacyjnej. W skład wymaganej minimalnej konfiguracji sprzętowej wchodzi: monitor alfanumeryczny i klawiatura, co najmniej jedna jednostka pamięci dyskowej oraz drukarka. System może pracować m.in. na takich mikrokomputerach, jak: ELWRO 513, IMP 85, RTDS-8, LIDIA oraz ROBOTRON 5110 i 5120.

Właściwości użytkowe systemu

System składa się z pakietu programów zapisanych na dyskietce w kodzie maszynowym mikroprocesora 8080 (8086). Ma on interpreter własnego języka manipulacji plikami (ponad 60 instrukcji i funkcji). Język ten umożliwia działania na plikach, wykonywanie obliczeń numerycznych, utrzymanie plików danych, tworzenie raportów, zestawień itp. Instrukcje tego języka mogą być wykonywane zarówno w trybie konwersacyjnym, jak i programowym. Wprowadzanie i wyprowadzanie danych może odbywać się w dostosowanej do potrzeb użytkownika postaci graficznej.

W systemie zastosowano relacyjny model bazy danych, w którym pliki składają się z rekordów o jednakowej — definiowanej przez użytkownika — długości i strukturze. Rozwiązanie takie wybrano ze względu na prosty i wygodny dla użytkownika sposób opisu zbiorów.

Wielkość plików danych, przetwarzanych w systemie, ograniczona jest następującymi parametrami:
— liczba rekordów w jednym pliku nie przekracza 65 535
— liczba znaków w rekordzie nie przekracza 1000
— liczba pól w rekordzie nie przekracza 32
— liczba znaków w polu nie przekracza 254
— liczba znaków klucza indeksowego nie przekracza 100.
Pola rekordu mogą być typu alfanumerycznego, numerycznego lub logicznego.

Zakładanie plików danych

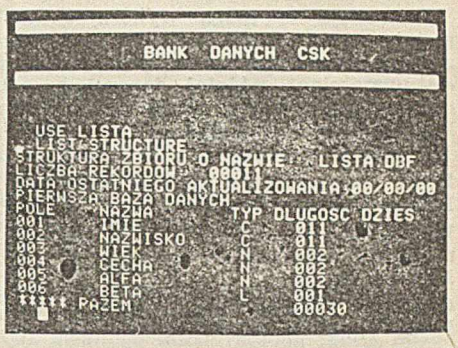
Podstawową instrukcją umożliwiającą zakładanie plików danych jest **CREATE.** Po wprowadzeniu tej instrukcji, użytkownik podaje dalej w trybie konwersacyjnym (odpowiadając na pytania) nazwę tworzonego pliku i strukturę rekordu. Struktura rekordu definiowana jest przez podanie opisów kolejnych pól, składających się z nazwy pola, typu i długości (wyrażonej liczbą znaków alfanumerycznych). W przypadku pola numerycznego możliwe jest zadeklarowanie liczby miejsc po przecinku dziesiętnym (wtedy długość wyraża sumaryczny rozmiar pola wraz z pozycją przecinka). Definiowanie struktury rekordu kończy się po wciśnięciu w odpowiedzi na kolejne pytanie o nazwę pola — klawisza CR.

Opis struktury pliku zapisywany jest na dyskietce i można go odtworzyć podając następujące instrukcje:

USE <nazwa pliku>

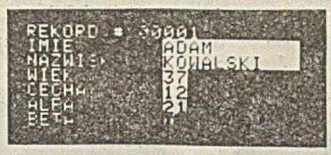
LIST STRUCTURE

USE zapewnia otwarcie dostępu do pliku o podanej nazwie, a LIST STRUCTURE powoduje wyświetlenie struktury rekordu danego pliku. Przykład działania tych instrukcji pokazano na zdjęciu 1. Plik o nazwie LISTA liczy jedenaście rekordów, z których każdy zawiera sześć pól o nazwach: IMIE, NAZWISKO, WIEK, CECHA, ALFA, BETA. Początkowe dwa pola są typu alfanumerycznego i mają długość jedenastu znaków. Następne trzy — to pola numeryczne o długości dwóch znaków; ostatnie jest polem zawierającym wartości logiczne o długości jednego znaku.



Fot. 1. Opis struktury przykładowego pliku danych

Po zakończeniu definiowania struktury pliku, użytkownik musi odpowiedzieć na pytanie, czy chce od razu wprowadzać dane. Gdy podana zostanie odpowiedź twierdząca (tak) — system wyświetla obraz pustego rekordu w standardowej postaci, oczekując na wprowadzenie danych. Na zdjęciu 2 pokazano przykładowy obraz z monitora po wprowadzeniu danych.



Fot. 2. Standardowa postać rekordu generowana przez system

Wciśnięcie klawisza CR spowoduje teraz przejście do wypełniania danymi następnego rekordu. Wprowadzanie danych kończy się wciśnięciem klawisza CR bezpośrednio po wyświetleniu kolejnego pustego rekordu. Utworzony plik zostaje zapisany w pamięci zewnętrznej.

Zakładanie plików danych można także realizować za pomocą instrukcji:

COPY — umożliwia kopiowanie całych plików lub wybranych rekordów spełniających określony warunek; kopiować można rekordy lub wybrane pola

JOIN — łączy dwa pliki danych, tworząc trzeci; również w tym przypadku do nowo tworzonego pliku można przenosić określone rekordy i pola rekordów łączonych plików.

Zakładając pliki danych, można korzystać z plików utworzonych za pomocą innych systemów lub języków programowania. Umożliwiają to odpowiednie opcje instrukcji tworzących pliki.

Aktualizacja plików danych

Dopisywanie danych do plików realizują instrukcje:

APPEND — dopisuje nowe rekordy na koniec pliku

INSERT — wpisuje nowe rekordy w dowolne miejsce pliku.

Instrukcje te mogą wykorzystywać standardową postać rekordu (jak na zdjęciu 2) lub dopisywać puste rekordy wypełniane później danymi przez odpowiedni program użytkownika.
Poprawianie i aktualizację założonych już plików umożliwiają instrukcje:

EDIT — wprowadza zmiany do określonego rekordu

BROWSE — poprawia na ekranie monitora jednocześnie kilkanaście rekordów

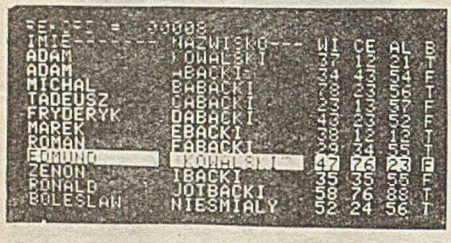
CHANGE — zmienia zawartość określonego pola w rekordzie

DELETE — zaznacza określone rekordy jako skreślone

PACK — fizycznie usuwa z pliku skreślone rekordy

UPDATE — aktualizuje plik danych na podstawie pliku zmian zapisanego w pamięci zewnętrznej.

Instrukcje EDIT i BROWSE wykorzystują możliwości edycji pełnoekranowej. Za pomocą odpowiednich klawiszy można sterować położeniem kursora i umieszczać go w dowolnym miejscu ekranu. Podobnie można zmieniać numer poprawianego rekordu. Dzięki temu, użytkownik może szybko przeglądać dane i redagować je w dowolnej kolejności.



Fot. 3. Działanie instrukcji BROWSE

Na zdjęciu 3 pokazano działanie instrukcji BROWSE. Kursor umieszczono na pozycji rekordu nr 8, który przez to wyświetlony został w odmienny sposób. Umieszczając teraz kursor na żądanym znaku można dokonać stosownych zmian.

Wybrane możliwości systemu

Oprócz wspomnianych wcześniej możliwości, system ma wiele innych instrukcji. Są to m.in.:

SORT — sortowanie pliku według określonego pola danych

INDEX — indeksowanie pliku danych

TOTAL — scalanie plików danych poprzez sumowanie rekordów o tej samej zawartości określonych pól

LOCATE — wyszukiwanie rekordów spełniających podany warunek logiczny

FIND — szybkie odnajdywanie rekordu na podstawie klucza indeksowego

LIST — drukowanie plików danych (lub ich części spełniających określony warunek logiczny) w postaci całych rekordów lub wybranych pól

REPORT — szybkie tworzenie standardowych zestawień z możliwością umieszczania w nich nie tylko danych z pliku, lecz również wyników obliczeń

COUNT — obliczanie liczby rekordów spełniających podany warunek

SUM — sumowanie wybranych pól danych rekordów spełniających określony warunek

SET ALTERNATION — tworzenie dziennika dostępu do plików.

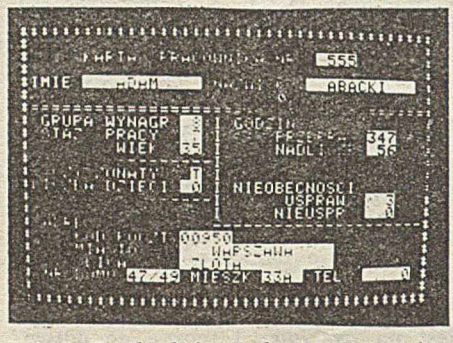
Podane instrukcje można wykonywać zarówno w trybie konwersacyjnym, jak i programowym.

Tworzenie systemów użytkowych

Bogaty repertuar instrukcji Banku Danych — CSK umożliwia tworzenie systemów użytkowych. Mogą być one pisane za pomocą dowolnego systemu redagowania tekstów. Można również do tego celu wykorzystać instrukcję MODIFY COMMAND, która w istocie dostarcza możliwości prostego systemu tego typu.

Programy użytkowe mogą korzystać ze zmiennych i stałych oraz kilkunastu funkcji numerycznych, logicznych i działających na ciągach znaków. Obliczenia numeryczne przeprowadzane są z dokładnością do dziesięciu cyfr.

Wśród instrukcji umożliwiających tworzenie programów, oprócz instrukcji warunkowych typu IF...ELSE czy organizujących pętle programowe DO WHILE..., znaleźć można takie, które pozwalają wprowadzać i wyprowadzać dane w dogodnej dla użytkownika postaci. Można zatem korzystać ze standardowej postaci rekordu, generowanej przez system, lub też przedstawić ją w innej formie na ekranie monitora — np. jak na zdjęciu 4.



Fot. 4. Przykład innej formy wprowadzania danych

Pola danych dla większej czytelności wyświetlane są w odmienny sposób od komunikatów. Wprowadzane dane mogą być na bieżąco kontrolowane przez program użytkowy. Podobnie, do wyprowadzania danych można użyć instrukcji LIST, REPORT lub też — używając innych dostępnych instrukcji — zaprojektować własną formę wydruku.

Tworząc konkretny system użytkowy, można go zrealizować w takiej postaci, że korzystanie z niego nie będzie wymagało umiejętności programowania (może być obsługiwany przez osoby nie przygotowane zawodowo do pracy z komputerem) i będzie on odporny na błędy użytkownika.

Przedstawiony system cechuje prostota opisu danych i duża uniwersalność zastosowanego języka. Walory użytkowe prezentowanego systemu zwiększa dodatkowo możliwość jego współpracy z innymi pakietami programów o szerokiej gamie zastosowań, np. TABPLAN-CSK — systemem kalkulacyjno-planistycznym lub TEKST-CSK — systemem redagowania tekstów.

MACIEJ GILARSKI
Computer Studio Kajkowski

- Ogłoszony przez ABAKUS pierwszy w Polsce konkurs na grę edukacyjną dobiegł końca. Zgłoszono 25 programów. Pierwszą nagrodę (ZX SPECTRUM 48K), zdobył Wojciech Dąbrowski za program „Rysowanie" (zabawa dla dzieci kilku- i kilkunastoletnich, ucząca myślenia algorytmicznego przy rysowaniu).
  Przyznano także trzy wyróżnienia:
  — Krzysztofowi Konieczce za program „GEO" (odgadywanie położenia geograficznego miast polskich na mapie)
  — Barbarze Chrzan-Feluch (poprzednio laureatka w konkursie na pomysł) za program „Kostka wielowymiarowa" (wyrabiający orientację w przestrzeniach wielowymiarowych)
  — Markowi Wojciechowskiemu za program „Nauka żeglowania".
  Jury odrzuciło wszystkie programy napisane na MERĘ — jako że żaden z autorów nie spełnił warunku dostarczenia komputera dla demonstracji działania programu.
  Konkurs obnażył... brak werwy w pisaniu i nadsyłaniu programów, także u ludzi z fantazją. Kilka niezłych pomysłów nie zostało dopracowanych; i to zarówno koncepcyjnie, jak i pod względem programowym. Uwaga ta odnosi się również do laureatów.
- PPZ AMEPROD wydało drugi numer „ZX MIKRO". Trochę o sprzęcie, trochę programów — w sumie godne uwagi. Cena 250 zł, dość wysoka dla prywatnego nabywcy, wynika z kosztów druku i opracowania.
- Uwaga właściciele COMMODORE C64. Bartosz Pastuszak (66-400 Gorzów Wielkopolski, ul. Marcinkowskiego 10g m. 3, tel. 26-401 lub 25-151) chętnie wymieni doświadczenia z inymi posiadaczami tego mikrokomputera.
- Nadsyłając przez najbliższy miesiąc pod adresem INFORMATYKI: zaadresowaną do siebie dużą kopertę ze znaczkiem, róg odkładki pisma z napisem „mikroKLAN 8", ewentualną ocenę dotychczasowych publikacji o ZX SPECTRUM — można otrzymać kserokopię artykułu (w języku angielskim) zawierającego procedury w kodzie maszynowym, ułatwiające wykorzystanie możliwości graficznych ZX SPECTRUM.

— prowadzi
Andrzej J. Pietrowski
tel. dom. 48-22-85

I/B. Indeksowanie danych i przeszukiwanie zbiorów

Niniejszy zestaw pytań [1] nie stanowi jakiegokolwiek standardu egzaminacyjnego i dlatego stosowanie go do sprawdzania wiedzy innej osoby — poza samym sobą — stanowi świadome pogwałcenie intencji ACM i zastrzeżonych praw autorskich tego stowarzyszenia.

PYTANIA

12. Dany jest zbiór dziewięciu elementów uporządkowanych następująco w pamięci: 6, 32, 3, 7, 11, 4, 14, 3, 2. Mamy do dyspozycji program sekwencyjnego przeszukiwania w przód (ang. forward sequential search). W jakiej kolejności zostanie zlokalizowany element „4"?

    a) 2, 3, 3, 4
    b) 6, 32, 3, 7, 11, 4
    c) 11, 7, 3, 4
    d) 2, 3, 14, 4

13. Dana jest tablica:

Indeksy:	1	4	7	8	12	13	15
Wartości:	M	A	B	X	K	J	R

    Załóżmy, że pewien algorytm przeszukujący doszedł do wartości B w kolejności indeksów: 8, 4, 7. Jaka byłaby najpoprawniejsza nazwa tego algorytmu?

    a) dwuliniowy (ang. bi-linear search)
    b) z kodowaniem mieszającym (ang. hashed search)
    c) indeksowo-sekwencyjny (ang. index-sequential search)
    d) binarny (ang. binary search)

14. Jaki jest najwłaściwszy obszar zastosowań metody przeszukiwania binarnego (zwanej niekiedy dychotomiczną lub metodą podziału artyleryjskiego)?

    a) elementy uporządkowane na nośniku sekwencyjnym
    b) elementy uporządkowane na nośniku bezpośrednim
    c) elementy nie uporządkowane na nośniku bezpośrednim
    d) elementy nie uporządkowane na nośniku sekwencyjnym

15. Rozważmy zbiór 1024 rekordów zapisanych w kolejności wartości kluczy. Załóżmy, że prawdopodobieństwo poszukiwania jest jednakowe dla każdego rekordu. Oszacujmy na 100 μs czas porównywania klucza badanego z żądanym. Jakiemu skróceniu — przy zastąpieniu przeszukiwania sekwencyjnego binarnym — ulegnie przeciętny czas lokalizacji dowolnego rekordu w tym zbiorze?

    a) 50 ms
    b) 30 ms
    c) 10 ms
    d) 1 ms

16. Dla struktur drzewiastych metoda drzew zbilansowanych (ang. balanced tree) jest...

    a) ...szczególnie dogodna przy niewielkiej liczbie elementów, rzędu kilkuset
    b) ...dogodna do reprezentowania list jednokierunkowo liniowych o długości N, gdyż z dużym stopniem efektywności realizowane są tutaj operacje: lokalizacji elementu o danym kluczu, lokalizacji k-tego elementu (przy zadanym k), wstawiania elementu na określone miejsce oraz usuwania określonego elementu
    c) ...inną nazwą metody drzew binarnych
    d) ... — mówiąc ogólnie — bardziej odpowiednia przy zewnętrznym niż przy wewnętrznym przechowywaniu danych

17. Które z poniższych sformułowań najtrafniej oddaje ideę przeszukiwania metodą kodowania mieszającego (ang. hashed search), zwanego także metodą tablic mieszanych (ang. hash-coding table search)?

    a) znajduje ona zastosowanie wyłącznie do jednosłowowych łańcuchów znakowych
    b) czas lokalizacji rośnie wraz z rozmiarami tablicy
    c) próbka inicjująca przeszukiwanie jest funkcją lokalizowanego elementu
    d) czas lokalizacji jest niezależny od liczby aktywnych elementów tablicy

18. Dana jest macierz zmiany stanów pewnego automatu, który sprawdza dane wejściowe:

Aktualny stan wewnętrzny automatu	Nowy stan po identyfikacji znaku				
	„pusty"	cyfra	+lub−	kropka	inny
(1) „pusty"	1	2	3	4	0
(2) liczba całkowita	1	2	0	4	0
(3) znak arytmetyczny	0	2	0	0	0
(4) kropka dziesiętna	1	0	0	0	0

    Załóżmy, że sprawdzanie zaczyna się w stanie wewnętrznym "1" (pusty). Który z poniższych ciągów znaków wejściowych nie zostanie zaakceptowany — tzn. spowoduje przejście tego automatu w stan "0"?

    a) 18, b) 9, c) 12.2, d) +0010

19. Jeżeli dane są zarejestrowane w postaci listy jednokierunkowej (ang. linearly linked list) to...

    a) ...wszystkie one muszą zawierać się w spójnym obszarze pamięciowym
    b) ...prócz obszaru pamięciowego na przechowywanie danych potrzebny jest także obszar na przechowywanie informacji o powiązaniach między tymi danymi
    c) ...wszystkie one muszą być zapamiętane znakowo, gdyż w listach jednokierunkowych mogą występować jedynie ciągi znaków
    d) ...żaden ich nowy element nie może zostać włączony do struktury takiej listy bez założenia nowej listy i unieważnienia dotychczasowej

20. Załóżmy, że duża tablica dwuwymiarowa, zajmująca kilkanaście stron pamięci wirtualnej, jest zapisana wierszami, przy czym prawdopodobieństwa poszukiwania są dla każdego elementu tej tablicy jednakowe. W którym z podanych poniżej przypadków zastosowanie metody przeszukiwania sekwencyjnego będzie najefektywniejsze;

    a) gdy wskaźnik pierwszego wiersza będzie zmieniać się szybciej niż wskaźnik drugiego wiersza
    b) gdy wskaźnik drugiej kolumny będzie zmieniać się szybciej niż wskaźnik pierwszej kolumny
    c) gdy oba wskaźniki będą się zmieniać jednakowo szybko
    d) gdy oba wskaźniki będą się zmieniać w jakiś inny sposób

[1] Communications of the ACM, vol. 19, (no. 5, May 1976 pp. 231. Stowarzyszenie A.C.M. zezwoliło na przedruk Self-Assessment Procedure I (Copyright 1976 by Association for Computing Machinery) do celów niekomercyjnych

ROZWIĄZANIA

12-b, 13-d, 14-b, 15-a, 16-b, 17-c, 18-c, 19-b, 20-b.

LITERATURA

ad 12. Knuth D.: The Art of Computer Programming. Vol. 3 Addison-Wesley, p. 393—405
ad 13. ibid. p. 406—414
ad 14. ibid. p. 406—407
ad 15. ibid. p. 110—111
ad 16. ibid. p. 451—471
ad 17. Maurer W., Lewis T.: Hash Table Methods. Computing Surveys, vol. 7 (No. 1, March 1975), p. 5—19
ad 18. Weinberg, Yasukawa, Markus: Structured Programming in PL/C: An Abecedarian. Wiley, p. 84—85
ad 19. Booth T., Chien Y.: Computing: Fundamentals and Applications. Wiley/Hamilton, p. 237—317
ad. 20. Hellerman H., Conroy T.: Computer System Performance. McGraw-Hill, p. 299—300

Elsdoff J.: Some Programming Techniques for Processing Multi-Dimensional Matrices in a Paging Environment. Proceedings NCC '74, vol. 43, p. 185—193.

* * *

Zdaniem testodawców — ACM Committee on Self-Assessment — osoba zainteresowana niniejszym samotestem może uznać go za bezwzględnie przydatny dla siebie o ile tylko:

— uświadomi sobie istnienie pewnych pojęć, uprzednio nie znanych lub niezbyt dobrze rozumianych
— rozszerzy swą wiedzę o głębsze rozumienie pojęć istotnych dla wykonywanej pracy lub osobistych zainteresowań.

Adaptowali z angielskiego:
ADAM B. EMPACHER
LUDWIK J. ROSSOWSKI

Z KRAJU

Szkoły Mikroprocesorowe

Refleksje słuchacza

Świat rzucił nam mikroelektroniczne wyzwanie. Czy potrafimy sprostać wymaganiom współczesności przygnieceni przez nasze krajowe problemy. Czy znajdziemy wśród nas dość osób rozumiejących nie tylko techniczny, ale i społeczny wymiar mikrokomputerowej walki o nasze prawo do wejścia w XXI wiek?

To nie pusta retoryka. To także pytanie o rację bytu i sens pięciu dotychczasowych Szkół Mikroprocesorowych. Czy są to spotkania służące rozwojowi uczestników, będące jednocześnie naszym wkładem w społeczny postęp? Czy to tylko rozpaczliwa obrona przed dalszym pogrążeniem się w zaścianku?

Trudne to pytania i wydawać się może, że zasięgiem swym przekraczają kilkudniowe spotkania nielicznej grupy mikroinformatyków. Sądzę jednak, że Szkoły Mikroprocesorowe są zjawiskiem znaczącym. Jerzy Dańda, twórca idei Szkół i animator pierwszej ich piątki, przypomniał niedawno [1]) początki tych spotkań oraz założenia programowe, jakie przyjęli organizatorzy. Ja zaś, zastanawiając się nad celem Szkół, dostrzegam kilka jego wymiarów:

- **dokumentalny:** przedstawienie konkretnych prac; nawet jeżeli ich wyniki nie zostaną opublikowane, to dzięki Szkole będą znane większej liczbie osób
- **organizacyjny:** Szkoła ułatwia wzajemne poznanie się i sprzyja dalszym kontaktom osobistym
- **informacyjny:** przekazywane są informacje o prowadzonych pracach, podaje się źródła różnych wyrobów i usług, wymienia doświadczenia zawodowe
- **edukacyjny:** możliwość nauczenia się czegoś nowego, posłuchania mądrzejszych, podyskutowania
- **psychologiczny:** stworzenie atmosfery ułatwiającej przekazanie ludziom nowych idei, podsunięcie im nowych sposobów wartościowania.

Pierwsza piątka Szkół Mikroprocesorowych, połączona „słowem ośmiobitowym" podjemowała różne tematy. Spotkania odbywały się w różnych warunkach i należałoby każde z nich oceniać z osobna. Nie jestem jednak pewien, czy oceny ich byłyby porównywalne. Spróbuję zatem spojrzeć z pewnego dystansu na całość.

- **W wymiarze dokumentalnym** — osiągnięcia Szkoły są niepodważalne. Co prawda Szkoła nie jest jedynym forum prezentacji nowości w mikroinformatyce, ale znaczący jest fakt, że omawiane są tu tylko własne, konkretne opracowania. Nie ma żadnych prac przeglądowych, pseudoteorii, itp.

- **W wymiarze organizacyjnym** — Szkoły Mikroprocesorowe rzeczywiście pozwoliły na poznanie się ludzi i w pewnym stopniu ułatwiły im dalsze kontakty. Bardziej wyraźne było to na początku, gdy uczestników było pięćdziesięciu, niż ostatnio, gdy ich liczba wzrosła do dwustu. Mam jednak wątpliwości, czy realnie udało się połączyć wysiłki różnych zespołów, czy przemyślenia wyniesione ze Szkoły przyczyniły się do zaniechania prac wątpliwych lub podjęcia tematów ważnych i perspektywicznych. Trudno jest sterować informatyką...

- **W wymiarze informacyjnym** — widać już korzystną wymianę doświadczeń zawodowych. Doskonałym przykładem tego jest referat z tegorocznej Szkoły poświęcony doświadczeniom eksploatacyjnym w zakresie przenoszenia zbiorów na dyskach elastycznych, budzący wielkie zainteresowanie uczestników. Taka rzeczowa informacja powinna być wyraźnie popierana przez organizatorów Szkoły, aby stanowiła przeciwwagę dla licznych pytań typu: „gdzie, co, za ile i jak załatwić?". Choć i takie dane pomagają w pracy.

- **W wymiarze edukacyjnym** — obserwuję zniechęcenie, wynikające zapewne z niedoceniana go przez słuchaczy. Odnoszę wrażenie, że wymiar ten wymknął się spod kontroli organizatorów. Często zdarzają się referaty o dużym ładunku bardzo potrzebnej wiedzy — lecz, niestety, bardzo „niedydaktycznie" przedstawione. Można to jeszcze wybaczyć młodym uczestnikom bez odpowiedniej praktyki w nauczaniu, ale że utytułowani prelegenci nie byli w stanie niczego ciekawego i wartościowego przekazać słuchaczom — tego już nie rozumiem. Jakże często była to powtórka z dostępnej literatury, bez krytycznego komentarza, bez pogłębionej analizy. Taki wykład szybko ulatuje z pamięci, nie pozostawia żadnych refleksji. Szczególny żal mam do uczestników dyskusji „pięciobocznego stołu". Panowie! Czy macie coś ważnego do powiedzenia ludziom? Nie wygłaszajcie przemówień „w powietrze" — słuchający czują, że to nie do nich, że to na pokaz. Czy wiecie, jak niewiele brakowało, a zostalibyście... wygwizdani?!

[1]) Dańda J.: Mikroprocesorowe Szkoły Zimowe — refleksje współorganizatora. INFORMATYKA nr 2, 1983

Podczas V Szkoły często czyniono próby, aby do dyskusji włączyć słuchaczy. Niestety, prowadzący dyskusję zręcznie „wyłączał" te osoby, które chciały powiedzieć coś konkretnego od siebie. Równocześnie pozwalał wygadać się „na okrągło" innym, wyróżnionym przez niego. Bywały gorzkie chwile, kiery słuchacze — przybyli z całej Polski — zaczęli przypuszczać, że stanowią tylko tło dla pokazowych wystąpień gości ze stolicy. Obym był przewrażliwiony! W Szkole jednakowo dobrze powinni czuć się wszyscy obecni. Powinna też uczyć wzajemnego szacunku.

• **W wymiarze psychologicznym** — Szkoły Mikroprocesorowe są z pewnością inne niż większość narad, konferencji i szkoleń. Mają one swój nastrój, koloryt. Niewątpliwe Szkoły zawdzięczają atmosferę Jerzemu Dańdzie. Chociaż... wielu słuchaczy było aż przytłoczonych Jego oryginalną osobowością. Nietypowy sposób prowadzenia spotkań był odczuwany przez wielu z nas jako tzw. „manipulacja". Ja odnoszą jednak wrażenie, że Jerzy Dańda chciał słuchaczom coś w ten sposób przekazać. Ale co? Może było to skierowane do naszej podświadomości? Może ów podtekst ma jeszcze raz przypominać znaną prawdę, że klucz do rozwiązania wielkich i trudnych problemów ludzkich (także zawodowych!) znajduje się w nas samych?

Uważam, że pierwsze Szkoły Mikroprocesorowe spełniły znaczną część stawianych przed nimi wymagań. Są też nadal potrzebne, zwłaszcza w dzisiejszych warunkach. Jednak w miarę ich rozwoju, potrzebne są istotne zmiany — także w programie.

Kolejna, VI Szkoła Mikroprocesorowa ma być poświęcona systemom 16-bitowym. Czy to jednak nie za wcześnie? Czy nie grozi to zamknięciem Szkół 8-bitowych? Ileż nieciągłości w naszym rozwoju jest zawinionych przez nas samych! Jeszcze wiele lat systemy 8-bitowe będą stanowiły podstawę krajowej mikroinformatyki. Nie sądzę, aby dziś osiągnęły one szczyt swego rozwoju w Polsce. Choć istnieją różne spotkania specjalistyczne poświęcone aplikacjom mikroprocesorów (8-bitowych), nie możemy zamykać dotychczasowych szkół będących jedynym forum interdyscyplinarnym w tej dziedzinie. Wypadałoby raczej poszukać nie tylko nowych treści, odpowiednich do potrzeb, ale i zmienionej formuły spotkań.

Znajdźmy też podczas Szkolnych spotkań czas na spokojną, głęboką zadumę nad kierunkami naszego zawodowego działania. Nie bójmy się trudnych pytań; choćby takiego — czy możemy mieć naszą, polską drogę w mikroinformatyce. Szkoła powinna wyprzedzać swoje czasy.

JACEK ŻEBROWSKI

Następna piątka

Piąta Szkoła Mikroprocesorowa odbyła się w Łodzi w dniach 5—7 marca br.[1]) Następna zaplanowania jest już na grudzień tego roku w Rydzynie, w pięknym ośrodku SIMP, w pałacu zamieszkiwanym w zamierzchłych czasach przez Leszczyńskich, a potem Sułkowskich, niedaleko Leszna, VI Szkoła rozpocznie serię seminariów poświęconych wyłącznie problemom mikroprocesorów 16-bitowych w zasadzie ograniczonym do typów: 8086/88, 68000 i Z8000.

Jeśli okaże się, że zgłoszeń jest dużo — pałac w Rydzynie mieści sto osób (a zaplanowana jest wystawa sprzętu 16-bitowego i dla uczestników zostanie tylko 75 miejsc) — to VI Szkoła Mikroprocesorowa BIS przeniesie się bezpośrednio (w tym samym tygodniu) z Rydzyna do Łodzi lub Uniejowa.

Autorzy referatów będą proszeni, aby swój tekst i swoje działania w Szkole organizowali zgodnie z trzema zasadami, określającymi „edukacyjny" cel Szkoły:

• Nauczyć wszystkiego, co może być użyteczne (i nowe!) dla innego uczestnika Szkoły.

• Uczyć tak, aby uczestnik wiedział, że Szkoła jest dla niego, a nie dla wykładowców i organizatorów.

• Nie bać się konkurencji. Nasze wiadomości muszą się stać dobrem społecznym. Plagiaty i zwykłe kradzieże pomysłów rozpoznamy łatwo.

Jeszcze jeden zwiastun: VII Szkoła Mikroprocesorowa odbędzie się w marcu 1985 i będzie poświęcona ikonice i grafice komputerowej. Chętnych do zgłoszenia referatu proszę o kontakt telefoniczny (tel. 47-12-61 lub 34-89-05 w Warszawie). Chciałoby się bowiem by materiały tego seminarium miały formę dorównującą treści. A to lubi długo trwać w naszych warunkach.

JERZY DAŃDA

[1]) Program Szkoły zamieściliśmy w numerze 1, 1984 — przyp. red.

Szansa w Szkole

Przywykliśmy chwalić pionierskie przedsięwzięcia, przymykając oczy na ich niedostatki — tylko po to, aby nie zniszczyć tzw. zalążka. Tym razem pomińmy komplementy dla organizatorów Szkół Mikroprocesorowych i zastanówmy się jakiej natury forum potrzebne jest profesjonalnej mikroinformatyce. Jak uniknąć powielenia tych samych błędów, które miały miejsce przy upowszechnianiu techniki 8-bitowej? Faktem jest, że Szkoły w swej dotychczasowej, zdawałoby się niezłej, formule nie zapobiegły ich popełnieniu.

Można oczekiwać, że już niebawem kilkadziesiąt ośrodków zabierze się do 8086, klecąc różne dziwolągi — mające spełnić zbliżone założenia. Kilkadziesiąt zespołów zdolnych i ambitnych ludzi będzie rozwiązywać dokładnie te same problemy. 98 procent tych prac zakończy się wykonaniem raptem kilku sztuk systemów mikroprocesorowych. Absurd! W końcu szanujący siebie inżynier nie powinien wykonywać prac praktycznie bezużytecznych. Jest istotą rozumną, a nie narzędziem chaotycznego systemu.

Wkraczamy w nowy etap rozwoju mikroinformatyki i zanosi się na powtórzenie jota w jotę tych samych błędów. Wkraczamy, więc jeszcze jest szansa ich uniknięcia. Tylko nie wzruszajmy ramionami, że to nie my, konstruktorzy, jesteśmy od organizacji — nie ma w tej sferze ludzi bardziej kompetentnych od nas! „Oni" mogą nam jedynie zafundować coś na kształt licencji Fergusona. Przy naszym mikropotencjale musimy znacznie intensywniej współpracować niż konstruktorzy z krajów rozwiniętych technicznie; i z pewnością musimy tę współpracę oprzeć na innych, nie tak sformalizowanych zasadach jak licencje czy patenty. Wspólny interes musimy też przedłożyć nad animozje i lokalne ambicyjki szefów. Właśnie Szkoła Mikroprocesorowa może stać się płaszczyzną wymiany doświadczeń. Nie suchych komunikatów o tym „co" zostało zrobione, lecz informacji „jak" należy to zrobić, jakie trudności pokonać.

Taka formuła nie może jednak opierać się na altruizmie, zresztą rzadko dzisiaj spotykanym. Większość z nas liczy, że wcześniej czy później uda się spieniężyć swój trud — trudno się więc dziwić, że przy nędznej płacy pilnie strzeże swych sekretów. Ustalenie (choćby nieoficjalnych) reguł i sposobów wzajemnego płacenia sobie za przekazane doświadczenia, błyskawicznie postawiłoby na nogi polską mikroinformatykę. Udrożniłoby wykorzystywanie rzeczy już zrobionych i pozwoliłoby skupić się na tym, co zostało do zrobienia.

Przypuszczam, że po takiej „reformie" organizatorzy Szkoły nie musieliby uganiać się za autorami dobrych referatów, a mikroprocesory zadziwiająco łatwo wkroczyłyby w te wszystkie zastosowania, w których nawet u nas powinny być już od lat.

ANDRZEJ J. PIOTROWSKI

Prezentujemy trzy opracowania z prasy zachodniej pozornie ze sobą nie związane. Pierwsze dotyczy sytuacji na informatycznym rynku pracy we Francji, drugie — kształcenia młodych kadr średniego szczebla w Wielkiej Brytanii, trzecie — nauczania wspomaganego komputerem w USA. Artykuły te nie pretendują do miana głębokiej analizy zagadnienia, ale — naszym zdaniem — dobrze odzwierciedlają wzrastające zainteresowanie problemami informatyzacji. Zainteresowanie, ale zarazem pewną bezradność czy wprost niezrozumienie nowych, nie przewidzianych zjawisk. Prasa zachodnia poświęca dużo uwagi spekulacjom na temat skomputeryzowanego społeczeństwa przyszłości (podobno już bardzo niedalekiej). Opinie i oceny są często sprzeczne. Spotyka się głosy tak o „elicie informatycznej", jak i o powszechnej dostępności nowoczesnej techniki komputerowej. Przedstawia się niezbyt optymistyczne prognozy gwałtownego spadku zatrudnienia spowodowanego rozwojem mikroelektroniki... Aż nagle okazuje się, że rzeczywistość nie dorosła do realizacji tych wizji. Francuzi ze zdziwieniem zauważają, że zawód informatyka nie jest ani lepszy, ani gorszy od innych. Anglicy borykają się z problemami szerokiego upowszechnienia zdobyczy technologii informatycznych. A Amerykanie, rzekomo najbliżsi zrealizowania idei „elektronicznego społeczeństwa", napotykają niemałe trudności przy wprowadzaniu komputerów do szkół. (Red.)

Francja

Informatycy zatrudnieni

Od ubiegłego roku komputery wkroczyły w życie domowe Francuzów. Wszyscy, albo prawie wszyscy, mogą zetknąć się bliżej z informatyką. Zawód związany z tą dziedziną nie wydaje się już tak odległy. Ale kim są informatycy? Jak się ich rekrutuje? Według jakich kryteriów? Jak przebiega potem ich kariera? Na te pytania poszukano odpowiedzi w cyklu spotkań z konstruktorami, przedstawicielami firm informatycznych i przedsiębiorstw usługowych.

Jedno jest od razu jasne: informatyka to raj. Nie ma bezrobocia, żadnych problemów z zatrudnieniem, brak troski o przyszłość, bardzo dobre wynagrodzenie. Słowem — świat sam w sobie. Ale sprecyzujmy: świat sam w sobie dla tych, którzy pasują do systemu. Zbyt często na rynku pracy spotyka się pseudoinformatyków, pospiesznie wykształconych w szkołach o wątpliwym poziomie. Ci nie mają żadnych szans. Informatyka jest jednym z rzadkich sektorów, gdzie mimo kryzysu można znaleźć dobrą pracę — o ile ma się dobre przygotowanie zawodowe.

Tendencję tę dobrze ilustrują rezultaty ostatniej ankiety kwartalnej APEC. W roku 1983 pojawił się w gospodarce francuskiej pewien spadek zatrudnienia ludzi o wysokich kwalifikacjach, a w szczególności zmniejszenie o 10% przyjęć do pracy. Tylko w informatyce sytuacja jest wciąż dobra. APEC stwierdza, że „przedsiębiorstwa dokonały w roku 1983 wysiłku na rzecz zwiększenia zatrudnienia informatyków, powiększając i tak już duży nabór: 14% rekrutacji w stosunku do 12% w roku 1982". Ta korzystna sytuacja powinna utrzymać się w pierwszym kwartale bieżącego roku. Jedynym powodem do niepokoju mógłby być gwałtowny spadek zapotrzebowania w grudniu 1983 (o 26% mniejsze w stosunku do tego samego okresu poprzedniego roku). Ale ostatecznie spadek ten okazał się sporadyczny i nie przyniósł zmian w sytuacji informatyków na francuskim rynku pracy. Jak więc widać, zawody informatyczne są dobrze chronione przed ekonomicznymi niespodziankami.

Dla wszystkich spotkanych informatyków — zarówno tych u dołu hierarchicznej drabiny, jak i dla kadry kierowniczej — stabilność zatrudnienia jest sprawą oczywistą. Większość traktuje stosunki ze swoim pracodawcą „po amerykańsku". Nie ma bojowych działaczy wśród związkowców. Jeśli nie następuje podwyżka płac, a stosunki służbowe układają się źle, porzuca się stanowisko bez wahania. Nie bez powodu takie firmy, jak IBM, TEXAS INSTRUMENTS czy DIGITAL EQUIPMENT mają siedziby nad Morzem Śródziemnym, albo — jak HEWLETT-PACKARD — w Grenoble. „Nie przeniesiemy się do Longwy, mimo że jest to strefa superbezcłowa — mówi szef wielkiego przedsiębiorstwa informatycznego — pracownicy nie pójdą za nami. Stracimy cały potencjał szarych komórek".

Inny ważny aspekt mentalności informatycznej to akceptacja pewnej formy samobójstwa zawodowego. Jeśli po upływie pewnego czasu dana osoba widzi, że nie uda jej się wybić czy rozwinąć, zgadza się odejść. Atmosfera wielkiego luzu, jaka panuje w informatyce, nie powinna nikogo łudzić. Mówi pracownik APPLE: „Kierownik jest ze wszystkimi na ty. Ale to mu nie przeszkadza pozbyć się inżyniera, któremu się nie udaje. Zresztą, taki sam zrozumie".

Ten stan ducha zależy też od firmy. Mniej wyczuwalny jest w przedsiębiorstwach francuskich, takich jak BULL czy THOMSON. Natomiast w IBM presja jest ogromna. Ludzie rzeczywiście odchodzą sami — po postawieniu sobie diagnozy. Świat IBM warunkuje trochę życie całego sektora informatyki. Osoba która opuściła IBM zaledwie po dwóch czy trzech latach, będzie miała więcej kłopotów ze znalezieniem pracy niż debiutant np. z Instytutu CONTROL DATA. Bo jeśli IBM odrzucił fachowca, to znaczy, że był on po prostu słaby.

Ważną cechą informatyki, spotykaną w przedsiębiorstwach typu CAP SOGETI, CISI czy UNILOG, jest brak jakichkolwiek rygorów formalnych. Czas pracy — 39 czy 35 godzin — to pojęcie abstrakcyjne. Nie jest się dobrym informatykiem, jeżeli śledzi się wskazówki zegara. Najważniejsze, by praca była wykonana dobrze i o czasie.

Tylko dwie kategorie zatrudnionych mogą obecnie narzekać. Należą do nich informatycy z dołu „drabiny", wyspecjalizowani w technice, która właśnie wychodzi z użycia — np. operatorzy urządzeń wprowadzania danych, niektórzy programiści. Bo pewne jest, że przyszłość należy do maszyn, których systemy wejścia pozwalają obyć się bez odpowiedniej pracy ludzkiej. Drugą kategorię stanowią informatycy zatrudnieni przez instytucje państwowe. Jako urzędnicy zarabiają znacznie gorzej od swoich kolegów z innych firm; mówi się, że o 30%, ale z pewnością jeszcze mniej — gdy wziąć pod uwagę konkurencyjne premie. Instytucje państwowe są sektorem zatrudniającym najwięcej informatyków, ok. 20 tys. Mają również największe trudności — z racji rygorów siatki płac. Rząd francuski zdecydował się ostatnio na utworzenie pionu informatycznego w służbach publicznych. Wydaje się to trudne do urzeczywistnienia, bo w żadnym przypadku przepisy nie pozwolą na zbliżenie płac w sektorze prywatnym do płac w sektorze publicznym. A nie można podwyższyć poziomu informatyki w ministerstwach i administracji bez udziału kompetentnego personelu.

Informatycy wykazują coraz większą potrzebę spokoju — pragnienie, które mogłoby znaleźć spełnienie w wygodnej pracy urzędniczej. Zobaczmy bowiem, jaki jest przebieg przeciętnej kariery zawodowej. Po zdobyciu solidnego wykształcenia najkorzystniej jest znaleźć pracę w przedsiębiorstwie z branży, najlepiej małym lub średnim. Następnie trzeba przejść do jakiejś „fabryki poznania" — renomowanej firmy. Ta pierwsza faza kariery pociąga za sobą częste zmiany miejsca pracy i to na krótkie okresy. Nie należy jej pomijać.

Pewne niedogodności tych, którzy wkraczają do świata informatyki, biorą się stąd, że trzeba się stale przeprowadzać. W końcu liczni informatycy zaczynają marzyć o zatrudnieniu przez klienta, który od lat zasięgał u nich konsultacji. Wiek krytyczny to mniej więcej 27—28 lat. Albo omija się tę granicę i nadal prowadzi życie wędrowne aż do osiągnięcia zadowalającego awansu, albo wybiera się „ciepłą posadę" u klienta, szefując serwisowi informatycznemu.

Motywacja konstruktorów wyraźnie przechodzi ewolucję. Nie dążą już oni do sprzedania urządzenia za wszelką cenę. Obecnie, dzięki mikrokomputerom i sieciom teleinformatycznym, można wziąć pod uwagę przede wszystkim potrzeby klienta. Następuje specjalizacja sprzedawców, dokonujących szczegółowych analiz tych potrzeb: w klinikach, laboratoriach, wśród odbiorców maszyn rolniczych, itp.

Rozmówcy z firmy NIXDORF zwracają uwagę, że najrzadziej spotyka się personel najbardziej poszukiwany, tzn. informatyków okopanych na flankach wielkich organizacji, jak banki, ubezpieczenia, administracja. W dziedzinach tych liczy się nie sprzedanie czy nawet zaproponowanie gotowego rozwiązania, ale przede wszystkim — nawiązanie kontaktów, stosunki pielęgnowane nieraz bardzo długo. Nic dziwnego, że taki informatyk jest bezcenną zdobyczą dla pracodawcy.

Opracowała MAGDALENA HEN
na podstawie LE FIGARO, 7 lutego 1984

*

USA

Nauczanie wspomagane komputerem

USA charakteryzuje — jak wiadomo — szczególnie zróżnicowany system oświatowy, odmienny nie tylko w każdym ze stanów, ale i w poszczególnych miastach. Lokalne podatki kształtują sposób finansowania, a tym samym — politykę rozwoju placówek oświatowych. Oprócz szkół publicznych, istnieje bardzo rozbudowany sektor szkolnictwa prywatnego. Sytuacja ta powoduje, że w dziedzinie nauczania wspomaganego komputerem (CAI — Computer Assisted Instruction), obok licznych ogólnokrajowych prac badawczych prowadzonych np. przez NSF (National Science Foudation), NIE (National Institute of Education) czy USOE (US Office of Education, tzn. Ministerstwo Oświaty), spotkać można mnóstwo pojedynczych nowatorskich rozwiązań. Zróżnicowanie to jest często bardzo duże — nawet w ramach poszczególnych szkół spotkać można zastosowania mikrokomputerów realizowane samorzutnie przez nauczycieli.

Prognozy na rok 1979 określały, że CAI obejmie w USA 6% ogólnej liczby szkół podstawowych i średnich. Ale już w roku następnym na 21 630 szkół ponadpodstawowych oraz 9,8 mln uczniów i studentów tylko 973 szkół (ok. 4%) oraz 1,3 mln słuchaczy (ok. 13%) nie było objętych CAI. Na przeszkodzie jeszcze szybszego rozwoju stanął jedynie brak dobrego oprogramowania dydaktycznego, a także niedostateczne jeszcze zrozumienie roli informatyki przez niektóre kręgi administracji oświatowej, a nawet nauczycieli i uczniów. Chcąc przełamać istniejące opory, amerykańska Narodowa Rada Nauczycieli Matematyki (National Council of Teachers of Mathematics) oświadczyła w 1980 roku, że „...szkoły wszystkich stopni powinny w pełni wykorzystywać zalety komputerów, a kierownictwa tych szkół — uznać to narzędzie za równie istotne, jak inne środki dydaktyczne, łącznie z podręcznikami. Każdy uczeń powinien wziąć udział w zajęciach z komputerem oraz zrozumieć istotę informatyki".

Nowojorski instytut naukowy ICIS (International Center for Integrative Studies) opracował w 1980 roku przegląd najważniejszych amerykańskich osiągnięć badawczych w dziedzinie CAI „The ICIS Guide to Educational Technology: Who's Doing What in Innovative Learning". Przegląd ten podaje informacje o następujących nowatorskich rozwiązaniach CAI oraz o działających w tej dziedzinie instytucjach.

Projekt LOGO

Autorem tego nowatorskiego systemu nauczania jest Seymour Papert, pracownik naukowy laboratorium sztucznej inteligencji Uniwersytetu MIT. Stwierdził on, że większość systemów CAI jest skonstruowana błędnie, ponieważ uczeń musi zachowywać się biernie (przyjmować i wykonywać polecenia maszyny). W systemie LOGO dziecko uczy się sukcesywnie, w sposób czynny. Polega to na tym, że poleca ono robotowi (pod postacią żółwia morskiego) rysowanie piórem na papierze figur geometrycznych, dzięki czemu stopniowo, drogą samodzielnego rozumowania i poszukiwania, opanowuje szkolny program geometrii. Pisząc na klawiaturze komputera odpowiednie słowa, dziecko daje żółwiowi polecenie rozpoczęcia lub przerwania operacji rysowania, wykonania ruchu piórem o odpowiednią liczbę skoków do przodu, obrócenia narysowanej prostej w prawo lub lewo o odpowiednią liczbę stopni. Może też ono podać żółwiowi ciąg procedur oraz powtórzyć poszczególne podprocedury określoną liczbę razy. W ten sposób uczeń może samodzielnie pojąć najtrudniejsze nawet problemy tradycyjnego szkolnego programu geometrii.

Projekt PLATO

System PLATO (Programmed Logic for Automated Teaching Operations), opracowany i eksploatowany na Uniwersytecie Illinois, uważa się za najbardziej rozbudowany ze stosowanych w USA systemów CAI. Ponad tysiąc użytkowników może równocześnie korzystać — poprzez zdalne terminale i publiczną sieć telefoniczną — z zasobu dziesiątków tysięcy godzin nagranych programów dydaktycznych. Programy te obejmują olbrzymi wachlarz przedmiotów nauczania [1]).

Oparty na bardzo dużej konfiguracji komputerowej, PLATO ma zalety i wady systemu scentralizowanego. Podstawową wadą są wysokie koszty łączności z komputerem, ze względu na konieczność blokowania linii telefonicznej przez cały czas trwania lekcji. Dlatego też przystąpiono do prób zastosowania mikrokomputerów oraz wideodysków, co powinno zredukować do minimum koszty łączności telefonicznej z komputerem centralnym (potrzebne oprogramowanie dydaktyczne przesyłane będzie z centralnej biblioteki na początku lekcji na dysk mikrokomputera-terminala).

Inteligentne CAI

Autorem tego systemu jest prof. Patrick Suppes z Uniwersytetu Stanford, jeden z pionierów CAI, który w ciągu ostatnich dwudziestu lat opracował i wdrożył 15 tego typu systemów. Z systemów tych w 1978 roku korzystało ponad 150 tys. uczniów w 24 stanach. Najbardziej znane są kursy matematyki dla dzieci do 12 lat oraz kursy wykorzystywane przez większość wydziałów Uniwersytetu Stanford.

Suppes jest obecnie pionierem badań nad symulacją dialogu człowieka z komputerem, w którym z jednej strony maszyna generuje pytania i odpowiedzi głosem ludzkim, a z drugiej — odbywa się przetwarzanie informacji sformułowanych w języku naturalnym. System ten nazwany mikroprogramowanym syntetyzerem mowy intonowanej (ang. microprogrammed intoned speech synthesizer) pozwala na razie przetwarzać stosunkowo proste fragmenty wypowiedzi w języku angielskim. Profesor twierdzi, że w ciągu najbliższych 10—20 lat komputer będzie mógł efektywnie przetwarzać również sformułowania złożone. Jedną z jego najbardziej obiecujących koncepcji jest tzw. inteligentne CAI, w którym komputer tworzy model rozumowania studenta, bada

[1]) System PLATO został szczegółowo opisany w artykule Marka Hołyńskiego: Symulacja procesu nauczania w systemie PLATO, INFORMATYKA nr 2, 1981 str. 4—8 (przyp. red.)

i koryguje model oraz stosuje go praktycznie do udzielania pomocy studentowi w pokonywaniu „pułapek", wynikających z błędnego toku rozumowania.

CONDUIT

Z chwilą gwałtownego obniżenia cen sprzętu komputerowego, stwierdzono, że podstawową przeszkodą dalszego szybkiego rozwoju CAI jest brak dobrego oprogramowania dydaktycznego. Jedną z najważniejszych instytucji amerykańskich, której podstawowym celem jest usunięcie tej przeszkody, jest CONDUIT. Ma on ułatwić rozpowszechnianie materiałów dydaktycznych CAI przeznaczonych dla szkół wyższych.

CONDUIT został założony przez National Science Foundation, ale obecnie jest już jednostką całkowicie samodzielną, działającą przy Uniwersytecie Iowa. CONDUIT wyszukuje opracowane programy dydaktyczne, przeprowadza fachową ocenę ich wartości użytkowej, wybiera rozwiązania najlepsze, opracowuje jednolitą dokumentację i formę gotowych produktów (pakietów) oraz rozprowadza je na rynku metodą zamawiania z katalogu. Ostatnio wydany katalog zawiera ok. 80 tego rodzaju pakietów, podzielonych tematycznie na 14 dyscyplin naukowych. Ceny pakietów wynoszą zaledwie od 5 do 125 dolarów, przy czym rozwiązania dostosowane są do różnych modeli komputerów (także mikrokomputerów).

MECC

Instytucja ta, której pełna nazwa brzmi Minnesota Educational Computing Consortium, od 1973 roku obsługuje w zakresie zastosowań CAI wszystkie placówki dydaktyczne stanu Minnesota (ok. 4 mln mieszkańców). Jej podstawowym celem jest udzielanie pomocy nauczycielom w stosowaniu komputera do dydaktyki szkolnej. Eksploatuje ona jeden z największych w świecie scentralizowanych systemów dydaktycznych, zapewniając 95% ogólnej liczby uczniów i studentów wspomnianego stanu bezpośredni dostęp do ok. 1000 programów dydaktycznych. Oprócz tego, instytucja ta zainstalowała w szkołach ponad 1000 mikrokomputerów, wspomagając je sztabem 150 specjalistów; udziela też pomocy nauczycielom w zaprojektowaniu i napisaniu oprogramowania dydaktycznego, a także udostępnia zainteresowanym skomputeryzowany system informacyjno-dokumentacyjny. Jako organ stanowy, MECC zapewniła jednolity poziom kosztów transmisji danych, bez względu na odległość szkoły od ośrodka komputerowego.

Opracował
WŁADYSŁAW KLEPACZ
na podstawie
AGORA,
styczeń—marzec 1982

*

Wielka Brytania

Sposób na adepta

W roku 1979 rozpoczął w Londynie działalność pierwszy ośrodek informatyczno-technologiczny, prowadzący kursy informatyki. Wyodrębnił się on z instytutu studiów urbanistycznych, w którym zaczęto interesować się problemami informatyzacji kraju. W roku 1981 ośrodek odwiedził minister przemysłu, Kenneth Baker. Doszedł on do wniosku, że utworzenie podobnych placówek pozwoliłoby na szersze upowszechnienie technologii informatycznej w całym kraju. Rząd brytyjski podjął decyzję o utworzeniu 150 ośrodków do końca roku 1983. Dzięki temu zaczęto tworzyć sieć placówek, w których uczy się nowicjuszy informatyki. Głównym zadaniem „ośrodków technologii informatycznej" (ITEC) jest kształcenie bezrobotnych absolwentów szkół technicznych, co w przyszłości zapewni im zdobycie zatrudnienia. Jednoroczne kursy, liczące po 30—70 słuchaczy, są finansowane przez rząd, który przeznacza na ten cel 25 mln funtów rocznie.

Najwięcej uwagi poświęca się nauce praktycznych umiejętności: operowaniu i programowaniu maszyn, konserwacji sprzętu, ogólnemu zapoznaniu się z pracą w biurach (w tym posługiwaniu się procesorami tekstowymi). Chociaż dziewczęta i chłopcy mogą uczęszczać na te same zajęcia, jednak o wiele częściej widzi się dziewczęta przy ekranach monitorów, niż np. przy konstruowaniu nowych modeli robotów.

Jednym z ubocznych skutków istnienia kursów ITEC jest wprowadzanie informatyki do małych przedsiębiorstw. Często dochodzi do porozumienia między organizatorami szkoleń a właścicielami miejscowych firm. Przez kilka miesięcy uczniowie zdobywają tam pierwsze doświadczenia w pracy, pomagają przy księgowaniu albo naprawiają urządzenia elektroniczne. „W wielu małych przedsiębiorstwach pracują ignoranci w dziedzinie komputeryzacji — zauważa jeden z przedstawicieli ITEC. Jeżeli jednak widzą szesnastolatka posługującego się maszyną bez najmniejszych problemów, zaczynają wierzyć, że komputer jest całkiem dobrym wynalazkiem".

Z bezpłatnych usług ITEC korzystają również osoby prywatne. Dobrym tego przykładem jest pastor z hrabstwa Yorkshire, który — sam będąc entuzjastą informatyki — nie miał czasu, by stworzyć oprogramowanie pozwalające skatalogować 1,5 mln pozycji ewidencjonujących członków parafii. Dane zgromadzone były w bardzo wielu księgach, niektórych aż z XVI wieku. Uczestnicy kursów bezpłatnie wykonali to zadanie, zdobywając przy tym pierwsze doświadczenia w pracy z komputerem, a parafianie mogą teraz bez kłopotu śledzić swoje drzewa genealogiczne.

Wszyscy uczestnicy kursów w pełnym wymiarze godzin to nastolatki, którzy niedawno opuścili szkoły. Niektóre ośrodki proponują również popołudniowe zajęcia dla dorosłych. Około 70% absolwentów znajduje zatrudnienie, najczęściej jako konserwatorzy urządzeń elektronicznych, operatorzy procesorów tekstowych, początkujący programiści. Niektórzy wybierają zawody nie związane z informatyką. Wielu uczniów opuszcza kursy przed upływem roku, ponieważ szybciej zdobywają kwalifikacje niezbędne do podjęcia pracy.

Techniki nauczania proponowane przez ITEC różnią się od stosowanych powszechnie w szkołach i innych placówkach tego typu. ITEC poświęca wiele uwagi zajęciom indywidualnym. Na jednego wykładowcę przypada średnio pięciu uczniów. Nauczyciele są zawodowo bardzo dobrze przygotowani. Większość z nich to pracownicy wielkich producentów sprzętu takich jak ICL czy FERRANTI, lub też własnych warsztatów obsługi sprzętu. Część stanowią normalni nauczyciele; są także naukowcy zatrudnieni w różnych uczelniach.

Uczniowie mają do czynienia ze sprzętem, który najczęściej widuje się głównie w nowoczesnych salonach wystawowych: mikrokomputery połączone z procesorami tekstowymi, terminale ekranowe, komputery o wielkich mocach obliczeniowych dostępne przy użyciu nowoczesnych technologii transmisji danych... Koszty przedsięwzięcia są w związku z tym dość wysokie. Kurs kosztuje ok. 5 tys. funtów rocznie, gdy uwzględnić płace instruktorów i ceny eksploatacji wyposażenia. Uczestnik płaci tylko 25 funtów tygodniowo, resztę pokrywa rząd.

Opiekę nad ośrodkami ITEC sprawują często firmy elektroniczne, a także władze lokalne. W zamian za ulgi finansowe, warsztaty ITEC naprawiają bezpłatnie sprzęt komputerowy w szkołach państwowych, a uczniowie prowadzą zajęcia dla nauczycieli i personelu szkół w zakresie obsługi i konserwacji urządzeń elektronicznych.

Obserwując wyniki działalności ITEC, można śmiało twierdzić, że — mimo powszechnego przeświadczenia, iż rozwój mikroelektroniki i komputeryzacji zmniejszy zatrudnienie — jest wciąż jeszcze dużo miejsc pracy związanych z wykorzystaniem komputerów. I — co najważniejsze — młodzi ludzie uczący się na kursach ITEC chcą te prace wykonywać.

Opracowała KATARZYNA ISAAK
na podstawie NEW SCIENTIST, 18.01.1983

Problemy związane z ochroną prawną oprogramowania były od pewnego czasu przyczyną niepokojów i sporów pomiędzy producentami. W ubiegłym roku odbyło się kilka procesów o nielegalne kopiowanie systemów operacyjnych komputerów osobistych, których bohaterem była najczęściej firma APPLE. Przebieg jednego z nich, przed Trybunałem Handlowym w Paryżu, przedstawiamy szczegółowo, ponieważ — jak to wynika z dalszej części artykułu — jest on bardzo charakterystyczny dla stanowiska prawników zachodnich wobec problemów ochrony oprogramowania prawem autorskim. Warto również zwrócić uwagę na istotne konsekwencje ekonomiczne wydanych wyroków i ich wpływ na kierunki dalszego rozwoju przemysłu informatycznego. (Red.)

Chronić dzieło programisty!

Niedługo przed otwarciem wystawy MICRO-EXPO'83 pojawił się na rynku mikrokomputer GOLEM o bardzo dobrych parametrach w swojej kategorii. Produkowano go na Tajwanie, a sprzedażą zajęły się firmy SEGIMEX, CONTROL DATA i SYBEX. Szybko dostrzeżono jednak niewątpliwe związki GOLEMA z maszynami APPLE II. Stwierdzone podobieństwo opierało się tym razem na bezpośrednim naśladownictwie oprogramowania podstawowego. Dotychczas większość posądzeń o piractwo komputerowe była związana z oprogramowaniem użytkowym lub niedozwolonym powielaniem znaków firmowych czy nazw handlowych.

Firma APPLE postanowiła wkroczyć na drogę sądową i oświadczyła, że komputer GOLEM zawiera oprogramowanie podstawowe serii APPLE II. Firma miała do wyboru dwa sposoby dalszego postępowania: domaganie się wyroku ogłoszonego zgodnie z ogólnymi zasadami prawa cywilnego bądź żądanie konfiskaty plagiatu w sensie prawa autorskiego. APPLE wybrało drugą możliwość. Konfiskaty dokonał komornik w asyście eksperta — informatyka. W efekcie firma dysponowała dowodem naśladownictwa. Jednakże importerzy, pewni swoich racji, postanowili nie rezygnować z wystawienia GOLEMA na MICRO-EXPO'83. W tej sytuacji APPLE musiało bronić swego honoru i sytuacji finansowej, domagając się w postępowaniu przyspieszonym:
— zakazu wystawiania komputera GOLEM podczas całego okresu trwania MICRO-EXPO'83
— zakazu jakiejkolwiek reklamy GOLEMA, zawierającej naśladownictwo charakterystycznego znaku w kształcie jabłka i sekwencji kolorów, pod groźbą kary 50 tys. franków za każdy dzień wystawy
— zakazu jakiegokolwiek importu bądź sprzedaży GOLEMA lub innych komputerów zawierających skopiowane oprogramowanie pod groźbą kary 10 tys. franków za każdy sprowadzony lub sprzedany egzemplarz.
Dowód stanowiły dostarczone na rozprawę wydruki odtwarzające oprogramowanie podstawowe APPLE II i GOLEMA.

Żądania wysunięte przez APPLE były wyjątkowo niebezpieczne, ponieważ mogły naruszyć równowagę ekonomiczną i finansową przedsiębiorstw pozwanych przed sąd paryski. Nic zatem dziwnego, że z dużym niepokojem i zainteresowaniem oczekiwano decyzji Trybunału. Aby uzyskać pełniejszy obraz sytuacji, sąd postanowił powołać eksperta, którego zadaniem było:
— opisać, na podstawie dokumentów przedstawionych przez firmę APPLE COMPUTER, podobieństwa i różnice między oprogramowaniem podstawowym serii APPLE II i GOLEMA
— stwierdzić, czy ewentualne podobieństwa wynikają z warunków technicznych, niezależnych od przyjętej koncepcji ogólnej.

Tym samym Trybunał dokonał wyraźnego rozróżnienia pomiędzy oprogramowaniem a rozwiązaniami technicznymi i cechami zewnętrznymi urządzenia. Wiadomo bowiem, że podobieństwo programów może być spowodowane:
— zwyczajowymi sposobami przedstawiania pewnych obrazów (np. wizerunki używane w grach) lub powszechnością stosowania określonej symboliki (np. przedstawienie rachunków w systemach księgowania)
— architekturą mikroprocesorów lub cechami urządzeń wejścia-wyjścia
— zastosowaniem algorytmów lub metod matematycznych występujących w opracowaniach z danej dziedziny wiedzy.

Decyzja eksperta była jednak jednoznaczna:

- stopień podobieństwa złożonych programów napisanych w języku wewnętrznym, niewielka liczba różnic i ich forma nie pozwalają sądzić, iż mamy do czynienia z dwoma różnymi oryginalnymi dziełami
- porównanie dostarczonych wydruków dowodzi, że oprogramowanie GOLEMA jest nieznacznie zmodyfikowaną wersją oprogramowania komputera APPLE
- oprogramowanie podstawowe obu komputerów jest w 99,7% instrukcji identyczne; bardzo nieznaczne różnice dotyczą słowa wyświetlanego przy włączeniu napięcia (6 bajtów różnych) i procedury sterowania ekranem (23 bajty różne)
- opisane podobieństwa nie dają się wytłumaczyć ani zwyczajowymi normami programowania, ani użyciem mikroprocesora 6502, ani ewentualnym dążeniem do kompatybilności.

Na podstawie tej opinii firma APPLE ponowiła swoje żądania. Pod silnym wpływem stanowiska eksperta sąd uwzględnił pozew firmy i uznał słuszność jej roszczeń. Jednakże przewodniczący Trybunału Handlowego w Paryżu, wziąwszy pod uwagę skutki finansowe ogłoszonego zakazu sprzedaży, zażądał od APPLE złożenia gwarancji w wysokości 200 tys. franków na wypadek konieczności wypłacenia odszkodowania firmom objętym zakazem.

Można było bez trudu przewidzieć, że ogłoszony wyrok nie zakończy sprawy. Z jednej strony firma APPLE pozwała pozostałe firmy, chcąc uzyskać ostateczne stwierdzenie podstaw prawnych swoich roszczeń, z drugiej zaś — pozwane przedsiębiorstwa podjęły postępowania odwoławcze w stosunku do decyzji sądu.

Już w pierwszym postępowaniu przewodniczący przyjął jako zasadę, że „oprogramowanie podstawowe komputera jest utworem w rozumieniu artykułu 2. prawa autorskiego, tj. jego oryginalne opracowanie i koncepcja noszą piętno swego autora, z uwzględnieniem tego, co jest wymuszone przez zdrowy rozsądek". Na podstawie tego stwierdzenia sędziowie rozpatrujący meritum sprawy stwierdzili bardziej precyzyjnie: „Jeśli nawet programy komputerowe nie są bezpośrednio przyswajalne przez człowieka, tak jak dzieła literackie czy muzyczne, to są jednak zrozumiałe i dostępne dzięki transkrypcjom na różne nośniki materialne — wydruki, ekrany, zapisy magnetyczne. Jeżeli w istocie lektura programów nie jest dostępna wszystkim i wymaga szczególnej techniki, to ta specyficzna własność nie może ich wykluczać z kategorii utworów. Na przykład, kompozycje muzyczne są również wyrażone w zakodowanym i złożonym języku, którego bezpośrednie rozumienie wymaga specjalistycznego wykształcenia. Programy komputerowe stają się zrozumiałe za pośrednictwem pewnego instrumentu — komputera, który ujawnia je niewtajemniczonym, tak jak głos lub każdy instrument muzyczny ujawnia zawartość partytur". Po tym poetyckim stwierdzeniu sąd zajął się problemem oryginalności i — wykorzystując opinię rzeczoznawcy — oświadczył, że „doświadczenie wskazuje, iż programy spełniające te same funkcje, a zrealizowane przez różnych programistów wykazują liczne różnice, nawet jeżeli są proste, krótkie i napisane w języku wysokiego poziomu".

Tak więc w przypadku oprogramowania podstawowego duża liczba podobnych instrukcji jest dowodem wykonania mniej lub bardziej dosłownej kopii. Trybunał zdecydowanie oddzielił pojęcie oprogramowania od pojęcia sprzętu. Sędziowie stwierdzili: „Sposób produkowania i zawartość materialna układów scalonych określa je jako dobra przemysłowe. Ale ich zawartość stanowiąca o oryginalności jednych w stosunku do drugich nie jest niczym innym jak wyrażeniem w zaawansowanej technologii oryginalnej koncepcji autora programu". Zatem — zgodnie z tym orzeczeniem — programy zapisane w pamięciach ROM i RAM podlegają ochronie na podstawie prawa o własności literackiej i artystycznej.

Do podobnych wniosków doszło sądownictwo amerykańskie. Decyzją Federalnego Sądu Apelacyjnego w Filadelfii, programowanie zostało uznane za dziedzinę twórczości — sytuującą się pomiędzy literaturą a wynalazkiem. Jeżeli decyzja będzie utrzymana, wyeliminuje ze sprzedaży wiele tanich komputerów osobistych z „pirackim" oprogramowaniem.

Amerykańscy prawnicy zawsze przyznawali, że programy przechowywane na oddzielnych dyskietkach, które należy fizycznie przyłączyć do maszyny, stanowią przedmiot ochrony prawnej. Kontrowersje wzbudzały jednak zapisy (np. systemów operacyjnych) utrwalane w układach (kostkach) konstrukcji komputerów.

Stwierdzenia te pojawiły się na tle procesu pomiędzy firmami APPLE i FRANKLIN z Filadelfii. APPLE stwierdziła, że FRANKLIN skopiował czternaście programów komputera ACE 100. Argumenty oskarżenia, że system nie był chroniony przez prawo autorskie, zdołały przekonać sąd pierwszej instancji. Dopiero podczas procesu apelacyjnego stwierdzono, że żaden program nie może być powielany bez zgody autora, nawet jeżeli kopiowanie dotyczy fragmentu sprzętu: „Prosty fakt, że system operacyjny może być zapisany w pamięci ROM (ang. read-only-memory) nie czyni z tego programu urządzenia, części urządzenia, ani jego odpowiednika".

W styczniu 1983, Don Edwards, kalifornijski kongresman, który reprezentuje wyborców m.in. z Silicon Valley, przedstawił projekt ustawy „o ochronie kostek półprzewodnikowych i masek przed nieautoryzowanym kopiowaniem". Po tym wydarzeniu i wpłynięciu analogicznego projektu do senatu, amerykańskie firmy mają nadzieję, że uda im się uzyskać ochronę prawną dla swoich wyrobów.

Międzynarodowa Organizacja Własności Intelektualnej (WIPO) przy ONZ rozpoczęła we wrześniu 1983 prace nad programem działań zmierzających do powstania międzynarodowej konwencji o prawie autorskim chroniącym kostki i obwody scalone. Zakończenie prac przewiduje się w roku 1985, ale Amerykanie mają nadzieję, że termin ten uda się przyspieszyć.

Precedens FRANKLINA rozwiązuje problem na kilka lat. Jeżeli nawet sąd apelacyjny wydałby później inny werdykt, to firmy elektroniczne i tak czują się w tej chwili na tyle bezpieczne, by zapisywać programy na kostkach. W takiej sytuacji należy przewidywać gwałtowny wzrost efektywności komputerów osobistych, ponieważ odtwarzanie danych z kostek jest około tysiąc razy szybsze niż z dyskietek.

Opracowali: KATARZYNA ISAAK i MAREK SOBCZYK

na podstawie: INFORMATIQUE MENSUEL, grudzień 1983 — styczeń 1984, oraz NEW SCIENTIST, 15 września 1983

Urządzenie do automatycznej analizy fotografii nieba

Automatyczna analiza fotografii nieba wymaga rozwiązania wielu problemów. Jeden z nich to fakt, że zaciemnienie emulsji kliszy fotograficznej nie jest wprost proporcjonalne do jasności obiektu. Podczas eskpozycji astronom umieszcza w rogu zdjęcia płytki skalujące z fotografiami źródła światła o znanej intensywności. Komputer porównuje z nimi każdy badany fragment kliszy. Najczęściej dokonuje się pomiarów kliszy używając bardzo wąskich strumieni światła. Analiza za pomocą szerszych strumieni wymaga stosowania specjalnie skonstruowanego sprzętu komputerowego do wyznaczania łącznej jasności małych elementów obrazu. Wykonanie tego zadania jest niezwykle czasochłonne dla normalnych maszyn cyfrowych.

Zobaczymy jak urządzenie działa w praktyce. Klisza jest umieszczana na stole, operator podaje informacje o położeniu na niebie środka fotografii a system — posługując się katalogiem gwiazd — ustawia zdjęcie zgodnie ze współrzędnymi astronomicznymi. Następnie mierzy się jasność tła. Dokonuje tego specjalny procesor, badający każdorazowo jedną czwartą milimetra kwadratowego powierzchni zdjęcia. Z takiego obszaru pobiera 64 × 64 próbki i liczy, ile z nich odpowiada danemu natężeniu promieniowania. Ponieważ przeważająca część próbki odzwierciedla „tło nieba", poziom tła jest poziomem najczęściej występującego natężenia światła. Obliczenia są dokonywane dla każdego analizowanego fragmentu nieba — jest to szczególnie ważne przy pomiarach światła gwiazd widocznych na tle galaktyk. Po przejrzeniu całej kliszy specjalne urządzenie określa w przedziałach półmilimetrowych tło między mierzonymi punktami i tworzy mapę zawierającą dane o wartości tła w każdym punkcie kliszy.

Następuje ponowne przeglądanie fotografii. Wyznacza się wartość progową, nieco powyżej poziomu tła i odrzuca się wszystkie punkty o jasności mniejszej niż progowa. Pomiary galaktyk są dokonywane bardzo blisko poziomu tła, a zatem przypadkowe wahania gęstości emulsji na kliszy powodują na zdjęciu nieoczekiwane olbrzymie zakłócenia (szumy). Zmiany jasności zależne od ułożenia drobin emulsji są jednak mniejsze od wywołanych przez fotografowane obiekty. Filtr elektroniczny odrzuca ponad 90 proc. „fałszywych" danych bez wpływu na „prawdziwy" obraz. Informacje są przekazywane do procesora, który przegląda linia po linii jednowymiarowe wykresy obliczając położenie obiektów i intensywność promieniowania. W tym czasie inny procesor analizuje na wszystkich wykresach linie zawierające dane o określonym obiekcie. Procesor ten przygotowuje dane wejściowe dla komputera o dużej szybkości, który tworzy dwuwymiarowy obraz wszystkich obiektów podając ich wzajemne położenie i całkowitą jasność. Maszyna ma dostęp do prawie wszystkich informacji dających się odczytać z kliszy i analizuje je zgodnie z życzeniami astronomów. Dane są zapamiętywane na taśmie magnetycznej i przetwarzane później wsadowo na dużym komputerze. **(K.I)**

Komputer HP 71B

Już dziesięć lat minęło od wprowadzenia na rynek pierwszego komputera osobistego — wtedy nazywanego kalkulatorem programowym — HP 65, który kosztował wówczas (1974 r.) około 700 dolarów. W dziesiątą rocznicę tego wydarzenia HEWLETT-PACKARD wypuścił coś, co chyba trzeba by nazwać „kalputerem osobistym". Oto krótki opis techniczny przenośnego komputera osobistego drugiej generacji lub — jak kto woli — kalkulatora programowanego trzeciej generacji o symbolu HP 71B (dlaczego **B** tego nikt nie wie... — może BASIC?) w cenie 525 dol. (ale to są dolary AD 1984, mniej więcej 2,5 raza mniej warte niż tez roku 1974!).

Wymiary HP 71B wynoszą 19×10×2,5 cm, ciężar ok. 350 g (bez czytnika kart i modułu HP-IL), zasilanie przez cztery baterie „paluszki" (nie ładowane) lub zasilacz sieciowy. Procesor CMOS typu custom, 4-bitowy (tak, tak, HEWLETT-PACKARD wcale nie fascynuje się procesorami 16- lub 32-bitowymi...). Komputer ma 56 klawiszy typu kalkulatorowego, a układ klawiatury typu QWERTY plus wydzielona część z cyframi. Wyświetlacz typu LCD 1 — liniowy, 22 znaki lub (uwaga!) 8 na 132 punkty sterowane programowo. Można wyświetlać standardowe znaki ASCII lub 128 znaków definiowanych przez użytkownika.

Podstawowy zestaw HP 71B ma 17,5 KB pamięci RAM z możliwością rozszerzania o moduły pojemności 4 KB (łącznie o cztery moduły). System operacyjny ma do dyspozycji 64 KB, nie licząc komend HP-IL. Maksymalna konfiguracja może wynosić 448 KB pamięci RAM/ROM — zamiast modułów RAM można przyłączać gotowe programy w modułach ROM o pojemności po 64 KB.

Istnieje możliwość redefiniowania klawiatury, jak też przyłączenia programów (funkcji) do konkretnych klawiszy. Zestaw funkcji poszerzono o zegar i trzy stopery, przy czym zegar może być korygowany programowo. System operacyjny zawiera BASIC — istnieje możliwość pracy z modułem ROM zawierającym FORTH lub ASSEMBLER. HP BASIC ma niewiele wspólnego ze standardowym językiem BASIC: ponad 240 funkcji, możliwość rekurencyjnego definiowania procedur (à la PASCAL) oraz etykiety liczbowe (à la FORTRAN), dynamiczne wymiarowanie macierzy oraz wiele typów plików (BASIC, LEX, data, assembly, text, klawisze). Umożliwia to naprawdę efektywne programowanie tego przecież na pewno już nie kalkulatora. Choć możliwość włożenia czytnika kart magnetycznych oraz niewielkie wymiary czynią HP 71B nieco podobnym do kalkulatora programowanego drugiej generacji, czyli do HP 41C.

Posiadaczy tego ostatniego spieszę uspokoić, że dzięki HP-IL możliwe jest przenoszenie danych (plik Sdata) — HP 71B jest pierwszym sterownikiem HP-IL, który może przekazać kontrolę nad pętlą innemu urządzeniu sterującemu! Umożliwia to połączenie ponad 900 egzemplarzy HP 71B w jedną sieć i sekwencyjne opracowywanie wyników obliczeń. Można też połączyć go z komputerem osobistym pierwszej generacji HP 75C.

Dobra wiadomość dla wszystkich amatorów przeróbek, poprawek i zmian — producent (po raz pierwszy w swej historii!) dostarcza pełny opis działania systemu operacyjnego HP 71B, jego architektury oraz implementacji algorytmów. Już nie trzeba będzie zgadywać skąd biorą się różne anomalie w pracy komputera. HP wyciągnął wnioski z „odkrycia" tzw. syntetycznego programowania HP 41C (poziom bajtów pamięci, który pozwala na syntezowanie nowych funkcji) i od początku ułatwił zabawę wszystkim hobbystom. Należy oczekiwać, że takie podejście do klienta zaowocuje lawiną bardzo dobrego oprogramowania — w przypadku HP 41C właśnie dzięki syntetycznemu programowaniu udało się znacznie rozszerzyć możliwości oprogramowania tego skromnego przecież komputera. A firma sprzedała już ponad 700 tysięcy egzemplarzy i ciągle ma olbrzymi pakiet zamówień.

Osobnej wzmianki wymaga tryb pracy CALC — jest to połączenie dotychczas używanego przez HP systemu RPN z systemem AOS, czyli notacji polskiej z notacją algebraiczną. Dokonano tego bardzo prosto: całe wyrażenie, które ma zostać obliczone, jest zapamiętywane w wielopoziomowym stosie. Kolejne zmiany parametrów (liczb, funkcji itd.) są uproszczone, gdyż wystarczy tylko odszukać właściwe miejsce i dokonać poprawki — wynik wyświetlany jest natychmiast, a wyrażenie dalej pamiętane. Oczywiście, cała pamięć HP 71B jest typu CMOS, a więc zarówno programy, jak i wszelkie dane są zachowywane w pamięci nawet po wyłączeniu komputera.

O zaletach opisanego sprzętu można by więcej napisać, ale przede wszystkim chciałoby się mieć możliwość jego zakupu w Polsce. Na razie pocieszmy się, że producent dopiero rozważa możliwość sprzedaży tego modelu w Wielkiej Brytanii. Znając dotychczasową politykę HP, można przypuszczać, iż cena europejska będzie niemal dwukrotnie wyższa niż amerykańska. A swoją drogą — jak daleką drogę przebył poczciwy kalkulator programowany w ciągu tych dziesięciu lat!

Opracował J.T.

na podstawie materiałów HP

BCS

British Computer Society (BCS — Brytyjskie Towarzystwo Informatyczne) założono w październiku 1957, w wyniku rozwijających się kontaktów pomiędzy dwiema nieformalnymi grupami użytkowników informatyki, reprezentującymi zastosowania ekonomiczne i naukowe. W owych czasach komputery były jeszcze wielką rzadkością; zapotrzebowanie na nie rodziło się z doświadczeń tych, którzy mieli już z nimi do czynienia. Założyciele sformułowali następujące cele Towarzystwa:

— rozwój i zwiększenie wykorzystania sprzętu komputerowego i technik z nim związanych
— ułatwienie wymiany informacji i poglądów oraz informowanie opinii publicznej o informatyce
— organizowanie konferencji i spotkań
— publikowanie informacji na użytek członków Towarzystwa.

Jedynym istotnym rozszerzeniem tych celów było wprowadzenie w 1968 roku stopni zawodowych.

Z biegiem lat BCS stało się głównym reprezentantem informatyków w Wielkiej Brytanii i zrzesza obecnie ponad 25 tys. członków. Dlatego też Towarzystwo ma liczący się głos w takich sprawach, jak normy, kształcenie, banki danych o zasięgu krajowym itd. Utrzymuje także ścisłe kontakty z podobnymi organizacjami za granicą, np. z ACM (USA), ACS (Australia), CIPS (Kanada), CSI (Indie), AICA (Włochy), DARA (RFN), AFCET (Francja) i IFIP.

Członkiem BCS może być każda osoba związana bezpośrednio lub pośrednio z informatyką. Każdy członek zobowiązuje się do przestrzegania dwóch kodeksów formułujących zasady postępowania informatyków oraz zasady ich dobrej pracy.

Towarzystwo jest również poważnym wydawcą periodyków i książek informatycznych. COMPUTER JOURNAL to jeden z najbardziej liczących się w świecie kwartalników, zawierający prace o szczególnie wysokim poziomie — poświęcone naukowym, użytkowym i ekonomicznym aspektom informatyki — oraz przegląd najważniejszych publikacji z tej dziedziny. COMPUTER BULLETIN, wydawany także co kwartał, stanowi pewnego rodzaju uzupełnienie COMPUTER JOURNAL, zajmując się informatykami, ich opiniami oraz będąc forum dyskusji o roli informatyki i jej wpływie na współczesne społeczeństwo. Tygodnik COMPUTING publikuje bieżące informacje o BCS. Oprócz tego Towarzystwo wydaje zbiory referatów ze swoich konferencji oraz autoryzuje podręczniki, które są dostępne dla jego członków ze zniżką 25%. Członkowie mają również do dyspozycji bogatą bibliotekę książek i czasopism informatycznych.

The British Computer Society
13 Mansfield Street
LONDON, W1M OBP
Wielka Brytania
Telefon (01) 637-0471

Oprac. MkS

Komputerowe okno na świat

Świat komputerów, tak jak moda damska, zmienia się co sezon, wykazując wyraźne dążenie, by raczej coś ukryć niż odsłonić — stwierdza filozoficznie autor artykułu w listopadowym numerze amerykańskiego tygodnika TIME. Programy roku 1982 były „przyjazne", ale często zupełnie niezrozumiałe dla użytkowników. Oprogramowanie w roku 1983 jest „zintegrowane", co oznacza, że czasem udaje się jakiemuś programowi wykorzystać wyniki innego. Mądrzy ludzie zastanawiają się na wyborem lapidarnego określenia dla komputerów roku 1984.

Dwie wiodące firmy zajmujące się oprogramowaniem, MICROSOFT i VISICORP proponują sprzęt z „okienkami" — systemy pozwalające uruchamiać jednocześnie kilka programów, wyświetlanych na oddzielnych częściach monitora ekranowego. Kalifornijski VISICORP, autor odnoszącego sukcesy rynkowe programu rozliczeń finansowych VisiCalc, rozpoczął w październiku 1983 sprzedaż pakietów VisiOn. W tym samym czasie najpopularniejszy producent oprogramowania, MICROSOFT (wartość sprzedaży w 1983 roku — 100 mln dolarów), zapowiada analogiczny sprzęt o nazwie WINDOWS. Kierownik firmy, Bill Gates, stwierdza: „Jest to kamień milowy w dziedzinie oprogramowania".

Każdy program wyświetla swoje dane na „myszce" wielkości pudełka zapałek. Pozwala to użytkownikowi podzielić ekran monitora na prostokątne bloki, nadając mu wygląd pulpitu pokrytego zapisanymi kartkami papieru. Proponowane rozwiązania znów wywołują rozważania o dwóch podstawowych problemach, z którymi boryka się przemysł komputerowy: jak uruchamiać dany program na sprzęcie wyprodukowanym przez różne firmy i jak w prosty sposób przekazywać dane między różnymi programami. Obecnie — na przykład — oprogramowanie IBM nie jest dostosowane do maszyn APPLE, a wielu użytkowników systemów rozliczeń finansowych nie może bez kłopotu przesyłać klientom danych za pomocą „elektronicznej poczty".

Propozycje CISICORP i MICROSOFT nie są wcale nowe. W rzeczywistości pierwszą elektroniczną „myszkę" stworzyli już w połowie lat sześćdziesiątych naukowcy ze Stanford Research Institute. XEROX oferował w roku 1981 system komputerowy STAR, posługujący się „okienkami". Następna, już bardziej wyrafinowana realizacja tego pomysłu to LISA firmy APPLE. Rozwiązanie okrzyknięto w styczniu 1982 roku wielkim sukcesem technologicznym. Sukcesu rynkowego jednak nie było, z powodu wysokiej (początkowo — 10 tys. dolarów) ceny i słabego zainteresowania klientów. Rok później IBM wyprodukował monitor za 5,5 tys. dolarów o podobnych cechach, do współpracy z wielkim komputerem. Kalifornijska firma QUARTERDECK OFFICE SYSTEMS zapowiedziała sprzedaż DesQ — programu za 395 dolarów, wyświetlającego jednocześnie na ekranie monitora przebieg ponad dziesięciu programów.

Jednakże prawdziwa walka o pierwszeństwo rozegra się pomiędzy najstarszymi rywalami — VISICORP i MICROSOFT. W roku 1975, wkrótce po opanowaniu rynku przez mikrokomputery, dziewiętnastoletni wówczas Bill Gates otrzymał pozwolenie opuszczenia Harvardu, by zająć się opracowaniem i sprzedażą pierwszego programu umożliwiającego zwykłemu użytkownikowi oprogramowanie komputera osobistego. Dzisiaj BASIC firmy MICROSOFT działa na prawie każdym sprzęcie — od APPLE po RADIO SHACK. Kiedy cztery lata temu IBM rozpoczynała produkcję komputerów osobistych, poproszono Gates'a o stworzenie oprogramowania dla ich nowego modelu. W rezultacie powstał system dyskowy o cechach uniwersalnego standardu. MICROSOFT odnosiła również sukcesy sprzedając gry i specjalistyczne oprogramowanie.

Dan Fylstra, prezes VISICORP, był w 1978 roku studentem Harvard Business School, kiedy usłyszał o nowym programie planowania gospodarczego, napisanym przez dwóch absolwentów Massachusetts Institute of Technology. Namówił ich, by przystosowali swoje dzieło do wprowadzanego właśnie na rynek komputera firmy APPLE i zajęli się reklamą przedsięwzięcia. VisiCalc stał się w ten sposób najpopularniejszym programem jaki kiedykolwiek napisano. Od roku 1979 sprzedano ponad 600 tysięcy jego kopii. VISICORP stało się firmą o rocznym obrocie rzędu 45 mln dolarów. Jednakże ostatnio traci swe wpływy z powodu coraz liczniejszej konkurencji, której przewodzi MICROSOFT.

Nowe propozycje obu firm są zaskakująco podobne do siebie, ale drogi ich powstania były bardzo różne. VISICORP poświęciło trzy lata pracy i 12 mln dolarów na stworzenie złożonego systemu „okienkowego" (cena detaliczna 495 dolarów) i działających w nim programów (500 dol. za pierwsze dwa). MICROSOFT postanowiło zająć się tylko budową podstawowego sprzętu, przekazywanego producentom za ok. 200 dol. Firma liczy na to, że różni programiści zajmą się przystosowaniem swojego oprogramowania do nowego systemu. MICROSOFT ma około pół roku opóźnienia w stosunku do VISICORP, ale uzyskała poparcie ze strony 23 producentów, w tym firm: HEWLETT-PACKARD, DIGITAL EQUIPMENT i RADIO SHACK, które zgodziły się dołączać WINDOWS do swoich komputerów.

Nie wiadomo, które rozwiązanie okaże się lepsze, ale jedno nie ulega wątpliwości — VISICORP i MICROSOFT wyznaczyły nowy kierunek w rozwoju oprogramowania. (ki)

W SKRÓCIE

■ W dniach 15—16 marca 1984 w Rydzynie koło Leszna odbyła się krajowa konferencja COMPCONTROL-84 poświęcona zastosowaniom komputerów w przemyśle. Obok podstawowej tematyki i pokazu sprzętu (m.in. sterowniki polskiej produkcji), na konferencji podsumowano udział grupy polskiej w międzynarodowej konferencji COMPCONTROL-83, która odbyła się w Bratysławie we wrześniu 1983. Zaprezentowano też propozycje referatów na konferencję w Budapeszcie w 1985 roku. Z materiałami z konferencji krajowej można zapoznać się w Ośrodku Doskonalenia Kadr SIMP w Warszawie. (AS)

■ Trzej czołowi europejscy producenci sprzętu komputerowego: brytyjski ICL, francuski BULL i zachodnioniemiecki SIEMENS postanowili utworzyć wspólny instytut naukowy dla realizacji badań w dziedzinie sztucznej inteligencji. Instytut ten ma rozpocząć działalność na początku br. i zatrudniać ok. 50 pracowników naukowych. Jego podstawowym zadaniem będzie realizacja projektu o kryptonimie ESPRIT wg założeń przygotowanych przez Europejską Wspólnotę Gospodarczą. Warto przypomnieć, że na początku lat siedemdziesiątych podobna trójka producentów europejskich — SIEMENS, BULL (wówczas pod nazwą CII-HB) oraz holenderski PHILIPS — podpisała porozumienie mające na celu ścisłą współpracę w dziedzinie projektowania konstrukcji sprzętu oraz wspólną produkcję rodziny komputerów UNIDATA. Porozumienie to zakładało ujednolicenie wyrobów oraz koncentrację produkcji sprzętu komputerowego i oprogramowania celem bardziej skutecznego przeciwstawienia się niszczącej konkurencji producentów amerykańskich. Po początkowych spektakularnych sukcesach porozumienia zostało w 1975 r. rozwiązane ze względu na znaczne rozbieżności poglądów na temat kierunków dalszego rozwoju sprzętu. Aktualny trzeci partner nowego porozumienia — firma ICL — mimo silnych nacisków i najgorszej może sytuacji rynkowej, nie przystąpił do poprzedniego porozumienia, ponieważ uzyskał wtedy znaczne subwencje od rządu brytyjskiego, ratujące go na kilka lat od nieuchronnego bankructwa. Należy przypuszczać, że obecnie ujawnione porozumienie zostało prawdopodobnie sprowokowane przez japońskie projekty komputerów V generacji, uwzględniające, jak wiadomo, wprowadzenie rozwiązań opartych o koncepcję sztucznej inteligencji. (K)

■ Rząd francuski sytematycznie zwiększa wydatki budżetowe na badania w dziedzinie informatyki i mikroelektroniki. Minister przemysłu udzielił ostatnio informacji, że nakłady na ten cel wzrosły w 1983 r. do kwoty ok. 8 mld franków (ok. 1 mld dol.) i były o 29% większe niż w roku poprzednim. Głównym akcentem planów średnioterminowych będzie rozwój produkcji elementów elektronicznych na potrzeby zarówno sprzętu komputerowego, jak i wyrobów konsumpcyjnych. Łączne nakłady inwestycyjne na informatykę i elektronikę w ciągu najbliższych pięciu lat wyniosą we Francji ok. 140 mld franków (ok. 17,5 mld dol.). (K)

W SKRÓCIE

Dokumentacja oprogramowania (2)

Po przerwie spowdowanej koniecznością dostosowania treści rubryki terminologicznej do tematyki mikroprocesorowej, w bieżącym numerze wracamy do omawiania pojęć związanych z dokumentacją oprogramowania. W pierwszej części artykułu omówiono cztery dokumenty powstające w cyklu istnienia oprogramowania, lecz nie przeznaczone bezpośrednio dla użytkowników programu — specyfikację wymagań, specyfikację projektu wstępnego i szczegółowego oraz plan testowania. Poniżej opisano pozostałe dokumenty stanowiące przedmiot dostawy, przeznaczone dla użytkowników.

Podstawowym dokumentem niezbędnym użytkownikowi programu jest **podręcznik użytkowania** (ang. user manual) czyli dokument, który zawiera informację niezbędną do wykonywania i użytkowania programu. Opisuje on funkcje programu i zawiera wzorce służące do przygotowania danych wejściowych, ustalenia parametrów i interpretacji wyników. Powinien zawierać opis zasobów niezbędnych do eksploatacji programu oraz opis procedur przygotowania danych wejściowych i wykonania programu.

Podręcznik użytkowania jest naprawdę użyteczny dopiero po zainstalowaniu i przetestowaniu programu, a więc — podczas eksploatacji. Jednak pełna dokumentacja oprogramowania powinna umożliwić użytkownikowi nie tylko wykonywanie programu, ale także jego instalowanie, modyfikowanie i testowanie.

Instrukcja instalowania (ang. installation instruction manual) jest to dokument, który opisuje procedury instalowania programu na określonym komputerze. Jej celem jest dostarczenie użytkownikowi wystarczającej informacji do zainstalowania programu zapisanego na taśmie magnetycznej, na jego własnym systemie komputerowym. Zawiera opis struktury programu, jego formatu na taśmie źródłowej, języka poleceń niezbędnego do zainstalowania programu i testów służących do zweryfikowania poprawności instalacji.

Podręcznik programisty (ang. program reference manual) jest dokumentem, który opisuje szczegółowo strukturę programu na poziomie umożliwiającym modyfikowanie i pielęgnowanie kodu. Jego celem jest dostarczenie programistom informacji wystarczających do zrozumienia programu, jego środowiska eksploatacyjnego i procedur pielęgnacji. Zawiera wszystkie szczegółowe informacje, dotyczące kodu źródłowego, wymagane do pielęgnowania i modyfikowania go przez osoby inne niż te, które opracowały program.

Sprawozdanie z testowania (ang. test analysis report) jest dokumentem, który opisuje wykonywane testy, analizę wyników testowania i działania podejmowane w przypadku uzyskania negatywnych wyników. Jego celem jest przedstawienie wyników testowania i ich analizy, wynikających stąd możliwości i ograniczeń programu, oraz — dostarczenie podstaw do stwierdzenia o przygotowaniu oprogramowania do eksploatacji. Sprawozdanie z testowania zawiera niezbędną liczbę szczegółów potrzebnych użytkownikowi do załadowania i wykonania opisanych programów testujących. Użytkownik powinien mieć możliwość porównania wyników przebiegu własnych testów z wynikami zawartymi w sprawozdaniu. Dzięki temu może ocenić przydatność programu do własnych potrzeb, jak również nabrać przekonania co do jego rzeczywistych możliwości.

→

Trojniar W.: FORTH — język i system programowania (3)

INFORMATYKA 1984, nr 8, s. 1

Ostatnia część charakterystyki języka i systemu programowania FORTH, zawierająca sposoby implementacji oraz definiowania nowych słów, a także przykład pełnego programu.

Zieliński C.: Roboty (3). Proste języki programowania robotów

INFORMATYKA 1984, nr 8, s. 6

Trzecia część problematyki robotów, zawierająca omówienie metod ich programowania. Na przykładach najbardziej znanych rozwiązań scharakteryzowano grupę prostych języków programowania.

Dańda J., Poznański Z.: Programowanie w języku PL/M (2)

INFORMATYKA 1984, nr 8, s. 9

Druga część charakterystyki uniwersalnego języka wysokiego poziomu PL/M, przeznaczonego dla mikrokomputerów. W oparciu o przykłady omówiono złożone konstrukcje tego języka.

Тройняр В.: FORTH — язык и система программирования (3)

INFORMATYKA 1984, № 8, стр. 1

Последняя часть характеристики языка и системы программирования FORTH, содержащая описание способов реализации и определения новых слов, а также пример полной программы.

Зелиньский Ц.: Роботы (3). Простые языки программирования роботов

INFORMATYKA 1984, № 8, стр. 6

Третья часть проблематики роботов, содержащая обсуждение методов их программирования. На примерах наиболее известных решений охарактеризована группа простых языков программирования.

Даньда Й., Познаньский З.: Программирование на языке PL/M (2)

INFORMATYKA 1984, № 8, стр. 9

Вторая часть характеристики универсального языка высокого уровня PL/M, предназначенного для микро-ЭВМ. На базе примеров обсуждаются сложные конструкции этого языка.

Trojniar W.: FORTH — the language and programming system (3)

INFORMATYKA 1984, No. 8, p. 1

Last part of the FORTH language and programming system characteristics, which includes implementation and new words definition methods, as well as an example of a complete program.

Zieliński C.: Robots (3). Simple programming languages for robots

INFORMATYKA 1984, No. 8, p. 6

Third part of robot's subject matter, which includes discussion of their programming. On examples of most famous solutions, the group of simple programming languages are characterized.

Dańda J., Poznański Z.: Programming in the PL/M language (2)

INFORMATYKA 1984, No. 8, p. 9

Second part of characteristics of the PL/M universal high level programming language for microcomputers. Based on illustrative examples, complex structures of the language are discussed.

Trojniar W.: FORTH — Programmiersprache und -system (3)

INFORMATYKA 1984, Nr. 8, S. 1

Letzter Teil einer Charakteristik von FORTH Programmiersprache und -system, die Implementierungsmethode, Definierung neuer Wörter und ein Beispiel des kompletten Programms, umfasst.

Zieliński C.: Roboter (3). Einfache Programmiersprachen der Roboter

INFORMATYKA 1984, Nr. 8. S. 6

Dritter Teil der Roboterproblematik, der eine Besprechung von Methoden ihrer Programmierung umfasst. Mit Beispielen bekanntester Lösungen wurde die Gruppe von einfachen Programiersprachen charakterisiert.

Dańda J., Poznański Z.: Programmierung in der PL/M Sprache (2)

INFORMATYKA 1984, Nr. 8. S. 9

Zweiter Teil einer Charakteristik von der universellen höheren Programmiersprache PL/M, die für Mikrorechner bestimmt ist. Auf Grund der Beispiele wurden zusammengesetzte Konstruktionen dieser Sprache charakterisiert.

TERMINOLOGIA

Rzadziej wymaganym dokumentem uzupełniającym jest tzw. **techniczny opis teorii** (ang. technical theory textbook), w którym przedstawia się podstawy teoretyczne i analizy niezbędne lub użyteczne do zrozumienia działania programu. Opis teorii powinien być przygotowany na poziomie nie wymagającym profesjonalnej wiedzy dotyczącej programowania, teorii na której oparto program, analizy numerycznej i — innych programów opracowanych dla tego samego zagadnienia.

Dokumentem obejmującym wszystkie informacje o programie, w ciągu jego istnienia, jest **dziennik programu** (ang. program log). Jego celem jest rejestracja wszystkich działań dotyczących określonego programu, włącznie z tworzeniem poszczególnych wersji, wykrywaniem i poprawianiem defektów oraz rozpowszechnianiem. Zawiera on historię planowania programu, jego kolejnych projektów i decyzji dotyczących implementacji, oraz — historię przebiegu testowania i modyfikacji kodu przez cały okres istnienia programu.

Ostatni dokument, nazywany **raportem końcowym** (ang. final report), stanowi rodzaj globalnego zestawienia, w którym opisuje się działania i doświadczenia w definiowaniu, opracowywaniu i testowaniu programu.

JANUSZ ZALEWSKI

SPIRALA ULAMA

Na okładce zamieściliśmy rysunek spirali ULAMA. Ci, którzy chcieliby uzyskać podobny obraz za pomocą własnego mikrokomputera, mogą skorzystać z poniższego programu.

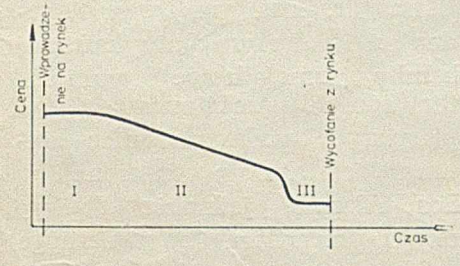
```
4 REM RYSOWANIE SPIRALI ULAMA,WIEKSZA
5 REM SPIRALA WYMAGA WIEKSZEJ MACIERZY
6 REM A,NALEZY TEZ ODPOWIEDNIO ZMIENIC
7 REM STALE W LINIACH 20,30,170,360,
8 REM 370,510 ORAZ 520.
10 DIM A(20,20)
20 FOR I=1 TO 20
30 FOR J=1 TO 20
40 A(I,J)=0
50 NEXT J
60 NEXT I
70 PR=2
80 L=1
90 X=0
100 Y=0
110 FX=0
120 FY=0
130 GOSUB 510
140 DX=1
150 DY=1
160 L=L+1
170 IF L)400 THEN GOTO 340
180 X=X+DX
190 IF L()PR THEN GOTO 220
200 GOSUB 510
210 GOSUB 440
220 IF X)0 AND ABS(X)(=ABS(FX) THEN GOTO 160
230 IF X(=0 AND ABS(X)(ABS(FX) THEN GOTO 160
240 DX=-DX
250 FX=X
260 L=L+1
270 Y=Y+DY
280 IF L()PR THEN GOTO 310
290 GOSUB 510
300 GOSUB 440
310 IF Y)0 AND ABS(Y)(=ABS(FY) THEN GOTO 250
320 IF Y(=0 AND ABS(Y)(ABS(FY) THEN GOTO 250
330 DY=-DY
340 FY=Y
350 GOTO 160
360 FOR I=1 TO 20
370 FOR J=1 TO 20
380 IF A(I,J)=1 THEN PRINT "X";
390 IF A(I,J)=0 THEN PRINT " ";
400 NEXT J
410 PRINT ""
420 NEXT I
430 END
440 PR=PR+1
450 IP=INT(SQR(PR)+.5)
460 FOR I=2 TO IP
470 M=PR-INT(PR/I)*I
480 IF M=0 THEN GOTO 440
490 NEXT I
500 RETURN
510 I=Y+10
520 J=X+10
530 A(I,J)=1
540 RETURN
```

Kupić — nie kupić

Chciałbym się zastanowić nad problemem, który prędzej czy później staje przed każdym potencjalnym posiadaczem mikrokomputera: kupić go czy jeszcze zaczekać. O ile podobny dylemat posiadaczy samochodów, kanarków czy też mebli „na wysoki połysk" jest łatwo rostrzygalny (decyduje ilość pieniędzy), o tyle w przypadku komputerów osobistych stale napotykamy dręczącą wątpliwość: a może jutro, za miesiąc, za rok pojawi się jeszcze lepszy, wygodniejszy, szybszy model XYZ?

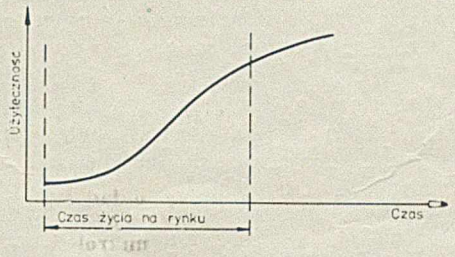
Co gorzej, dotychczasowa praktyka zdaje się potwierdzać nasze obawy — rzeczywiście, na rynku coraz częściej pojawia się nowy model, który nie tylko nie jest kompatybilny z poprzednimi modelami komputerów osobistych, do których już zdążyliśmy się przyzwyczaić, ale — co gorsze — jest on tańszy od wspaniałego modelu XYZ, który właśnie ma nam przywieźć miła ciocia z Ameryki. Niestety, przemysł mikroelektroniczny rozwija się zgodnie z przebiegiem funkcji wykładniczej, to znaczy — liczba nowych modeli komputerów jest proporcjonalna do liczby modeli znajdujących się już na rynku. Stąd absolutne pomieszanie standardów, kodów, połączeń — istna komputerowa wieża Babel. Nawet poczciwy BASIC nie jest tym samym językiem w wyrobach jednej firmy, a cóż dopiero mówić o różnych producentach!

Całe szczęście, że ceny z dnia na dzień lecą w dół — i oczywiście łatwiej kupić tanio komputer, który się producentowi nie udał, niż model naprawdę dobry. Udany wyrób sprzedaje się sam, po co więc obniżać cenę. W całym tym galimatiasie cenowym widać jednak pewną prawidłowość, którą można przedstawić na prostym wykresie:



Pierwsza faza to okres nowości — producent stara się utrzymać cenę nowości jak najwyżej, czyli zwrócić sobie koszty badań, oprzyrządowania itd. Potem następuje obniżanie ceny, bo rynek nasyca się, a konkurencja nie śpi. Wreszcie faza trzecia — gdy cena spada gwałtownie do poziomu, a często poniżej kosztów produkcji: jest to paniczna wyprzedaż zapasów przed wprowadzeniem nowego modelu lub przed... bankructwem firmy.

Można też narysować analogiczny przebieg czasowy wzrostu wartości użytkowej komputera, mierzonej ilością dostępnego oprogramowania, możliwością uzyskania porad eksploatacyjnych i oprzyrządowania pomocniczego:



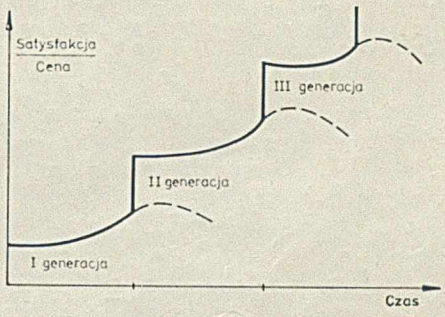
Jak widać krzywa użyteczności jest przesunięta względem krzywej cenowej — nawet po zaprzestaniu produkcji danego modelu dotychczasowi posiadacze dalej rozwijają oprogramowanie i są aktywni w różnego rodzaju klubach, zrzeszeniach itp.

A więc kiedy kupić?!

Wydaje się, że inne czynniki decydują o wyborze w krajach, gdzie cena komputera osobistego wynosi mniej niż miesięczna pensja dobrze zarabiającego fachowca, a zupełnie inne u nas, gdzie cena ta jest astronomiczna. Użyteczność mierzona ilością dostępnego oprogramowania jest — z punktu widzenia polskiego użytkownika — mało ważną cechą komputera osobistego, gdyż niewielu stać na zakupienie pełnego pakietu programów; zwykle sam użytkownik przerabia lub kopiuje cudze programy. Natomiast bariera cenowa ogranicza możliwości zakupu w dwóch pierwszych okresach sprzedaży rynkowej. Dlatego też obserwujemy teraz w Polsce masowe zakupy COMMODORE 64, który sprzedawany jest po cenie dumpingowej (w ciągu ostatnich dwóch lat staniał przeszło trzykrotnie!).

Podobnie dzieje się z wieloma innymi komputerami, nabywanymi czy to dla zabawy, czy też do poważnych prac naukowych (aczkolwiek śmiem wątpić, czy rzeczywiście używanie SINCLAIRA ZX81 w instytucie naukowym tylko dlatego, że można go kupić za stosunkowo niewielką kwotę złotówkową, jest naprawdę na dłuższą metę dobrym pomysłem...).

W ten sposób dochodzimy do problemu nowości: każda kolejna generacja komputerów osobistych jest naprawdę lepsza i bardziej wydajna, nie tylko bezwzględnie, ale też w przeliczeniu na złotówkę czy dolara. Problem, czy kupić już czy też poczekać — ma więc dodatkowy aspekt tzw. happiness to price ratio (stosunek satysfakcji do ceny):

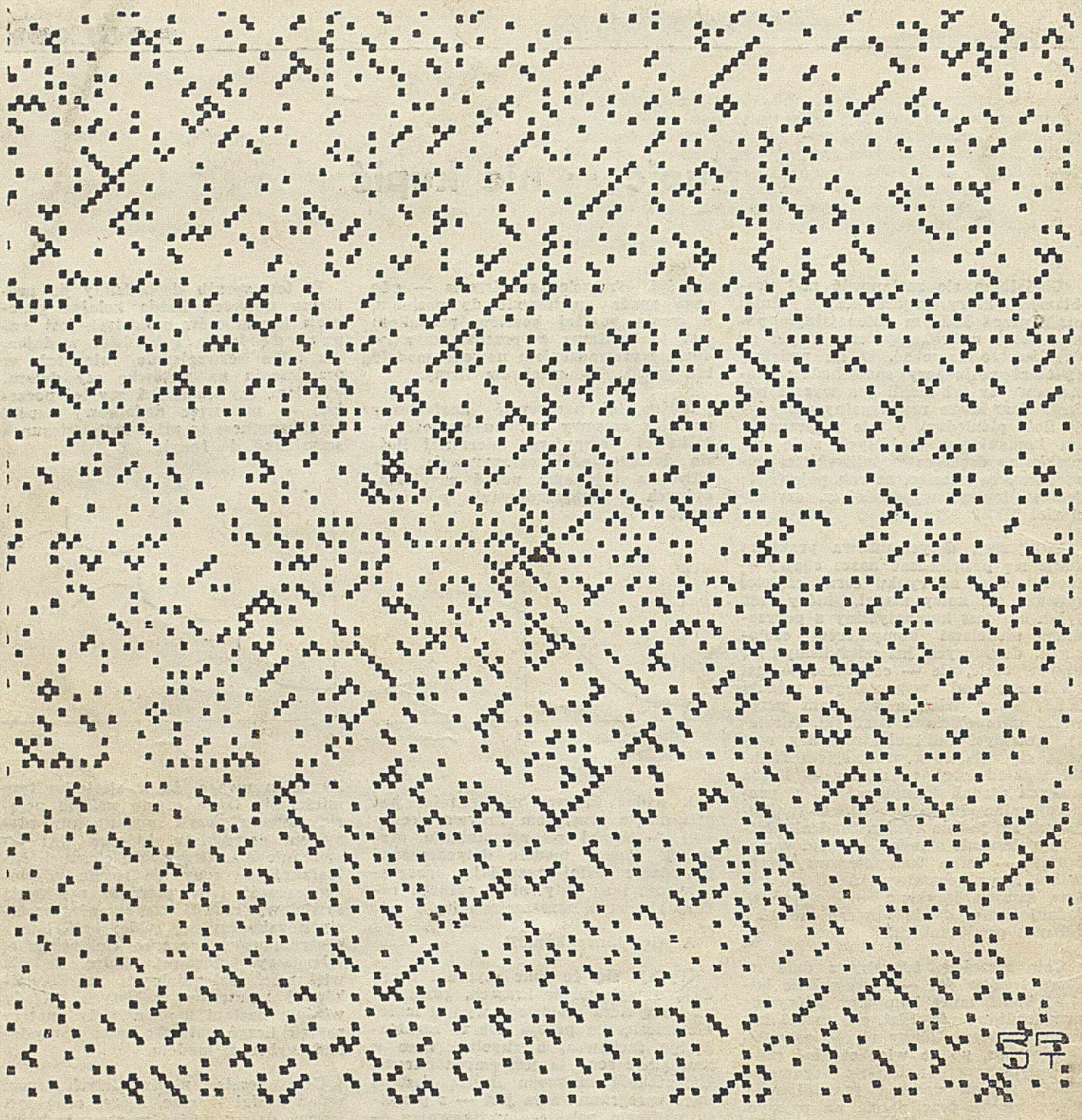


Zwracam uwagę, że w stosunku tym mieści się także czysto polskie pojęcie „szpanu", czyli imponowania zdobyciem przedmiotu, którego nikt jeszcze nie ma. Oczywiście stosunek satysfakcji do ceny nie rośnie w nieskończoność i w pewnym momencie żywiołowy rozwój komputerów osobistych zakończy się (vide: kalkulatory programowane, zegarki elektroniczne, telewizory kolorowe). Sądzę, że dla większości potencjalnych użytkowników komputerów osobistych wtedy właśnie nastąpi moment, gdy należy zacząć liczyć gotówkę oraz pomyśleć nad wyborem modelu.

Cała reszta niecierpliwych musi wziąć udział w grze, której ceną jest jednak dalszy postęp w rozwoju komputerów osobistych. Bo przecież posiadacze wczesnych modeli płacą za badania i wdrażanie do produkcji nowych generacji. Jeśli świadomość ryzyka finansowego w najdynamiczniej rozwijającym się przemyśle jest jakąś pociechą dla tych, którzy zainwestowali dużo pieniędzy w dziś już przestarzałe wyroby, to niech im kieszeń lekką będzie!

Mam nadzieję, że INFORMATYKA na łamach mikroKLANU ogłosi któregoś dnia: kupujemy komputery osobiste! Czego sobie oraz Czytelnikom życzy

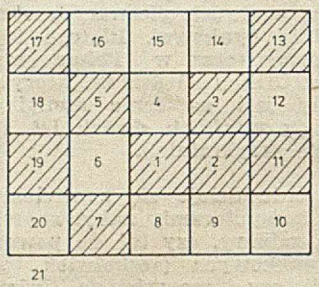
JAKUB TATARKIEWICZ



SPIRALA ULAMA

powstaje poprzez kolejne wpisywanie liczb naturalnych w kratkowaną sieć w kierunku odwrotnym do ruchu wskazówek zegara (por. SCIENTIFIC AMERICAN, marzec 1964, str. 122).

Przedstawiony rysunek, zawierający zaczernione pola liczb pierwszych w takiej sieci, został wykonany przez maszynę MERA 400 w czasie ok. 3 godzin (bardzo wolny ploter!) i zawiera ok. 2,5 tys. liczb pierwszych (rozpatrzono ponad 22,5 tys liczb naturalnych). Wydruk programu zamieszczamy na str. 32.



Legenda mówi, że Stanisław Ulam (ur. 1909 we Lwowie, jeden z twórców amerykańskiej bomby termojądrowej) wymyślił ten sposób przedstawiania liczb pierwszych podczas bardzo nudnego referatu na konferencji w Los Alamos, jednakże dopiero użycie komputera do graficznego przedstawienia większej ilości liczb pozwoliło w pełni podziwiać regularność rozłożenia liczb pierwszych (wyraźnie widoczne diagonale).

RAFAŁ PIETRAK
JAKUB TATARKIEWICZ