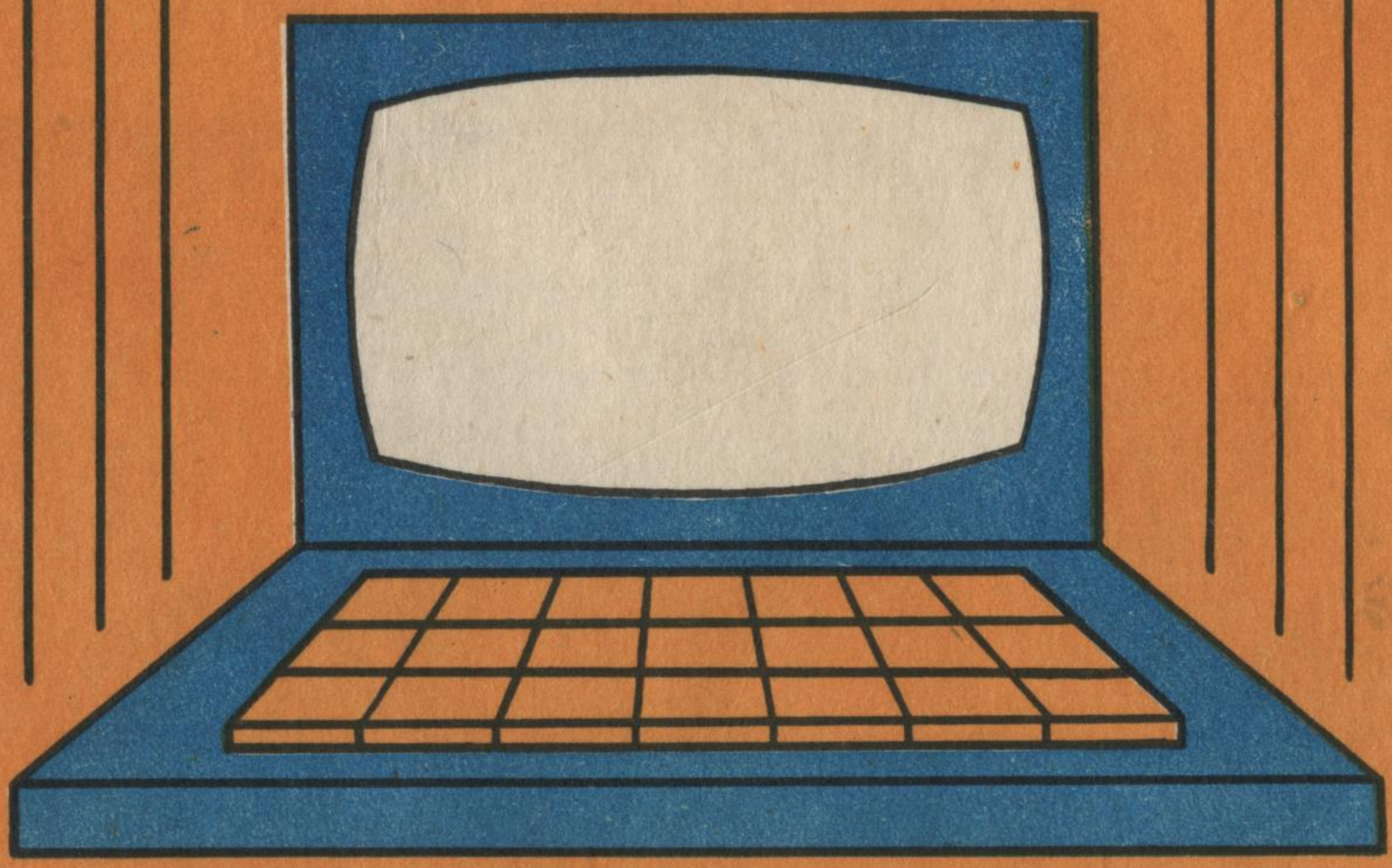


Andrzej Sirko

Basic

dla mikrokomputera COBRA 1

Wydawnictwa Naukowo-Techniczne

Basic

dla mikrokomputera

COBRA 1

Andrzej Sirko

Basic
dla mikrokomputera
COBRA 1

Wydawnictwa Naukowo-Techniczne
Warszawa 1986

Redaktor naukowy *Ryszard Pełka*
Redaktor WNT *Ewa Zdanowicz*
Okładkę projektowała *Anna Prokopowicz-Stasina*
Redaktor techniczny *Bogumił Marczak*

681.3

Książka zawiera opis obsługi mikrokomputera COBRA 1 z językiem Basic w wersji TRS-80L2. Omawiane instrukcje języka Basic zilustrowano licznymi przykładami.

Książka jest przeznaczona dla szerokiego kręgu czytelników, a zwłaszcza dla użytkowników mikrokomputerów COBRA 1, MERITUM lub TRS-80L2.

ISBN 83-204-0822-9

WNT Warszawa 1986. Wyd. I. Nakład 50 000+300 egz. Ark. wyd. 3,7. Ark. druk. 5,0. Format A-5. Papier offset. kl. III 70 g. Oddano do składania 17.XII.1985 r. Podpisano do druku w lipcu 1986 r. Druk ukończono w sierpniu 1986 r. Symbol Et/82206/WNT.
Poznańskie Zakłady Graficzne im. M. Kasprzaka — Zam. 71022/86. I-7/599.

Spis treści

Wstęp

Język Basic powstał w 1964 roku w Dartmouth College w Stanach Zjednoczonych. Został on opracowany pod kierownictwem profesorów J.G. Kemeny'ego i T.F. Kurtza. Należy do grupy języków konwersacyjnych i jest przeznaczony przede wszystkim do rozwiązywania problemów naukowo-technicznych i ekonomicznych. Struktura języka umożliwia konwersacyjne wprowadzanie tekstu programu oraz jego kolejne uruchamianie po usunięciu sygnalizowanych błędów. Prezentowany język Basic jest zmodyfikowaną wersją języka Microsoft-Basic dla TRS-80L2 stosowanego też w mikrokomputerze MERITUM.

1

Opis programu w języku Basic

Program w języku Basic składa się *z linii*. Każda linia programu może zawierać *instrukcje* realizujące poszczególne kroki algorytmu (*instrukcja czynna*) lub instrukcje przekazujące językowi Basic informacje o programie (*instrukcja bierna*). Linie programu zawierające instrukcje języka Basic muszą zaczynać się liczbami — *numerami linii*. Liczby te określają kolejność wykonywania instrukcji programu, od linii o najniższym numerze do linii o najwyższym numerze, jeżeli nie wystąpią instrukcje skoku. Kolejność wprowadzania linii programu jest dowolna. Wprowadzenie linii programu o numerze linii już wprowadzonej wcześniej powoduje usunięcie treści linii wcześniejszej i zastąpienie jej nową. Wprowadzenie tylko numeru linii powoduje usunięcie z programu linii o podanym numerze. Przy pisaniu programu warto numerować linie z pewnym krokiem, np.: 5, 10, itp. Umożliwia to w przypadku wprowadzenia zmian do programu wpisanie nowych linii bez konieczności modyfikacji linii już istniejących.

2

Alfabet języka Basic-COBRA

W każdym języku jest konieczne określenie zbioru znaków, których używanie jest dopuszczalne. Możemy podzielić je wstępnie na trzy grupy:

1. *Litery* alfabetu łacińskiego:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

2. *Cyfry:*
0 1 2 3 4 5 6 7 8 9

3. *Symbole:*

+ − / * [	(operatory arytmetyczne)
= > <	(operatory relacji)
, ; : ”	(znaki separatorów)
.	(kropka dziesiętna)
# ? $ @ % ( ) ′ ␣	(pozostałe symbole)

3

Informacje o składni języka Basic

Możemy wyróżnić następujące elementy składni języka Basic-COBRA:

a) zmienne (proste, łańcuchowe, tablicowe)
b) operatory (arytmetyczne, logiczne, relacji)
c) funkcje matematyczne
d) instrukcje:
   — wejścia-wyjścia
   — organizacyjne i sterujące przebiegiem programu
   — operacji na łańcuchach

3.1. Definicje typów zmiennych

W celu rozróżnienia wielkości występujących w programie nadaje się im nazwy. *Zmienna* więc jest nazwą, która może służyć do przechowywania wartości numerycznych lub sekwencji znaków (łańcuchów) w kodzie ASCII.

Zasady tworzenia zmiennych:

— nazwa zmiennej musi zaczynać się od dużej litery (A ÷ Z);
— nazwa musi składać się z liter lub litery i cyfry (0 ÷ 9);
— nazwa może być dwuznakowa, jak np. AA, AZ, A1, A7 itd.;
— nazwa nie może zawierać ciągu znaków będącego słowem kluczowym (tzn. nazwą funkcji, instrukcji, zlecenia itd., używanych w opisywanej wersji języka).

W przypadku podania nazwy dłuższej niż dwa znaki (np. BASIC) tylko dwa pierwsze znaki "BA" będą używane przez Basic do rozróżniania nazwy. Nazwy "CO", "COBRA", "COP" oznaczają tę samą zmienną. Wśród *zmiennych prostych numerycznych* rozróżnia się następujące typy zmiennych:

a) całkowite (*integer*)
b) pojedynczej precyzji (*single precision*)
c) podwójnej precyzji (*double precision*)

Zmienne te w zależności od wymogów programu będziemy oznaczać następującymi znakami deklaracji:

% — liczby całkowite z zakresu $-32768 \div 32767$, np. S%, A2%;
! — pojedynczej precyzji (6 cyfr znaczących), np. S!, A2!;
— podwójnej precyzji (16 cyfr znaczących), np. S #, A2 #;
D — podwójnej precyzji w notacji naukowej, np. B = 3.1499977D4.

Wszystkie zmienne bez znaków deklaracji (%, !, #) będą traktowane jako zmienne pojedynczej precyzji. Te same nazwy zmiennych z różnymi znakami deklaracji będą traktowane jako różne zmienne, np. A% = 1, A! = 2, A # = 3.

Powyższa zasada dotyczy innych funkcji, w których definicji posługujemy się zmiennymi numerycznymi.

Oprócz zmiennych numerycznych istnieje typ *zmiennej łańcuchowej* oznaczany znakiem dolara.

$ — łańcuch do 255 znaków, np. A$ = "COBRA", A2$ = "ANNA".

W przypadku konieczności wprowadzenia dużej ilości zmiennych stosuje się *macierze*. Deklarowane w zależności od rodzaju zmiennych, mogą zawierać liczby całkowite, pojedynczej precyzji, podwójnej precyzji oraz łańcuchy, np. DIM A% (7), DIM A! (3), DIM A # (11), DIM A$ (33).

Każdy element macierzy jest *zmienną indeksowaną*, a jego nazwa składa się z nazwy macierzy i indeksów umieszczonych w nawiasach. Pierwszy indeks określa *wiersz*, drugi — *kolumnę*, a trzeci *warstwę*. Istnieje możliwość deklaracji macierzy o większej ilości indeksów niż trzy. Praktycznie maksymalna liczba indeksów nie przekroczy ośmiu.

Po zainicjowaniu programu Basic przy wersji 48 kbajtów, ilość bajtów pamięci operacyjnej dostępnej dla języka Basic wynosi 31701.

Po deklaracji: DIM A (2,2,2,2,2,2,2,2), dla programu Basic zostanie tylko 5435 bajtów.

```
G:0 CR
PRINT MEM CR
31701
DIM A (2,2,2,2,2,2,2,2) CR
PRINT MEM CR
5435
```

3.2. Operatory

3.2.1. Operatory arytmetyczne

Wyrażenia arytmetyczne są budowane ze stałych, zmiennych oraz *operatorów arytmetycznych* i nawiasów. Operatory, jakie mogą występować w wyrażeniach arytmetycznych, to:

+	dodawanie
–	odejmowanie
/	dzielenie
*	mnożenie
[	potęgowanie

3.2.2. Operatory logiczne

Operatory logiczne, do których zaliczają się:

AND — iloczyn logiczny
OR — suma logiczna
NOT — negacja

służą do budowy wyrażeń logicznych, np.

```
25 IF A=7 AND B=3 THEN GOSUB 1000 CR
30 IF A=0 OR B=0 THEN 1200 CR
```

Te same operatory logiczne mogą również być użyte do wykonywania operacji boole'owskich na bajtach i bitach, jak np.:

```
10 A=133 CR
20 PRINT A AND 127 CR
RUN CR
5
```

Badanie stanów binarnych układów wejściowych np. po instrukcji INP (N) umożliwia nam obsługę tych urządzeń bez konieczności użycia procedur napisanych w języku maszynowym mikroprocesora. Podczas wykonywanych operacji logicznych operandy podlegają konwersji na 16-bitowe liczby całkowite ze znakiem ($-32768 \div 32767$). Operacje przeprowadzane są bit po bicie, jak to pokazują tablice 3.1 i 3.2.

Tablica 3.1

A AND B			A OR B			NOTA	
A	B	WYNIK	A	B	WYNIK	A	WYNIK
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1	—	—
1	1	1	1	1	1	—	—

Tablica 3.2

A	OPERATOR	B	WYNIK	KOMENTARZ
15	AND	2	2	...1 1 1 1 15 ...0 0 1 0 2 ——— ...0 0 1 0 2
15	OR	16	31	...0 1 1 1 1 15 ...1 0 0 0 0 16 ——— ...1 1 1 1 1 31
	NOT	0	−1	...0 0 0 0 0 ——— ...1 1 1 1
	NOT	2	−3	...0 0 1 0 ——— ...1 1 0 1

Uwaga. Kropki przed znakami w komentarzu oznaczają zera aż do 15 bitu. Bit 16 jest bitem znaku.

3.2.3. Operatory relacji

Operatory relacji służą do zapisu wyrażeń logicznych, które są przeznaczone do porównania wartości dwóch wyrażeń. Wynikiem tego porównania jest wartość logiczna *prawda* (*true*) lub *fałsz* (*false*).

Wyrażenia logiczne skonstruowane przy użyciu operatorów relacji mogą być wykorzystane w warunkowych instrukcjach sterujących.
Do operatorów relacji zaliczamy:

=	znak równości
< > lub > <	nie równe (różne)
<	mniejsze
<= lub =<	mniejsze lub równe (nie większe)
>	większe
>= lub =>	większe lub równe (nie mniejsze)

Podany przykład przedstawia poprawną konstrukcję wyrażenia logicznego zapisanego przy pomocy operatorów relacji i operatora logicznego w instrukcji warunkowej IF:

```
20 IF A=B AND B<C THEN GOSUB 100
```

Niedopuszczalna jest następująca konstrukcja wyrażenia:

```
20 IF A=B<C THEN GOSUB 100
```

Wymienione operatory relacji zachodzących między wyrażeniami arytmetycznymi mogą być użyte do konstrukcji wyrażeń logicznych zawierających zmienne łańcuchowe i interpretacja ich jest wtedy następująca:

=	równość łańcuchów
< >	nierówność łańcuchów
<	poprzedza alfabetycznie
<=	poprzedza alfabetycznie lub równy
>	następuje alfabetycznie
=>	następuje alfabetycznie lub równy
+	składanie łańcuchów

Przykłady relacji:

1) "ABC" < "ABD"
Łańcuch "ABC" poprzedza alfabetycznie "ABD", ponieważ litera C poprzedza literę D.

2)
```
10 A$="CO" : B$="BRA" CR
20 PRINT A$+B$ CR
RUN CR
COBRA
```

3) 10 DATA KO,T,ZA,S,BZA,BOL
20 FOR I=1 TO 6
30 READ A$(I)
40 NEXT I
50 FOR I=1 TO 5
60 PRINT A$(1)+A$(I+1)
70 NEXT I
RUN
KOT
KOZA
KOS
KOBZA
KOBOL

Przykłady powyższe ilustrują składanie łańcuchów.

3.2.4. Kolejność wykonywania działań

Biorąc pod uwagę wszystkie operatory (arytmetyczne, logiczne, relacji), kolejność wykonywania działań jest następująca:

1. [	potęgowanie
2. *,/	mnożenie i dzielenie
3. +,−	dodawanie i odejmowanie
4. <,>,=,< =, = >, < >	relacje
5. NOT	negacja
6. AND	iloczyn logiczny
7. OR	suma logiczna

Jeżeli uwzględnimy operację zmiany znaku, to jest ona w kolejności po potęgowaniu, a przed mnożeniem i dzieleniem. Jeżeli wyrażenie zawiera kilka operacji na tym samym poziomie, to będą one wykonywane od lewej strony do prawej. Kolejność działań można zmienić za pomocą nawiasów. Jako pierwsza zostanie wykonana operacja z najbardziej zagłębionej pary nawiasów.

3.3. Funkcje matematyczne

Każdy język programowania dysponuje *funkcjami matematycznymi*, które mogą być użyte w wyrażeniach arytmetycznych. Wywołanie funkcji uzyskuje się przez podanie jej nazwy oraz jej argumentu w

nawiasie. W czasie działania programu najpierw następuje obliczenie wartości argumentu, jeżeli występuje w postaci wyrażenia, a potem samej funkcji.
Poniżej zostały opisane funkcje matematyczne dostępne w tej implementacji języka Basic. Dają one wynik w pojedynczej precyzji (6 cyfr). Funkcje CINT, CDBL dają wynik o dokładności zależnej od funkcji. Funkcje ABS, FIX, INT dają wynik o dokładności zależnej od argumentu.

ABS(X) — oblicza wartość bezwzględną argumentu:

$$ABS(X) = \begin{cases} X \text{ dla } X => 0 \\ -X \text{ dla } X < 0 \end{cases}$$

Przykłady:

ABS(7) = 7 ABS(−7) = −(−7) = 7

ATN(X) — oblicza arcustangens argumentu X. Otrzymany wynik jest podany w radianach. Aby otrzymać wynik w stopniach należy pomnożyć go przez liczbę 57.29578.

CDBL(X) — wyznacza odpowiednik podwójnej precyzji argumentu X. Funkcja ta jest stosowana w celu wymuszenia operacji podwójnej precyzji na argumentach całkowitych lub argumentach pojedynczej precyzji. Wyznaczona wartość ma 16 cyfr, lecz tylko cyfry zawarte w argumencie będą znaczące.

Przykład:

```
10 FOR I%=1 TO 3:PRINT 1/CDBL(I%):NEXT I%
RUN
 1
 .5
 .333333333333333
```

CINT(X) — oblicza największą liczbę całkowitą nie większą od wartości argumentu. Argument jest ograniczony do zakresu liczb całkowitych z przedziału (−32768, 32767)

Przykłady:

CINT(−32.3) = −33 CINT(32.3) = 32

COS(X) — oblicza cosinus argumentu X. Argument X musi być podany w radianach. Jeżeli X podany jest w stopniach, to wtedy używamy następującego wyrażenia: COS(X*.0174533)

Przykład:

```
10 FOR I=1 TO 5
20 PRINT COS(I/5)
30 NEXT I
RUN
.980067
.921061
.825336
.696707
.540302
```

CSNG(X) — wyznacza z argumentu X o podwójnej precyzji jego odpowiednik pojedynczej precyzji. Funkcja zostawia 6 cyfr znaczących z zaokrągleniem typu 4/5. Jeżeli na siódmej pozycji licząc od pierwszej cyfry w argumencie X jest cyfra >= 5, to po funkcji CSNG(X) na pozycji szóstej będzie cyfra większa o 1 od cyfry w argumencie na tej samej pozycji. Jeżeli cyfra na siódmej pozycji jest mniejsza od 5, to po funkcji CSNG(X) wynik będzie równy pierwszym sześciu cyfrom argumentu o podwójnej precyzji.

EXP(X) — funkcja wykładnicza (e^X). Podnosi podstawę logarytmu naturalnego e ≈ 2.71828 do potęgi X. Jest funkcją odwrotną do LOG(X).

INT(X) — oblicza wartość całkowitą argumentu X. Obliczona wartość INT(X) jest równa lub mniejsza od argumentu X. Stosowanie funkcji nie ogranicza argumentów do przedziału −32768 ÷ 32767.

Przykłady:

INT(3.3) = 3 INT(−3.3) = −4

FIX(X) — wyznacza część całkowitą argumentu X:

$$FIX(X) = \begin{cases} INT(X) \text{ dla } X >= 0 \\ SGN(X)*INT(ABS(X)) \text{ dla } X < 0 \end{cases}$$

Przykłady:

FIX(3.3) = 3 FIX(−3.3) = −3

LOG(X) — oblicza wartość logarytmu naturalnego z X.

Przykłady:

1) LOG(5) = 1.60944
   LOG(10) = 2.30259

2) 5 A=0
   10 FOR I=1 TO 5
   15 A=A+LOG(I)
   20 NEXT I
   25 PRINT EXP(A)
   RUN

RND(X) — funkcja generatora liczb pseudolosowych. Argument X określa sposób inicjalizacji generatora oraz wyznaczania liczby RND(X).

RND(0) oblicza wartość z przedziału (0,1) w pojedynczej precyzji. Dla X należącego do zbioru liczb całkowitych z zakresu 1 ÷ 32767, funkcja RND(X) oblicza wartość z powyższego zakresu.

Przykłady:

1) 10 FOR I=1 TO 5 : PRINT RND(0) : NEXT I
   RUN
   .262145
   .06800311
   .930171
   .368143
   .140917

2) 10 FOR I = 1 TO 5 : PRINT RND(30000) : NEXT I
RUN
24058
27009
29076
15690
805

SGN(X) — funkcja przybiera trzy wartości:

$$SGN(X) = \begin{cases} -1 & \text{dla } X < 0 \\ 0 & \text{dla } X = 0 \\ 1 & \text{dla } X > 0 \end{cases}$$

SIN(X) — oblicza sinus argumentu X. Argument X musi być podany w radianach.

SQR(X) — oblicza pierwiastek kwadratowy z argumentu X.

TAN(X) — oblicza tangens argumentu X. Argument musi być podany w radianach.

RANDOM — ma charakter instrukcji i służy do uruchomienia generatora liczb pseudolosowych. Umieszczenie RANDOM na początku programu, zapewni generowanie powtarzalnej sekwencji liczb pseudolosowych za pomocą funkcji RND(X) każdorazowo po uruchomieniu programu.

a [X — podnoszenie liczby "a" do potęgi X. Funkcja umożliwia też obliczanie pierwiastka dowolnego stopnia.

Przykłady:

1) 4.4 [4.4 = 677.941
4.4 [(1 4.4) = 1.40036
1.5 [27 = 56815.4
10000 [.01 = 1.09648

2) 10 FOR I = 1 TO 5 : PRINT 2 [I : NEXT I
RUN
2
4
8
16
32

3.4. Instrukcje języka Basic

3.4.1. Instrukcje wyprowadzania znaków

PRINT *lista-elementów-do-wydruku*

Do wyprowadzania wyników obliczeń i prowadzenia przez użytkownika konwersacji z programem służy instrukcja wyświetlania znaków na ekranie monitora. Instrukcja ta składa się ze słowa PRINT, po którym są podane elementy do wydruku oddzielone separatorami. Basic dopuszcza znak zapytania "?" zamiast słowa PRINT. Elementami do wyprowadzenia mogą być:

— zmienne
— wyrażenia arytmetyczne
— łańcuchy
— tabulatory

Jako separatory są używane przecinek i średnik.
Przecinek umieszczony między dwoma elementami przeznaczonymi do wyprowadzenia, powoduje ich wyświetlenie w kolejnych sektorach szesnastopolowych tej samej linii ekranu. Jeżeli jako separator użyty jest średnik, to elementy są wyprowadzane obok siebie, ale z zachowaniem następujących reguł:

— każda liczba dodatnia jest poprzedzona spacją (czystym polem) a ujemna znakiem minus;
— po każdej liczbie jest drukowana jedna spacja;
— wartości są drukowane od pierwszej wolnej pozycji (dosuwanie w lewo);
— przed łańcuchami i po łańcuchach brak znaku spacji.

Powyższe zasady dotyczą też sytuacji, kiedy elementy przeznaczone do wyprowadzenia znajdują się w kolejnych liniach programu i w każdej linii po ostatnim elemencie występuje przecinek lub średnik. Jeżeli po ostatnim elemencie w danej linii nie ma separatora, to następny wydruk odbędzie się w nowej linii. Jeżeli instrukcja wyprowadzania składa się tylko ze słowa PRINT i nie posiada *listy-elementów*, to po wykonaniu jej nastąpi zmiana linii i kursor zajmie pozycję na początku nowej linii.
Liczby (wartości wyrażeń arytmetycznych) są wyprowadzane następująco:

— liczby całkowite mające od 1 do 6 cyfr wyprowadzane są jako liczby całkowite;
— liczby niecałkowite są przedstawiane jako ułamek dziesiętny o sześciu cyfrach (w pojedynczej precyzji), przy czym zera kończące liczbę nie są wyprowadzone;
— liczby, które nie dadzą przedstawić się w jednej z poprzednich postaci, są wyprowadzane w postaci z wykładnikiem, tj. jedna cyfra przed kropką dziesiętną i pięć po kropce, litera E, wykładnik potęgi poprzedzony znakiem.

Przykłady:

1)
```
10 A=1 : B=-3 : A$="ABC" : B$="DEF"
20 PRINT A,B;A$
30 PRINT A;B;A$
40 PRINT A$;B$
RUN
1             -3 ABC
1 -3 ABC
ABCDEF
```

2)

Liczba	Postać wydruku
3.0	3
7.3120	7.312
-17.0	-17
0.33	.33
1234567	1.23457E+06
0.00011	1.1E-04

PRINT @ *pozycja*, *lista-elementów*

Dopuszczalne są następujące formy zapisu instrukcji:

```
PRINT@100, A
PRINT @100 ,A
PRINT@ 100 ,A
PRINT @ 100,A
```

gdzie parametr *pozycja* zawiera się w przedziale: $0 \leqslant pozycja < 768$.
Instrukcja powoduje wyprowadzenie na ekran *listy-elementów* począwszy od określonej pozycji ekranu.

Przy organizacji ekranu 32 * 24, instrukcja PRINT@64, "ABC" powoduje wyświetlenie liter ABC począwszy od pierwszego pola w trzeciej linii ekranu.

PRINT TAB (*wyrażenie-arytmetyczne*)

Instrukcja powoduje przesunięcie kursora na pozycję określoną *wyrażeniem arytmetycznym*. Po napotkaniu powyższej instrukcji działanie jest następujące:

1. Obliczana jest wartość wyrażenia arytmetycznego.
2. Określana jest część całkowita.
3. Po przekroczeniu liczby 255 jest sygnalizowany błąd FC.
4. Jeżeli wartość wyrażenia arytmetycznego jest równa A, to

   dla $A \leqslant 31$ TAB(A)

   dla $31 < A \leqslant 255$ TAB(A − INT(A/32) * 32)

Przykład:

```
10 FOR I= −8 TO 8 : PRINT TAB(I * I/2.2); "*" : NEXT
```

Program wykreśli parabolę za pomocą znaku gwiazdki.

PRINT USING *łańcuch*; *lista-elementów*.

Bardzo często rozmieszczenie wyprowadzanych wyników możliwe do uzyskania za pomocą separatorów jest niewystarczające. Niejednokrotnie podczas wyprowadzania tablic wyników niektóre parametry chcemy uzyskać z dokładnością np. do dwóch miejsc po kropce dziesiętnej i dwóch miejsc przed kropką lub np. trzech cyfr plus wykładnik potęgi. Takie możliwości daje nam instrukcja PRINT USING, gdzie *łańcuch* określa wzorzec wydruku. Wzorzec ten musi być zadeklarowany w linii programu o numerze mniejszym niż numer linii programu z instrukcją PRINT USING. Do tworzenia *łańcucha* wzorca używamy następujących znaków:

\# + − . , $$** **$! [%n-spacji%

\# — Określa pozycję cyfry danej zmiennej numerycznej. Ilość użytych znaków określa liczbę cyfr wydruku. Jeżeli liczba znaków na lewo od kropki dziesiętnej będzie większa niż liczba cyfr, to na każdej wolnej pozycji będzie wydrukowana spacja. Jeżeli liczba znaków na prawo od kropki będzie większa niż

liczba cyfr, to na każdej wolnej pozycji będzie wydrukowane zero.

\+ — Postawiony znak "+" na początku pola lub na końcu wzorca wymusza wydruk znaku liczby przed pierwszą cyfrą znaczącą lub za ostatnią.

\− — Znak "−" umieszczony na końcu wzorca powoduje wydrukowanie spacji dla liczby dodatniej i znaku "−" dla ujemnej.

. — Kropka dziesiętna może być umieszczona w dowolnym miejscu wzorca określonego znakami #.

, — Umieszczenie przecinka w dowolnym miejscu z lewej strony pomiędzy cyframi i kropką dziesiętną powoduje oddzielenie każdej trójki cyfr na lewo od przecinka. Przecinek stanowi dodatkową pozycję w polu numerycznym.

$$ — Dwa znaki dolara umieszczone na początku wzorca powodują wyświetlenie znaku dolara przed pierwszą cyfrą wydruku.

** — Dwie gwiazdki umieszczone przed wzorcem powodują w przypadku niewykorzystania pozycji na lewo od kropki dziesiętnej zaznaczenie ich gwiazdkami. Dwie gwiazdki we wzorcu stanowią dodatkowe dwie pozycje w wydruku.

**$ — Ta sekwencja, poprzedzająca wzorzec, powoduje w przypadku niewykorzystanych miejsc na lewo od kropki, zaznaczenie ich gwiazdkami i postawienie przed pierwszą cyfrą znaku dolara.

! — Powoduje wyprowadzenie pierwszego znaku z łańcucha występującego na *liście-elementów*.

[— Łącznie ze znakiem # określa format wydruku liczby. Ciąg czterech znaków [oznacza wyprowadzenie litery E, znaku + lub − i dwóch cyfr wykładnika.

%n-spacji % — wyprowadzenie łańcucha znakowego o długości określonej przez ilość spacji plus 2.

Przykłady użycia instrukcji PRINT USING A$:

Wzorzec A$	Wartość zmiennej	Postać wydruku
1) ##.##	1.222	1.22
	12.1	12.10
+##.##	1.11	+1.11
	−2.22	−2.22
##.#+	11.11	11.1+
	−11.11	11.1−
##.#−	11.11	11.1
	−11.11	11.1−
##,####.#	11223.4	11.223.4
$$####.##	1.234	$1.23
**##.#	1.234	***1.2
	345.22	*345.2
**$####.#	1.23	****$1.2
!	"ABCD"	A
##.##[[[[	100	1.00E+02
%%%	"ABCDE"	AB
%␣␣%	"ABCDE"	ABCD

```
2) 5 Z=1985
   10 A$="#####.##    ##.#####[[[[    .#####[[[["
   15 PRINT USING A$;Z;Z;Z
   RUN
   1985.00      1.9850E+03      .0199E+05
3) 5 B=19985.5 : A=1322.1
   10 A$="ZAPLACONO ZL #####.## ZOSTALO
      ######.## ZL"
   15 PRINT USING A$;A;B-A
   RUN
   ZAPLACONO ZL 1322.10 ZOSTALO 18663.40 ZL
```

PRINT # −1, *lista-elementów*.

Instrukcja służy do zapisania zmiennych numerycznych i łańcuchowych na taśmie magnetofonowej. Informacje umieszczone na *liście-elementów*

za pojedynczą instrukcją PRINT # −1 nie mogą zajmować razem więcej niż 255 bajtów. Wszystkie bajty powyżej 255 nie zostaną zapisane na taśmie.

Przykład:

```
10 A=5 : B=17:A$="COBRA"
20 PRINT # -1,A,B,A$
```

LPRINT

Instrukcja służy do wyprowadzania danych na drukarkę. Sposób użycia jest taki, jak dla instrukcji PRINT. Dotyczy to również instrukcji LPRINT TAB(*n*) i LPRINT USING A$;A;B;...

3.4.2. Instrukcje wprowadzania danych

INPUT *lista-elementów.*

Instrukcja służy do wprowadzania danych numerycznych lub łańcuchowych zgodnych z *listą-elementów* umieszczoną za słowem INPUT. Po napotkaniu tej instrukcji program zostaje zatrzymany do momentu wprowadzenia z klawiatury wartości, które zostaną przyporządkowane kolejnym zmiennym. Przy wykonywaniu tej instrukcji zostanie wyświetlony w nowej linii znak zapytania i spacja:
? _ (kreska pozioma określa położenie kursora)

Użytkownik powinien wtedy wprowadzić odpowiednią liczbę danych oddzielonych przecinkami i zakończyć znakiem CR. Jeżeli po każdej danej naciśniemy klawisz CR, to w nowej linii będą wyświetlone dwa znaki zapytania i spacja:

```
?? _
```

Jeżeli będziemy wprowadzać dane oddzielając je przecinkami, to może się zdarzyć, że podamy ich więcej niż żąda instrukcja INPUT. W tej sytuacji zostanie wyświetlony na ekranie komunikat:

```
? EXTRA IGNORED
```

i program będzie się wykonywał dalej.
Jeżeli po każdej danej będziemy naciskać klawisz CR, to program po

wprowadzeniu ostatniej danej z listy przejdzie sam do wykonywania instrukcji z następnej linii programu Basic.
Przy próbie wprowadzenia danej niezgodnej z typem podanym na liście za słowem INPUT, zostanie wyświetlony komunikat:

? REDO

Dotyczy to takiej sytuacji, w której program żąda danej numerycznej, a zostanie wprowadzona dana łańcuchowa lub wyrażenie. Po wyświetleniu komunikatu "? REDO" należy podać żądane dane od początku według listy za słowem INPUT.
W instrukcji INPUT można umieścić komunikat przed listą ujęty w cudzysłów i zakończony średnikiem:

10 INPUT "PODAJ DANA X"; X

Komunikat ujęty w cudzysłowie zostanie wyświetlony przed znakiem zapytania.

INPUT # −1, *lista-elementów*.

Instrukcja służy do wczytania zmiennych zapisanych na kasetę magnetofonową za pomocą instrukcji PRINT # −1, *lista-elementów*. Po wczytaniu zmiennych przyporządkowuje je zmiennym umieszczonym na *liście-elementów*. Dane umieszczone na *liście-elementów* muszą być zgodne ze strukturą danych umieszczonych na taśmie. Zgodność ta musi dotyczyć ich liczby, typu i kolejności wystąpienia zmiennych. W przypadku niezgodności są sygnalizowane następujące błędy:

— jeśli wystąpi zły typ zmiennych, jest sygnalizowany błąd złego pliku danych (BAD FILE DATA) o kodzie FD;
— przy małej liczbie danych na taśmie w stosunku do liczby elementów zostanie zasygnalizowany błąd braku danych (OUT OF DATA) o kodzie OD.

DATA

Podczas wprowadzania linii programu istnieje możliwość tworzenia zestawu danych, które będą następnie używane przez program. Do tworzenia zestawu danych służy instrukcja składająca się ze słowa

DATA, po którym następują dane oddzielone przecinkami. Jako dane mogą występować stałe numeryczne lub łańcuchy. Jeżeli łańcuchy zawierają tabulatory lub spacje, należy je umieścić w cudzysłowie.

Przykłady:

```
10 DATA 1,7,17,33
20 DATA COBRA, COBRESPU, ESPU
30 DATA " COBRA ","",ESPU"
```

Do pobierania danych z zestawu DATA i umieszczania ich pod zmienne programu służy instrukcja

READ *lista-elementów*

Pobieranie danych zaczyna się od pierwszego elementu z zestawu DATA i zostaje on przyporządkowany pierwszej zmiennej. Jeżeli program zażąda więcej danych niż zawiera zestaw DATA, to zostanie wyświetlony błąd (OUT OF DATA) o kodzie OD.

Przykłady:

1)
```
10 DIM A(5)
20 FOR I=1 TO 5 : READ A(I) : NEXT I
30 DATA 11,13,17,29,31
```

Po wykonaniu tego programu kolejnym elementom macierzy A(I) zostaną przyporządkowane liczby z zestawu DATA, np. A(3)=17.

2)
```
10 CLS
20 DATA 0,0000,1,0001,2,0010,3,0011,4,0100,5,0101,6,0110,7,0111,
        8,1000,9,1001,A,1010,B,1011,C,1100,D,1101,E,1110
        F,1111
25 RESTORE
30 INPUT" PODAJ ZNAK HEX " ; A$
32 FOR I=1 TO 16 : READ C$,B$
36 IF C$=A$ THEN 40 ELSE NEXT
38 PRINT " TO NIE JEST ZNAK HEX." : GOTO 25
40 PRINT " ZNAK ";A$;" MA WARTOSC BINARNA ";B$ :
   GOTO 25
```

3) 10 CLS
20 DATA 0,0000,1,0001,2,0010,3,0011,4,0100,5,0101,6,0110,7,
0111,8,1000,9,1001,A,1010,B,1011,C,1100,D,1101,E,
1110, F,1111
30 INPUT "PODAJ DWA ZNAKI HEX. ";A$
40 Z1$ = LEFT$(A$,1) : GOSUB 200
50 PRINT A$;" = ";Z2$;
60 Z1$ = RIGHT$(A$,1) : GOSUB 200
70 PRINT Z2$: GOTO 30
200 RESTORE : FOR I = 1 TO 16 : READ B$,Z2$
220 IF Z1$ = B$ THEN RETURN ELSE NEXT
230 PRINT " TO NIE JEST ZNAK HEX. " : GOTO 30

RESTORE

Instrukcja umożliwia wielokrotne korzystanie z tych samych danych zawartych w DATA.

3.4.3. Instrukcja komentarza

REM

Instrukcja ta służy do umieszczania komentarzy dotyczących działania programu. Składa się ona ze słowa REM oraz ciągu znaków:

10 REM PROGRAM MINIMALIZACJI KOSZTOW
20 REM PRODUKCJI TELEWIZOROW

Basic po identyfikacji instrukcji REM ignoruje wszystko to, co jest umieszczone za słowem REM do końca linii programu i przechodzi do realizacji następnej linii. Basic dopuszcza używanie znaku apostrofu "'" zamiast słowa REM.

10' PROGRAM MINI.....
20' PROD.....

3.4.4. Instrukcja podstawienia

LET

Instrukcja podstawienia pozwala zmienić wartości poszczególnych zmiennych występujących w programie. Umożliwia to przypisanie zmiennym wyników obliczeń z możliwością ich dalszego wykorzystania.

Oprócz zmiennej numerycznej możemy przypisać zmiennej całe wyrażenie arytmetyczne.

```
10 LET A=5
20 LET B=I+I[2+33
30 LET C=2*SIN(I/3)+SIN(I)/I
```

Wyrażenie po prawej stronie może zawierać zmienne, stałe, operatory arytmetyczne, funkcje matematyczne. W opisywanej wersji języka Basic używanie słowa LET nie jest konieczne i podstawienie można zapisać następująco:

```
10 A=5
20 B=I+I[2+33
30 C=2*SIN(I/3)+SIN(I)/I
```

3.4.5. Instrukcja zerowania zmiennych

CLEAR X

Instrukcja powoduje wyzerowanie wszystkich zmiennych, które wystąpiły wcześniej w programie oraz powoduje przydzielenie tylu bajtów pamięci do przechowywania łańcuchów, ile wynosi stała "X" podana bezpośrednio po słowie CLEAR, np. CLEAR 100. Basic po wpisaniu i zainicjowaniu rezerwuje automatycznie 50 bajtów pamięci na zmienne łańcuchowe. Jeżeli ilość znaków wczytanych łańcuchów przekroczy liczbę zarezerwowanych bajtów, to zostanie wyświetlony komunikat (OUT OF STRING SPACE) o kodzie OS.

3.4.6. Bezwarunkowe instrukcje zatrzymujące program

STOP

Instrukcja powoduje zatrzymanie programu w określonej przez programistę linii. Zatrzymany program wyświetla komunikat BREAK IN (numer linii). Po zatrzymaniu programu możemy sprawdzić np. zmienne używając instrukcji PRINT. Jeżeli nie wznawiamy programu instrukcją CONT, możemy przykładowo modyfikować program lub zmienne.

```
10 A$=INKEY$ : IF A$ < > "X" THEN 10
20 STOP
 RUN
```

Naciśnięcie dowolnego klawisza spowoduje wygenerowanie sygnału dźwiękowego. Po naciśnięciu litery "X" pojawi się komunikat BREAK IN 20.

END

Instrukcja ta bywa umieszczana w miejscu logicznego końca programu. Przy sprawdzaniu i uruchamianiu programu, linie o numerach wyższych od linii, w której znajduje się instrukcja END, są przez Basic ignorowane.

ERROR *numer*

Instrukcja powoduje taką reakcję języka Basic, jak na rzeczywisty błąd. Reakcją tą jest przerwanie programu i wyświetlenie komunikatu. Zestaw błędów — patrz rozdział 8.

Przykład:

```
10 ERROR 6
 ? OV ERROR IN 10
```

3.4.7. Instrukcje definiujące typy zmiennych

DEFINT
DEFSNG
DEFDBL
DEFSTR

lista-liter-lub-zakresów-literowych

Wszystkie zmienne, których nazwa rozpoczyna się od dowolnej litery z listy, są interpretowane jako:

— całkowite dla DEFINT (zakres $-32768 \div 32767$);
— pojedynczej precyzji dla DEFSNG z dokładnością do 6 cyfr;
— podwójnej precyzji dla DEFDBL z dokładnością do 16 cyfr;
— łańcuchowe dla DEFSTR do 255 znaków przy uprzedniej właściwej deklaracji CLEAR X.

Znaki deklaracji typu zmiennych (#, %, !) mają wyższy priorytet od deklaracji typu podanych za pomocą powyższych instrukcji.
Wymienione instrukcje definiujące umożliwiają optymalizację programu i zwiększenie szybkości. Po zdefiniowaniu np. macierzy składającej się z liczb całkowitych, pamięć mikrokomputera będzie w stanie zmieścić większą ilość jej elementów niż np. przy liczbach podwójnej precyzji (różnica 4-krotna). Różnica w czasie trwania pętli FOR — NEXT jest prawie 2-krotna, jeżeli zmienna pętli jest liczbą zdefiniowaną przez DEFINT w porównaniu do zmiennej pętli w pojedynczej precyzji.

```
10 DEFINT A-D : DEFDBL L-O : DEFSTR X-Z
```

Brak deklaracji pozostałych liter oznacza, że zmienne, których będą nazwami, są pojedynczej precyzji. Wszystkie zmienne, których nazwa zaczyna się od liter A–D będą występowały w programie jako całkowite, od liter L–O jako podwójnej precyzji, a od liter X–Z jako łańcuchy.

3.4.8. Macierze

DIM *nazwa* (*i,j k,...*)

Instrukcja powoduje zadeklarowanie macierzy o zadanej liczbie elementów w każdym wymiarze. Argumenty określające wymiar mogą być stałą lub wyrażeniem. Jeżeli w programie zostanie użyta instrukcja, która będzie wymagała zastosowania tablicy a nie nastąpiła wcześniejsza deklaracja jej wymiarów, to Basic automatycznie przyjmie każdy wymiar równy 11(0 ÷ 10).
Instrukcja *deklaracji wymiarów* służy programowi do rezerwacji miejsca w pamięci na przechowywanie elementów indeksowanych. Próba deklaracji nowej tablicy o innych wymiarach niż poprzednia lecz o tej samej nazwie jest traktowana jako błąd i oznaczona kodem DD. Instrukcja deklaracji DIM umożliwia nam deklarację tablic ze zmiennymi typu INT, DBL, SNG oraz łańcuchowymi:

```
10 DIM A%(3,4), B(3,5,5), C#(7), D$(2,2,3)
```

3.4.9. Instrukcje skoku

GOTO *numer-linii*

Instrukcja skoku bezwarunkowego GOTO pozwala zmienić kolejność wykonywania programu i przekazuje sterowanie do instrukcji umieszczonej w linii o wymienionym numerze linii. W przypadku skoku programu do nieistniejącej linii, Basic sygnalizuje błąd (UNDEFINED LINE) o kodzie UL oraz podaje numer linii, w której ten błąd wystąpił.

ON *wyrażenie-arytmetyczne* **GOTO** $n_1, n_2, ..., n_k$

Instrukcja ta jest rozszerzeniem instrukcji GOTO i służy do przekazywania sterowania programu do różnych miejsc w zależności od wartości *wyrażenia-arytmetycznego*. Po napotkaniu tej instrukcji program oblicza wartość *wyrażenia-arytmetycznego* i określa jego część całkowitą. Jeżeli wartość INT (*wyrażenie-arytmetyczne*) jest mniejsze od zera, to jest sygnalizowany błąd (ILLEGAL FUNCTION CALL) o kodzie FC. Jeżeli część całkowita jest równa zero lub przewyższa liczbę pozycji listy, to sterowanie programu przechodzi do następnej linii.

Przykład:

```
10 ON A+1 GOTO 30,40,50,60
20 PRINT A$ ; B$
```

Po obliczeniu wyrażenia arytmetycznego A + 1 program przejdzie do realizacji linii:

30 jeżeli część całkowita A + 1 = 1
40 jeżeli część całkowita A + 1 = 2
50 jeżeli część całkowita A + 1 = 3
60 jeżeli część całkowita A + 1 = 4

Dla wartości INT(A + 1) = 0 oraz INT(A + 1) > 4 program przejdzie do wykonywania linii 20.

3.4.10. Instrukcje obsługi podprogramów

GOSUB *numer-linii*

Instrukcja ta powoduje przekazanie sterowania do linii programu podanej za słowem GOSUB. Numer tej linii jest numerem pierwszej instrukcji podprogramu. Instrukcji tej używamy w programie wtedy, gdy

trzeba kilkakrotnie wykonać tę samą część programu. Po wykonaniu ciągu instrukcji należących do podprogramu, należy wykonać *instrukcję powrotu*. Podczas wykonywania instrukcji GOSUB zostaje zapamiętany numer linii następnej instrukcji. Dozwolone jest wykonanie wielu instrukcji GOSUB zanim zostanie wykonana instrukcja RETURN.

RETURN

Instrukcja ta kończy realizację podprogramu, a jej wykonanie powoduje przejście do linii o numerze, który został zapamiętany przy ostatnim wykonaniu instrukcji GOSUB. Jeśli program napotka instrukcję RETURN bez uprzedniego wykonania GOSUB, to zostanie zasygnalizowany błąd o kodzie RG. Program może zawierać wiele instrukcji RETURN, lecz liczba wykonanych nie może być większa od liczby wykonanych wcześniej instrukcji GOSUB.

Przykład:

```
10 A=7 : B=3 : C=11
20 GOSUB 200
30 A=5 : B=2 : C=16
40 GOSUB 200
50 END
200 FOR I=0 TO A
210 C(I)=2[ B+C*I+1
220 NEXT I : Z=0
230 FOR I=0 TO A
240 Z=Z+C(I)
250 NEXT I : PRINT Z : RETURN
 RUN
 380
 270
```

ON *wyrażenie-arytmetyczne* **GOSUB** n_1, n_2, ..., n_k

Instrukcja przekazuje sterowanie do podprogramu rozpoczynającego się od numeru linii podanego za słowem GOSUB. Zasady obliczania numeru linii, do której nastąpi skok są takie same, jak w instrukcji ON...GOTO. Po wykonaniu skoku i podprogramu obowiązuje instrukcja RETURN. Zasady są takie same jak dla instrukcji GOSUB — RETURN.

ON ERROR GOTO *numer-linii*

Instrukcja powoduje skok do określonej linii programu (podprogramu) po wystąpieniu błędu w programie głównym. Powyższej instrukcji należy użyć wcześniej przed miejscem wystąpienia błędu. Po wykonaniu tej instrukcji nie będzie sygnalizowany komunikat o błędzie i nie nastąpi przerwanie programu. Podprogram obsługi błędu powinien być zakończony instrukcją RESUME. Działanie instrukcji ON ERROR GOTO... może być przerwane poprzez powtórne jej użycie z numerem linii 0.

ON ERROR GOTO 0

Przywraca to normalną reakcję języka Basic na napotykane błędy.

RESUME *numer-linii*

Instrukcja ta kończy podprogram obsługi błędu przez określenie miejsca, od którego ma być wznowione dalsze wykonywanie programu. Jeżeli po instrukcji RESUME nie zostanie podany numer linii lub podamy RESUME 0, to program będzie kontynuowany od instrukcji, w której wystąpił błąd. Przy połączeniu RESUME NEXT program będzie kontynuowany od następnej instrukcji, po której wystąpił błąd. Jeżeli zostanie podany numer linii, np. RESUME 200, to program będzie dalej wykonywany od linii 200.

Przykład:

```
10 ON ERROR GOTO 200
20 PRINT "PROGRAM LICZY N SILNIA"
30 INPUT "PODAJ N" ; N
40 IF N=0 THEN N=1 ELSE IF N < 0 THEN 30
50 B=0 : A=1 : FOR I=1 TO N
60 A=A*I
70 NEXT I
80 IF B=0 THEN PRINT A : GOTO 30
100 A$=RIGHT$(STR$(A),2)
110 B=B+VAL(A$)
120 PRINT LEFT$(STR$(A),LEN(STR$(A))-2)+STR$(B)
130 GOTO 30
200 A=A/1E30 : B=B+30
210 RESUME
```

Jeżeli zmienna A obliczona w linii 60 przekroczy największą liczbę dopuszczalną w operacjach arytmetycznych (ok. 1.7E+38), to wystąpi błąd przepełnienia (OVERFLOW) i program będzie wykonywany od linii 200.

3.4.11. Instrukcje pętli

FOR *zmienna = wyrażenie-1* **TO** *wyrażenie-2* **STEP** *krok*

Instrukcja służy do cyklicznego wykonywania bloku programu określoną liczbę razy. Blok *pętli* składa się z linii zawierającej instrukcję otwarcia pętli: FOR...TO...STEP, ciągu linii, które będą wykonywane cyklicznie oraz linii zawierającej instrukcję zamknięcia pętli: NEXT. *Wyrażenie-1* po słowie FOR podaje początkową wartość zmiennej sterującej, *wyrażenie-2* podaje końcową wartość zmiennej sterującej, a parametr *krok* określa wartość o jaką jest zmieniana zmienna sterująca w każdym kolejnym cyklu.

Przykłady:

1)
```
10 FOR A = -5 TO 5 STEP .5
   15 PRINT A,
   20 NEXT A
```

Jeżeli zapis instrukcji FOR... będzie następujący:

```
10 FOR A = -5 TO 5
15 PRINT A
20 NEXT A
```

i nie zostanie podany parametr *krok*, to Basic automatycznie przyjmie STEP 1.

2)
```
10 FOR X=1 TO 9
   20 FOR Y=0 TO 9
   30 FOR Z=0 TO 9
   40 IF X*100+Y*10+Z=X*X*X+Y*Y*Y+Z*Z*Z THEN 100
   50 NEXT Z
   60 NEXT Y
   70 NEXT X
   80 END
   100 PRINT X;Y;Z
   110 GOTO 50
```

Program znajduje takie cyfry, których suma trzecich potęg równa jest liczbie złożonej z tych cyfr.

```
RUN
1  5  3
3  7  0
3  7  1
4  0  7
```

3.4.12. Instrukcje warunkowe

IF *wyrażenie-logiczne* **THEN** *program-1* **ELSE** *program-2*

Instrukcja warunkowa IF służy do sterowania wykonywanym programem w zależności od wartości wyrażenia logicznego (patrz — "Operatory logiczne"). Jeśli wartością wyrażenia jest *prawda*, to program jest wykonywany dalej od pierwszej instrukcji po słowie THEN (*program-1*). Natomiast dla wartości *fałsz* sterowanie jest przekazywane do pierwszej instrukcji po słowie ELSE (*program-2*). Jako *program-1* lub *program-2* może być zadeklarowana instrukcja, kilka instrukcji lub numer linii, od której należy kontynuować program. Bardzo często korzysta się z instrukcji warunkowej z pominięciem części "ELSE *program-2*". I tak: jeśli wynikiem jest *prawda*, to program tak jak poprzednio jest wykonywany od instrukcji po słowie THEN (*program-1*), a w przypadku *fałsz* jest wykonywana instrukcja z następnej linii programu.

Przykłady:

1)
```
10 INPUT A,B
20 IF A<B THEN A=A/2 ELSE A=B/3
30 END
```

2)
```
10 INPUT A,B
20 IF A<B THEN 40 ELSE 60
40 A=A/2
50 END
60 B=B/3
70 END
```

3)
```
10 IF INKEY$="" THEN 10
20 PRINT "ZOSTAL NACISNIETY KLAWISZ" ; INKEY$
```

3.4.13. Instrukcje obsługi błędów

ERL

Instrukcja ta powoduje przypisanie zmiennej ERL numeru linii programu, w której wystąpił błąd. Jeżeli błąd wystąpił w trybie bezpośrednim, to ERL = 65535. Instrukcja jest używana do obsługi błędów wewnątrz programu.

Przykład:

```
10 FOR I= -2 TO 2
20 PRINT 1/ I
30 NEXT I
 RUN
 -.5
 -1
?/0 ERROR IN 20
PRINT ERL
  20
PRINT 1/0
?/0 ERROR
PRINT ERL
65535
```

ERR/2+1

Instrukcja ta powoduje przypisanie zmiennej ERR/2+1 numeru błędu (patrz rozdz. 8 — tablica błędów). Zmienna ta nie może być użyta wewnątrz programu obsługi błędu.

Przykład:

```
PRINT 1/0
?/0 ERROR
PRINT ERR/2+1
11          (błąd dzielenia przez zero)
```

3.4.14. Instrukcje sterujące wizją

CLS

Instrukcja służy do kasowania ekranu. Zrealizowana jest za pomocą dwóch operacji:

— ustaw kursor na pozycję "0" (HOME), (kod 28);
— skasuj od kursora do końca ekranu (kod 31).
Odpowiada temu wykonanie poniższej instrukcji:

```
PRINT CHR$(28); CHR$(31);
```

POS(X)

Instrukcja powoduje przypisanie zmiennej POS(X) liczby z zakresu 0 ÷ 31 i wskazuje bieżącą pozycję kursora w zapisywanej linii ekranu. Jako argument X można przyjąć dowolną wartość, o ile nie przekroczy ona największej liczby zmiennoprzecinkowej, tj. około 1.701E+38.

Przykłady:

1)
```
10 PRINT TAB(RND(28)); POS(0)
20 GOTO 10
```

2)
```
10 INPUT A : IF A <0 OR A > 28 THEN 10
20 PRINT TAB(A); POS(0)
30 GOTO 10
```

SET(X, Y)

Instrukcja powoduje zapalenie punktu semigraficznego w miejscu określonym przez współrzędne X i Y. Współrzędna X może zawierać się w przedziale 0 ÷ 63, a Y w przedziale 0 ÷ 71. Parametrami X i Y w funkcji SET(X,Y) mogą być dowolne wyrażenia arytmetyczne. W przypadku, gdy wartość jednego z parametrów przekroczy maksymalną dopuszczalną wartość, zostanie zasygnalizowany błąd (ILLEGAL FUNCTION CALL) o kodzie FC. Rozdzielczość semigraficzna ogranicza ilość pól na ekranie, gdyż każde pole podzielone jest na sześć części. Każdej kombinacji punktów w ramach jednego pola odpowiada inny znak w nowym generatorze znaków.

RESET(X,Y)

Instrukcja powoduje zgaszenie punktu semigraficznego. Wymagania takie, jak w instrukcji SET(X,Y).

Przykład:

```
5 CLS
10 A=3.1415/70
20 FOR I=1 TO 140
30 X=32+20*COS(A*(I+30))          Y=38+20*SIN(A*(I+30))
   SET(X,Y)
40 X=32+20*COS(A*I) : Y=38+20*SIN(A*I) : RESET(X,Y)
50 NEXT I
60 GOTO 20
```

Program pokazuje działanie instrukcji SET(X,Y) i RESET(X,Y).

POINT(X,Y)

Instrukcja służy do testowania punktu o podanych współrzędnych. Jeżeli dany punkt jest zapalony, wtedy POINT dla PRINT przyjmuje wartość: −1, a dla funkcji logicznej:1. Jeżeli punkt jest zgaszony, to wartość POINT równa się 0.

Przykład:

```
 5 CLS
10 FOR I=1 TO 30
20 SET(I,I)
30 NEXT I
40 PRINT POINT(20,20)
50 IF POINT(20,20) THEN PRINT "PUNKT ZAPALONY"
 RUN
 −1
PUNKT ZAPALONY
```

3.4.15. Instrukcje operacji na zawartości pamięci

PEEK(*adres*)

Instrukcja powoduje odczytanie zawartości komórki pamięci o adresie *adres* i funkcja PEEK (*adres*) przyjmuje jej wartość w postaci dziesiętnej.

```
A = PEEK(30000)
```

Instrukcja powoduje odczytanie zawartości pamięci spod adresu 30000 i

przyporządkowanie jej zmiennej A. Jeżeli adres należy do przedziału (0. 32767), to zapis jest taki jak pokazano wyżej. Jeżeli adres należy do przedziału (32768, 65535), to zapis wygląda następująco:

A = PEEK (adres — 65536)

POKE *adres, wyrażenie*

Instrukcja powoduje zapisanie bajtu o wartości określonej przez *wyrażenie* do komórki pamięci o adresie *adres*. Wartość *wyrażenia* zawiera się w przedziale (0, 255). Zasady zapisu parametru *adres* takie, jak dla instrukcji PEEK (*adres*).

3.4.16. Instrukcja wywołania procedury w języku maszynowym

USR(X)

Instrukcja umożliwia wywołanie procedury napisanej w języku maszynowym mikroprocesora Z-80 z programu realizowanego w języku Basic. Przed wywołaniem USR(X) należy najpierw umieścić w dwóch komórkach pamięci adres pierwszej instrukcji procedury napisanej w języku maszynowym. Należy go umieścić w komórkach o adresach 16526 (młodszy bajt) i 16527 (starszy bajt) za pomocą instrukcji POKE *adres, wyrażenie.*

Przykład:

Zakładamy, że napisana przez nas w języku maszynowym procedura jest umieszczona od adresu 3D00 (tj. 15616)

```
10 POKE 16526, 0 : ' (00)
20 POKE 16527, 61 : ' (3D)
30 PRINT USR (0) : ' lub A = USR (0)
```

Procedura napisana w języku maszynowym musi być zakończona instrukcją RET, jeżeli z procedury maszynowej nie będzie przekazywany parametr do języka Basic. Program w języku maszynowym należy umieścić za pomocą systemu MONITOR COBRA i uruchomić Basic. Jeżeli program został najpierw napisany w języku Basic, a potem chcemy wpisać do pamięci program maszynowy, to należy:

— wykonać reset mikroprocesora;
— po zgłoszeniu systemu MONITOR wpisać program maszynowy do pamięci mikrokomputera za pomocą procedury M;
— obliczyć adres (młodszy i starszy bajt) za pomocą procedury C;
— wykonać start języka Basic (adres 66H) za pomocą procedury G.

Po zgłoszeniu się języka Basic uzupełnić używając instrukcji POKE zawartość komórek pamięci o adresie 16526 i 16527 przeliczoną wartością młodszego i starszego bajtu adresu programu i uruchomić za pomocą instrukcji RUN. Każda pomyłka w programie napisanym w języku maszynowym może zmienić zawartość całej pamięci RAM razem z językiem Basic. Wszystkie programy w języku maszynowym używane przez Basic można umieszczać w obszarze pamięci od adresu 3100H(12544) do adresu 3FFFH(16383), ponieważ ten obszar pamięci w mikrokomputerze COBRA nie jest używany przez Basic-COBRA.
W przypadku, gdy program będzie dłuższy od 3800 bajtów, należy go umieścić przed końcem pamięci operacyjnej. Wymaga to jednocześnie ograniczenia obszaru pamięci przydzielanej językowi Basic i po pytaniu MEMORY SIZE? — podać wartość pomniejszoną o obszar zarezerwowany dla programu maszynowego.
Jeżeli chcemy przekazać daną z programu Basic do programu maszynowego, to umieszczamy ją jako "X". Po wejściu w procedurę maszynową parametr ten odzyskujemy za pomocą instrukcji CALL 0A7FH (CD 7F 0A). Jeżeli chcemy przekazać parametr z procedury maszynowej do programu Basic, to umieszczamy go w parze rejestrów HL mikroprocesora i wykonujemy instrukcję JP 0A9A (C3 9A 0A). Instrukcja ta powoduje umieszczenie parametru z pary HL w komórkach pamięci o adresach 4121H i 4122. Należy pamiętać, aby ilość użytych instrukcji PUSH i POP była taka sama w momencie powrotu do języka Basic.

3.4.17. Instrukcje obsługi urządzeń zewnętrznych

INP (*port*)

Instrukcja ta jest przeznaczona do czytania danej (bajtu) z urządzenia lub układu zewnętrznego (portu). Urządzeniem zewnętrznym może być dowolny układ elektroniczny, który umożliwi przekazanie danej na szynę danych mikroprocesora. Jako układ zewnętrzny może być zasto-

sowany PIO, CTC, SIO, itd. Argument *port* może przyjmować wartość z zakresu $0 \div 255$.

A = INP(148)

OUT *port, wyrażenie*

Instrukcja powoduje wysłanie bajtu danych o wartości *wyrażenie* do urządzenia (układu) zewnętrznego o adresie *port*. Argument *wyrażenie* oraz *port* może zawierać się w zakresie $0 \div 255$.

OUT25,0

3.4.18. Instrukcje dodatkowe

MEM

Instrukcja określa liczbę wolnych bajtów w pamięci dostępnej dla programu Basic.

Przykład:

Po zainicjowaniu języka Basic pojawi się komunikat

MEMORY SIZE? 30000 CR(wartość zadeklarowana)
PRINT MEM
12548

VARPTR (*nazwa-zmiennej*)

Instrukcja powoduje wyznaczenie wartości adresów komórek pamięci zajmowanych przez zmienne występujące w programie. Wyznaczona wartość adresu jest przypisana zmiennej VARPTR. W komórce pamięci o adresie wyznaczonym za pomocą tej instrukcji jest umieszczony najmłodszy bajt zmiennej numerycznej lub długość łańcucha dla zmiennej łańcuchowej.

AB=2
PRINT VARPTR(AB)
17134

Zmienna AB nie posiada znaku deklaracji, a więc jest zmienną pojedynczej precyzji — SNG(4 bajty).

Przykład:

Wykonamy krótki program w trybie bezpośrednim:

```
FOR I=17132 TO 17137 : PRINT PEEK (I) : NEXT CR
```

Na ekranie otrzymamy sześć wyświetlonych liczb:
65 (17132) — pierwsza litera nazwy zmiennej
66 (17133) — druga litera nazwy zmiennej
0 (17134) — najmłodszy bajt zmiennej AB
0 (17135) — starszy bajt zmiennej AB
0 (17136) — najstarszy bajt zmiennej AB
130 (17137) — wartość wykładnika z nadmiarem 128

Rozmieszczenie bajtów danych dla różnego typu zmiennych:

— dla liczb całkowitych (INT)
(VARPTR) — mniej znaczący bajt (zapis w uzupełnieniu do 2)
(VARPTR+1) — bardziej znaczący bajt (jak wyżej)
— dla liczb pojedynczej precyzji (SNG)
(VARPTR) — najmniej znaczący bajt mantysy
(VARPTR+1) — starszy bajt mantysy
(VARPTR+2) — najstarszy bajt mantysy
(VARPTR+3) — wykładnik potęgi z nadmiarem 128
— dla liczb podwójnej precyzji (DBL)
(VARPTR) — najmniej znaczący bajt mantysy
(VARPTR+1) — starszy bajt mantysy
(VARPTR+2) — starszy bajt mantysy
(VARPTR+3) — starszy bajt mantysy
(VARPTR+4) — starszy bajt mantysy
(VARPTR+5) — starszy bajt mantysy
(VARPTR+6) — najstarszy bajt mantysy
(VARPTR+7) — wykładnik potęgi z nadmiarem 128
— dla zmiennej łańcuchowej (STR)
(VARPTR) — długość łańcucha
(VARPTR+1) — młodszy bajt adresu początku łańcucha
(VARPTR+2) — starszy bajt adresu początku łańcucha

Dla liczb pojedynczej i podwójnej precyzji najstarszy bit w najstarszym

bajcie mantysy jest *bitem znaku* liczby. Dla zmiennej AB = −2 wydruk w poprzednim przykładzie byłby następujący:

```
65
66
0
0
128
130
```

3.4.19. Instrukcje operacji na łańcuchach

Łańcuch jest stałą lub zmienną alfanumeryczną i podczas deklaracji jest ujęty w cudzysłów, np. "COBRA". *Długość łańcucha* jest to liczba wszystkich znaków ujętych w cudzysłowie łącznie ze znakiem spacji. Maksymalna długość łańcucha wynosi 255. Jako zmienna łańcuchowa może być zadeklarowana każda nazwa przez instrukcję DEFSTR lub przez znak deklaracji ($), np.: A$, AB$.

Wprowadzanie zmiennych łańcuchowych może odbyć się przy użyciu instrukcji DATA, READ oraz INPUT i INPUT# −1. Zmienne łańcuchowe wprowadzamy za pomocą instrukcji PRINT, LPRINT i PRINT# −1. Przed wprowadzeniem zmiennych łańcuchowych należy zarezerwować odpowiednio duży obszar pamięci operacyjnej używając instrukcji CLEAR *n*. Zmienne łańcuchowe (alfanumeryczne) mogą być proste i indeksowane, przy czym zmienne wprowadzane za pomocą instrukcji INPUT lub READ muszą być stałymi alfanumerycznymi. Jeżeli we wprowadzanej stałej będzie użyty znak zastrzeżony, np. tabulator, to należy ją ująć w cudzysłów.

LEFT$ (*łańcuch-X*, *n*)

Instrukcja tworzy nowy łańcuch biorąc *n* pierwszych znaków z łańcucha *łańcuch-X*.

Przykłady:

1)
```
10 A$ = "INSTRUKCJA"
20 B$ = LEFT$(A$, 3)
30 PRINT B$
 RUN
INS
```

2) 10 DATA CPM, OLA, BAR, RET, ALFA
20 FOR I=1 TO 5 : READ A$(I) : NEXT
30 B$=""
40 FOR I=1 TO 5 : B$=B$+LEFT$(A$(I),1) : NEXT
50 PRINT B$
RUN
COBRA

RIGHT$ (*łańcuch-X*, *n*)

Instrukcja tworzy nowy łańcuch składający się z *n* ostatnich znaków łańcucha *łańcuch-X*.
Przykład:

```
10 A$="SPECTRUM"
20 B$= RIGHT$(A$,3)
30 PRINT B$
 RUN
RUM
```

MID$ (*łańcuch-X*, *n*, *m*)

Instrukcja tworzy nowy łańcuch z *łańcucha-X* biorąc *m* kolejnych znaków zaczynając od pozycji *n*.
Przykład:

```
10 A$="COBRA " : B$= "KONGRES " : C$= "SPUTNIK "
20 B$=LEFT$(A$,3)+MID$(B$,5,2)+LEFT$(C$,3)
30 PRINT B$
 RUN
COBRESPU
```

STRING$ (*k*, *argument*)

Instrukcja generuje łańcuch składający się z *k* takich samych elementów, jakim jest *argument*.

```
PRINT STRING$ (10, 65)
AAAAAAAAAA
PRINT STRING$ (10, "A")
AAAAAAAAAA
```

Parametr *k* oraz wartość dziesiętna odpowiadająca *argumentowi* może zawierać się w przedziale $0 \div 255$.

ASC (*łańcuch*)

Instrukcja powoduje wyznaczenie wartości dziesiętnej kodu ASCII pierwszego elementu (z lewej strony) *łańcucha* będącego operandem.

```
A$ = "ABECADLO"
PRINT ASC(A$)
65
```

CHR$ (*wyrażenie-numeryczne*)

Instrukcja tworzy łańcuch jednoelementowy. Wartość wyrażenia numerycznego jest traktowana jako dziesiętna postać kodu ASCII wyznaczonego w ten sposób znaku alfanumerycznego. Jest to instrukcja o działaniu odwrotnym do ASC. Wartość *wyrażenia-numerycznego* zawiera się w zakresie $0 \div 255$. Dzięki tej instrukcji można generować nie tylko cyfry i litery, ale też znaki sterujące (np. LF, CR, itd.) i semigraficzne.

```
PRINT CHR$(65)
A
```

STR$ (*wyrażenie-numeryczne*)

Instrukcja tworzy zmienną łańcuchową z wyrażenia numerycznego. Utworzony w ten sposób łańcuch zawiera jedną spację przed pierwszym znakiem łańcucha, jeżeli wyrażenie było liczbą dodatnią. Podczas wyświetlania funkcja PRINT dołącza jedną spację na końcu. W przypadku tworzenia łańcucha z liczby ujemnej, znak minus wyświetlany jest zamiast pierwszej spacji. Na łańcuchach utworzonych za pomocą instrukcji STR$ (...) można realizować tylko operacje łańcuchowe.

Przykład:

```
10 A = 1985
20 B$ = " ANNO DOMINI"
30 PRINT B$ + STR$(A)
 RUN
ANNO DOMINI 1985
```

VAL (*łańcuch*)

Instrukcja realizuje funkcję odwrotną do STR$ (...) i wyznacza wartość oznaczoną znakami numerycznymi *łańcucha*. Jeżeli łańcuch jest zbudowany ze znaków numerycznych, to w utworzonej liczbie są one wszystkie uwzględnione. Jeżeli łańcuch zawiera znaki numeryczne rozdzielone znakami liter, to utworzona liczba zawiera tylko pierwsze cyfry. W przypadku, gdy pierwszym znakiem łańcucha jest litera, a cyfry znajdują się na dalszych pozycjach, wartość VAL (*łańcuch*) = 0.

Przykłady:

```
1) A$ = "123"
   PRINT VAL(A$)
   123
2) A$ = "123 ZLOTE I 20 GROSZY"
   PRINT VAL(A$)
   123
3) A$ = "WINNY 123 ZLOTE I 20 GROSZY"
   PRINT VAL(A$)
   0
```

INKEY$

Instrukcja tworzy jednoznakowy łańcuch odpowiadający naciśniętemu znakowi. Jeżeli nie został naciśnięty klawisz (znak), to wynikiem działania funkcji jest łańcuch pusty. Instrukcja nadaje się szczególnie do realizacji programów gier.

Przykład:

```
10 CLS
20 A$=INKEY$ : IF A$="" THEN 20
30 PRINT TAB(ASC(A$)-64) ; A$
40 GOTO 20
 RUN
```

Przykładowy program powoduje generację kolumny utworzonej z litery naciśniętego klawisza. Dla litery A kolumna jest utworzona na pozycji 1, dla B na drugiej itd.

LEN (*łańcuch*)

Instrukcja służy do wyznaczania długości *łańcucha* i wartość ta zostaje przypisana zmiennej o tej samej nazwie.

Przykłady:

```
1) A$ =" TO " : B$ =" JEST " : C$ =" FUNKCJA "
   PRINT LEN(A$),LEN(B$),LEN(C$)
   4                6
   9
```

```
2) 10 INPUT A$
   20 FOR I=LEN(A$) TO 1 STEP -1
   30 PRINT MID$(A$,I,1) ;
   40 NEXT I
   50 PRINT : GOTO 10
```

Wartość LEN(A$) określa liczbę liter łańcucha, które należy wyświetlić w odwrotnej kolejności niż zostały podane.

FRE (*łańcuch*)

Funkcja powoduje wyznaczenie ilości wolnych bajtów pamięci przeznaczonej na przechowywanie łańcuchów. Po zainicjowaniu programu Basic automatycznie jest rezerwowane 50 bajtów.

Przykład:

```
PRINT FRE(A$)
50
A$ =" COBRA " : PRINT FRE(A$)
45
```

4

Tryby pracy

Mikrokomputer z językiem Basic może pracować w jednym z czterech trybów pracy:

— bezpośrednim
— edycji
— pracy w programie
— systemowym

4.1. Tryb bezpośredni — opis zleceń

AUTO *numer-pierwszej-linii, krok.*

Zlecenie powoduje włączenie automatycznej numeracji kolejnych linii programu napisanego w języku Basic. Po słowie AUTO należy podać numer pierwszej linii, od której będzie zaczynał się program oraz parametr *krok*. Numery następnych linii będą różniły się o krotność tego parametru, tj. *krok*, 2∗*krok*, 3∗*krok* itd. Jeżeli pominie się parametr *krok*, to Basic przyjmie jego wartość równą 10.

Przykład:

AUTO 5,5 (CR) — pierwszej linii programu zostanie przyporządkowany numer 5, a następnym 10, 15, 20, ... itd.,

AUTO 5 (CR) — pierwsza linia ma numer 5, a następne 15, 25, 35, ... itd.

Zakończenie wprowadzania programu z automatyczną numeracją linii odbywa się przez użycie klawisza BREAK (CTR A).

CLEAR *k*

Zlecenie powoduje wyzerowanie wszystkich zmiennych numerycznych i łańcuchowych (zmienne łańcuchowe przyjmą wartość łańcuchów pustych) oraz zarezerwuje k bajtów pamięci operacyjnej na zmienne łańcuchowe.

CLOAD "*nazwa*"

Zlecenie powoduje wczytanie z taśmy magnetofonowej programu napisanego w języku Basic i uprzednio zapisanego na taśmie za pomocą rozkazu CSAVE "nazwa". Podczas przeszukiwania kasety programy o nazwach innych niż podana po słowie CLOAD są ignorowane. Nazwa programu jest sprawdzana z dokładnością do pierwszej litery. Jeżeli nie podamy nazwy programu, to do mikrokomputera zostanie wczytany pierwszy napotkany program.

Każdy program zapisany na taśmie jest poprzedzony sekwencją znaków tworzących rozbiegówkę oraz bajtem startowym. W momencie wykrycia tej sekwencji, w prawym górnym rogu ekranu pojawią się dwie gwiazdki **. Druga gwiazdka miganiem sygnalizuje wczytywanie programu z taśmy. W przypadku napotkania programu o innej nazwie niż zadeklarowana, jego nazwa zostanie wyświetlona w miejscu pierwszej gwiazdki. Każdy wczytany nowy program niszczy poprzedni znajdujący się w pamięci. Jeżeli podamy rozkaz CLOAD i w pamięci znajduje się poprzedni program, a jednocześnie Basic nie wykrył jeszcze sekwencji startowej, to możemy zrezygnować z wczytania nowego programu kasując mikrokomputer przyciskiem RESET. Po uruchomieniu języka Basic instrukcją G : 66 (CR) (wejście gorące), możemy używać dalej poprzedniego programu w języku Basic, ponieważ nie został on skasowany.

CLOAD? "*nazwa*"

Zlecenie służy do weryfikacji programu zapisanego na taśmie magnetofonowej z programem zawartym w pamięci operacyjnej. W przypadku napotkania różnicy między programami, wyświetlony zostanie komunikat "BAD". Podczas weryfikacji programu, bez względu na jej wynik

końcowy, program w pamięci operacyjnej nie będzie zniszczony lub zmodyfikowany.

CONT

Zlecenie powoduje wznowienie programu wstrzymanego przy pomocy BREAK lub instrukcji STOP. Po zatrzymaniu programu istnieje możliwość zmiany wartości zmiennych pod warunkiem, że nie będzie dokonana zmiana treści programu.

CSAVE "*nazwa*"

Zlecenie powoduje zapisanie na taśmie magnetofonowej programu znajdującego się w pamięci operacyjnej i napisanego w języku Basic.

DELETE *numer-1 – numer-2*

Zlecenie powoduje usunięcie z programu napisanego w języku Basic wszystkich linii programu od linii *numer-1* do linii *numer-2*.

DELETE 60–90 usunięcie linii od nr 60 do 90, włącznie z linią 60 i 90;
DELETE –40 usunięcie wszystkich linii od początku do linii 40 włącznie;
DELETE 100 usunięcie linii 100.

EDIT *numer-linii*

Wprowadza Basic w tryb edycji (patrz opis zleceń EDIT).

LIST *numer-1 – numer-2*

Zlecenie powoduje wyświetlenie na ekranie linii o numerach zadeklarowanych za słowem LIST.

LIST 10–70 wyświetlenie linii od nr 10 do 70;
LIST –70 wyświetlenie wszystkich linii od początku do 70;
LIST 70– wyświetlenie wszystkich linii od linii 70 do końca programu;
LIST 100 wyświetlenie linii 100.

LLIST *numer-1 – numer-2*

Zlecenie służy do wyprowadzania treści programu na urządzenie zewnętrzne np. drukarkę lub perforator. Działa tak jak LIST. Program

wyprowadzany na drukarkę za pomocą zlecenia LLIST nie jest wyświetlany na ekranie.

NEW

Zlecenie powoduje wyzerowanie wszystkich zmiennych i usuwa program z pamięci operacyjnej. Nie powoduje zmiany deklaracji obszaru zarezerwowanego dla zmiennych łańcuchowych w ostatniej deklaracji CLEAR *k*.

RUN *numer-linii*

Rozpoczyna wykonanie programu od zadanego numeru linii. Jeżeli po słowie RUN nie będzie podanego numeru linii, wykonanie programu rozpocznie się od najniższego numeru linii. Instrukcja RUN zeruje wszystkie zmienne. Nie powoduje zmiany ostatniej deklaracji CLEAR *k*.

SYSTEM

Wprowadza Basic w tryb systemowy. Tryb ten umożliwia wpisywanie programów lub zbiorów danych i wykonanie programów w języku maszynowym. Po wywołaniu SYSTEM CR, na ekranie zostanie wyświetlona gwiazdka i znak zapytania *?. Po podaniu nazwy programu jest rozpoznawana pierwsza litera bez cudzysłowu i wczytany zostaje program z taśmy. Po wczytaniu programu pojawia się znowu znak *? w nowej linii. Podanie znaku dzielenia "/" i adresu dziesiętnego zakończonego znakiem CR spowoduje wykonanie programu. Jeżeli po znaku dzielenia "/" nie zostanie podany adres, to program będzie wykonywany od adresu zawartego w części organizacyjnej programu zapisanego na taśmie.

TROFF

Zlecenie powoduje wyłączenie funkcji śledzenia przebiegu wykonywanego programu.

TRON

Zlecenie powoduje włączenie funkcji śledzenia przebiegu realizowanego programu. Śledzenie to polega na ciągłym wyświetlaniu numerów linii wykonywanych po sobie w programie.

4.2. Tryb edycji

Proces poprawiania programu za pomocą funkcji EDIT może odbywać się tylko w obrębie jednej linii programu. Do trybu EDIT można przejść przez napisanie słowa EDIT i dalej za nim numeru linii. W przypadku działającego programu, w sytuacji napotkania błędu w danej linii, Basic sam przechodzi do trybu EDIT. Po wywołaniu trybu EDIT, Basic wyświetla numer linii z błędem i czeka na znaki sterujące, które w zależności od rodzaju poprawki, poda operator.

Przykład wywołania trybu EDIT:
EDIT 200 (CR)
200 _

W takiej sytuacji treść linii przeznaczonej do poprawek została przepisana do tymczasowego bufora linii utworzonego w pamięci. Po podaniu, po słowie EDIT, numeru linii nieistniejącej w programie, zostanie wyświetlony błąd ?UL. (?UL ERROR).

4.2.1. Opis funkcji trybu EDIT

Funkcja A

powoduje skasowanie poprawek wykonanych w danej linii programu. Do bufora linii zostaje ponownie przepisana wywołana linia i wyświetlony jej numer. Program Basic jest nadal w trybie edycji danej linii, ale nie przechodzi do trybu bezpośredniego.

Funkcja Q

powoduje usunięcie wszystkich poprawek dokonanych w danej linii przy użyciu funkcji EDIT i kończy edycję wychodząc do trybu bezpośredniego. Zawartość poprawianej linii po wyjściu jest bez zmian.

Funkcja E

powoduje zakończenie edycji linii i wyjście do trybu bezpośredniego. Wszystkie zmiany zostały zaakceptowane i zawartość bufora linii zostaje umieszczona w programie w miejsce pierwotnej treści.

Funkcja CR

powoduje wyjście z trybu edycji w każdej sytuacji. Wychodząc z trybu EDIT, Basic powoduje wprowadzenie wszystkich poprawek w danej linii, wyświetlenie całej linii programu bez względu na położenie kursora i przejście do trybu bezpośredniego.

Funkcja L

powoduje wyświetlenie całej linii programu bez względu na pozycję kursora. Po wyświetleniu treści poprawianej linii, kursor przechodzi do nowej linii na ekranie i zostaje wyświetlony ponownie numer linii. Przy wykonywaniu funkcji L program Basic nie może być w zleceniu edycji.

4.2.2. Zlecenia edycji

Do znaków sterujących edycją należą: SP, ←, CR, CTR [(szczegółowy opis znaków — patrz rozdział "Opis klawiszy funkcyjnych").

I — wstaw (*insert*)

Zlecenie powoduje wstawienie tekstu od aktualnej pozycji kursora. Wyjście z trybu I odbywa się za pomocą CTR [. Po wyjściu CTR [, Basic oczekuje na znak nowego zlecenia edycji. Jeżeli wstawiony nowy tekst jest jedyną poprawką w tej linii, to można od razu wyjść z edycji naciskając CR. Wstawiony nowy tekst powoduje rozsunięcie znaków w edytowanej linii.

X — wyświetl linię + I (insert)

Zlecenie powoduje wyświetlenie wszystkich znaków w edytowanej linii i ustawienie kursora na pierwszej wolnej pozycji oraz automatyczne przejście do zlecenia I (insert). Zlecenie to umożliwia dopisanie dalszych instrukcji w istniejącej już linii programu. Kończąc poprawianie linii możemy wyjść naciskając CR. Jeżeli po dopisaniu nowej instrukcji chcemy wyjść ze zlecenia X i nadal pozostać w EDIT, to naciskamy CTR [.

H — kasuj na prawo + I (insert)

Zlecenie powoduje usunięcie wszystkich znaków z bufora linii od aktualnej pozycji kursora do końca linii oraz automatycznie przechodzi do trybu I (insert). Wyjście za pomocą CTR [lub CR.

*k***D** — usuń *k* znaków

Zlecenie powoduje usunięcie *k* znaków od następnej pozycji kursora. Usunięte znaki zostają wyświetlone pomiędzy ukośnymi kreskami (znakiem dzielenia).

*k***C** — zastąp *k* znaków

Zlecenie powoduje usunięcie *k* znaków od aktualnej pozycji kursora i przechodzi w mod I (insert). Po wprowadzeniu nowych *k* znaków następuje automatyczne wyjście z I i Basic oczekuje na dalszą edycję.

*k***S** *znak* — poszukiwanie *znaku* w linii

Zlecenie powoduje ustawienie kursora na *k*-te wystąpienie poszukiwanego *znaku* zaczynając analizę linii programu na prawo od aktualnej pozycji kursora. Jeżeli szukany znak nie wystąpi, to kursor zajmie pierwszą wolną pozycję na końcu linii tekstu.

*k***K** *znak* — usuń do *k*-tego *znaku*

Zlecenie powoduje usunięcie z bufora linii wszystkich znaków od aktualnego położenia kursora do *k*-tego wystąpienia wskazanego *znaku* (bez usunięcia wskazanego znaku). Usunięty łańcuch znaków zostanie wyświetlony na ekranie pomiędzy znakami dzielenia (/).

Przykłady działania funkcji EDIT i zleceń edycji.

Przykład 1.

W pamięci operacyjnej znajdują się następujące błędne linie programu:

```
10 IPUT A,B
20 FOR I=1 TO 100 STEP -1
```

W pierwszej linii należy wstawić literę N po literze I (INPUT) oraz dopisać zmienną C po zmiennej B. Natomiast w drugiej linii należy usunąć drugie 0 z liczby 100 oraz zmienić parametr za słowem STEP z −1 na 2.

EDIT 10	CR
10 _	SP (wyświetl pierwszy znak)
10 I_	I (insert)
10 I__	N (wstaw literę N)
10 IN__	CTR [(wyjście z trybu I)
10 IN__	X (wyświetl linię do końca)
10 INPUT A,B__	,C (dopisanie ,C)
10 INPUT A,B,C__	CR lub E (kończy EDIT)
EDIT 20	CR
20 _	13SP
20 FOR I=1 TO 10__	1D (usuń jeden znak)
20 FOR I=1 TO 10/0/__	X (wyświetl linię do końca)
20 FOR I=1 TO 10/0/ STEP −1__	2 razy SH+A
20 FOR I=1 TO 10/0/ STEP __	2 (dopisz 2)
20 FOR I=1 TO 10/0/ STEP 2__	CR lub E kończy EDIT

Po usunięciu błędów linie 10 i 20 mają następującą postać:

```
10 INPUT A,B,C
20 FOR I=1 TO 10 STEP 2
```

Przykład 2.

```
10 DATA 11,13,17,22,33
```

Dane w DATA należy doprowadzić do postaci:

```
10 DATA 11,13,15
```

EDIT 10	CR
10 _	12SP
10 DATA 11,13,1__	H (kasuj na prawo + insert)
10 DATA 11,13,1__	5 (dopisz 5)
10 DATA 11,13,15__	CR lub E kończy EDIT

Powyższą zmianę można wprowadzić następująco:

EDIT 10	CR
10 _	1S7 (odszukanie pierwszej cyfry 7)
10 DATA 11,13,1__	H (kasuj na prawo + insert)
10 DATA 11,13,1__	5 (dopisz 5)
10 DATA 11,13,15	CR lub E kończy EDIT

Przykład 3.

10 DATA 1223,17,22, 33,44, 55

Z linii 10 należy usunąć liczby 1223 i 17.

EDIT 10	CR
10 __	5SP
10 DATA __	3K2 (usuń znaki od kursora do trzeciego wystąpienia cyfry 2 bez jej usunięcia)
10 DATA 1223,17, __	CR lub E kończy EDIT

Przykład 4.

10 IF A >= B THEN 40

Ciąg znaków A >= B należy zastąpić ciągiem znaków C <= D.

EDIT 10	CR
10 __	3SP
10 IF __	4C
10 IF __	C <= D (dopisz C <= D)
10 IF C <= D__	CR lub E kończy EDIT

4.3. Tryb realizacji programu

Mikrokomputer wykonuje program w języku Basic po podaniu rozkazu RUN. Rozpoczynając wykonanie programu Basic zeruje wszystkie zmienne numeryczne, a zmiennym łańcuchowym przypisuje łańcuchy puste.

4.4. Tryb systemowy

Umożliwia wywołanie programów w języku maszynowym — patrz SYSTEM w trybie bezpośrednim.

5

Opis klawiszy funkcyjnych

CR

Kończy wprowadzanie informacji przez użytkownika we wszystkich trybach pracy.

SP

Znak spacji.

SH

SHIFT. Przełącza klawiaturę na górny rejestr znaków.

CTR (SH + W)

Klawisz funkcyjny. Powoduje wygenerowanie kodu dużej litery pomniejszonego o 64, np. C = 67, CTR + C = 3

← (SH + A)

Cofnięcie kursora i skasowanie ostatniego znaku z ekranu.

→ (SH + S)

Znak tabulacji poziomej. Powoduje przesuwanie kursora do każdej pierwszej pozycji tabulacji poziomej, tj. 0,8,16,24...

↓ (SH + Z)

Znak nowej linii. Powoduje przesunięcie kursora do pierwszej pozycji nowej linii.

CTR Y (SH + W,Y)

Znaki są pisane na ekranie w polach o parzystych numerach. Instrukcja CLS kasuje CTR Y.

CTR X (SH + W,X)

Kasowanie pisanej linii i cofnięcie kursora do początku linii.

CTR [(SH + W,SH + U)

Znak powoduje wyjście ze zleceń trybu EDIT.

CTR A (SH + W,A)

BREAK. Przerwanie wykonywanego programu.

CTR L (SH + W,L)

CLEAR. Kasowanie całego ekranu i ustawienie kursora na pozycję HOME oraz kasowanie CTR Y.

SH P (SH + P)

Powoduje skasowanie całego ekranu bez zmiany położenia kursora.

SH I (SH + I)

Znak pauzy. Powoduje zatrzymanie wykonywanego programu łącznie z zatrzymaniem wyświetlania komunikatów i wyników obliczeń. Akcja programu jest wznawiana po puszczeniu klawisza I.

6

Obsługa mikrokomputera

Po włożeniu taśmy magnetofonowej z programem Basic do magnetofonu, należy ustawić właściwy poziom sygnału na jego wyjściu głośnikowym przy pomocy regulatora siły głosu. Przy poprawnie ustawionym poziomie sygnału na wyjściu, poziome paski wyświetlane na ekranie wokół pola alfanumerycznego podczas trwania tzw. rozbiegówki będą węższe niż przerwy między nimi. Po ustawieniu poziomu sygnału należy cofnąć taśmę do początku rozbiegówki i przy pomocy procedury L systemu MONITOR COBRA wczytać program z taśmy do pamięci operacyjnej. Przy prawidłowo wczytującym się programie mikrokomputer po przyjęciu kolejnego bloku programu Basic, sygnalizuje sygnałem dźwiękowym oraz zgaszeniem lub zapaleniem prawego górnego pola ekranu gotowość przyjęcia następnego bloku. Po wczytaniu się całego programu pojawi się napis STOP TAPE!. W przypadku wystąpienia przekłamania pojawi się napis TAPE ERROR, przestanie migać narożne pole oraz nie będzie uruchamiany sygnał dźwiękowy. W tej sytuacji należy zatrzymać magnetofon i cofnąć taśmę o 20 ÷ 30 cm oraz uruchomić magnetofon powtórnie. Program na taśmie jest podzielony na 32-bajtowe *bloki*, z których każdy posiada w pamięci *adres*, od którego ma być wpisany oraz *sumę kontrolną*. Taka organizacja w przypadku wystąpienia przekłamania podczas wczytywania programu nie wymaga przewijania do początku taśmy. Po wczytaniu programu Basic należy go uruchomić za pomocą instrukcji:

G : 0 CR

Jeżeli istnieje możliwość podłączenia np. drukarki poprzez układ PIO lub USART, to przed uruchomieniem języka Basic należy dopisać dwie procedury w języku maszynowym. Pierwsza procedura będzie programować nasz układ PIO lub USART i musi ona znajdować się w obszarze pamięci od adresu 3000H do 300FH. Procedura musi kończyć się instrukcją zerowania akumulatora i skoku do adresu 0674H (AF, C3, 74, 06). Druga procedura realizuje wysłanie znaku na drukarkę oraz badanie gotowości przyjęcia nowego znaku przez drukarkę. Procedurę tę należy umieścić w obszarze pamięci od adresu 3010H i zakończyć instrukcją RET(C9). Przy wywołaniu procedury wysłania znaku na drukarkę, program powoduje umieszczenie znaku w rejestrze A mikroprocesora (w akumulatorze). Po umieszczeniu obu procedur w pamięci, można je zapisać na taśmie bezpośrednio za językiem Basic za pomocą instrukcji SAVE.

S : 3000,302F (włączyć magnetofon na nagrywanie i nacisnąć CR).

W ten sposób po wczytaniu języka Basic oraz procedur dodatkowych i po uruchomieniu programu, możemy używać takich instrukcji jak LPRINT, LLIST, itd.

Chcąc korzystać z instrukcji graficznych, jak SET, RESET należy wymienić generator znaków na pamięć EPROM 2716 oprogramowaną według zamieszczonego wydruku.

Po naciśnięciu CR ekran zostanie skasowany oraz w lewym górnym rogu pojawi się napis MEMORY SIZE? —

Język Basic wymaga minimum 32 kbajty pamięci, ponieważ sam zajmuje obszar od 0 do 302F, a zmienne oraz jego pamięć operacyjna są umieszczone począwszy od adresu 4000H. Przy obszarze pamięci 32 kbajty po pytaniu MEMORY SIZE? należy podać liczbę 32500 i zakończyć znakiem CR. Jeżeli w mikrokomputerze jest 48 kbajtów pamięci RAM, to możemy nie podawać liczby 49151, a tylko nacisnąć znak CR. Po naciśnięciu CR, Basic sam określi koniec pamięci operacyjnej. Po uruchomieniu języka Basic, na ekranie pojawi się napis:

```
MEMORY SIZE? 32500
*BASIC COBRA*
READY
>—
```

Pozioma kreska za znakiem większości jest kursorem i wskazuje miejsce wprowadzenia następnego znaku.
Ekran posiada organizację: 32 znaki w linii i 24 linie, czyli 768 pól alfanumerycznych. Przy użyciu instrukcji graficznych SET i RESET, na ekranie można umieścić 64 punkty poziomo i 72 pionowo. W lewym górnym rogu znajduje się pole alfanumeryczne oznaczone numerem 0, a w prawym dolnym — pole o numerze 767. Dla instrukcji graficznych współrzędna X przyjmuje wartość od 0 do 63 (0 z lewej strony ekranu) a współrzędna Y od 0 do 71 (0 na górze ekranu). W normalnym trybie pracy w linii ekranu znajdują się dwa pola szesnastoznakowe.
Podczas redakcji tekstu możemy przesuwać kursor do początku pól ośmioznakowych za pomocą znaku → (SH+S) i wtedy będą to pola o numerach 0, 8, 16, 24. Jest to tabulacja pozioma.

7

Wykaz sygnalizowanych błędów

KOD	OZNACZENIE	PRZYCZYNA WYSTĄPIENIA BŁĘDU
1	NF	**NEXT without FOR** Brak instrukcji FOR przed użytą instrukcją NEXT.
2	SN	**Syntax Error** Błąd składni.
3	RG	**RETURN without GOSUB** Brak instrukcji GOSUB przed użytą instrukcją RETURN.
4	OD	**OUT of DATA** Brak danych dla instrukcji READ z instrukcji DATA. Występuje też przy instrukcji INPUT # −1 w przypadku braku danych na taśmie.
5	FC	**Illegal Function Call** Nieprawidłowe użycie funkcji (np. LOG(0), SQR(−5), RND(−3)).
6	OV	**Overflow** Wartość liczbowa użyta w operacji przekracza zakres argumentu użytej instrukcji (np. dla CINT liczba mniejsza niż −32768).

KOD	OZNACZENIE	PRZYCZYNA WYSTĄPIENIA BŁĘDU
7	OM	**Out of Memory** Brak pamięci dla programu lub macierzy.
8	UL	**Undefined Line** Skok do nieistniejącej linii programu.
9	BS	**Subscript out of Range** Wskaźnik elementu macierzy poza zadeklarowanym wymiarem.
10	DD	**Redimensioned Array** Próba deklaracji macierzy (DIM) o nazwie macierzy już istniejącej.
11	/0	**Division by zero** Dzielenie przez zero.
12	ID	**Illegal Direct** Próba wykonania INPUT w trybie bezpośrednim.
13	TM	**Type Mismatch** Użycie w wyrażeniu arytmetycznym zmiennej łańcuchowej i odwrotnie (np. A$ = 123, SIN(A$)). Występuje też w sytuacji, gdy zmienna pętli FOR-NEXT była deklarowana przez instrukcję DEFDBL.
14	OS	**Out of String Space** Zapełnienie obszaru pamięci łańcuchów.
15	LS	**String Too Long** Długość łańcucha większa niż 255 znaków.
16	ST	**String Formula Too Complex** Operacja zbyt skomplikowana na zmiennych łańcuchowych. Wykonać przy pomocy kilku prostszych.

KOD	OZNACZENIE	PRZYCZYNA WYSTĄPIENIA BŁĘDU
17	CN	**Can not Continue** Brak możliwości dalszego wykonywania programu po BREAK.
18	NR	**NO RESUME** Brak instrukcji RESUME po ON ERROR GOTO.
19	RW	**RESUME without ERROR** Brak instrukcji ON ERROR GOTO przed użytą instrukcją RESUME.
20	UE	**Unprintable Error** Kod błędu w instrukcji ERROR jest większy od 23.
21	MO	**Missing Operand** Brak parametru (np. PRINT 2[).
22	FD	**BAD File Data** Błędne dane z taśmy lub inna kolejność.

8

Adresy wybranych procedur maszynowych języka Basic-COBRA

Wszystkie kody i adresy są podane w kodzie heksadecymalnym.

Adres HEX.

0002 Skok do procedury programującej układ współpracujący z drukarką.

0028 Skok do BREAK. Po wykonaniu instrukcji:
POKE 16396, 175 : POKE 16397, 201
((400C)←AF, (400D)←C9) nastąpi zablokowanie działania BREAK.

0049 Pobierz znak z klawiatury.

0066 Start BASIC (tzw. „gorący").

0132 POINT A = 00 (A — argument)

0135 SET A = 80

0138 RESET A = 01

013A Wyznaczenie adresu pamięci RAM ekranu i maski dla funkcji semigraficznych.

019D INKEY$

0169 CLS — kasowanie ekranu.

01D3 Odczytanie rejestru R (REFRESH) dla RANDOM.

01D9 Zapisanie impulsu na taśmę magnetofonową.

022C Miganie gwiazdki w prawym górnym rogu ekranu.

0235 Czytanie bajtu z taśmy — wynik w A.

0241 Czytanie bitu z taśmy i składanie bajtu
(A = 2*A + bit)

0261 Dwukrotne zapisanie bajtu z A na taśmę.

0264 Zapisanie bajtu z A na taśmę.

0284 Zapis 255 bajtów zer (00) i bajtu A5 na taśmę, tzw. „rozbiegówka".

0293 Czytaj taśmę i czekaj na znak A5 oraz zapalenie gwiazdki na ekranie.

02A9 Czytanie dwóch bajtów będących adresem startu dla SYSTEM.

02B2 SYSTEM

02CE Wczytanie zbioru z taśmy.

0314 Wczytanie z taśmy dwóch bajtów do pary rejestrów HL.

032A Znak w A. W zależności od zawartości komórki pamięci (409C) znak z A jest zapisany
— na ekran, gdy (409C) = 00
— na drukarkę, gdy (409C) = 01
— na taśmę, gdy (409C) = 80

033A Znak z A na ekran i modyfikacja POS w komórce 40A6.

0348 Ustawienie nowej wartości POS w A.

0361 Wczytanie max. 240 znaków do bufora. W parze rejestrów (HL) jest adres bufora.

038B Wysłanie znaku CR (0D) na drukarkę.
Jeżeli głowica nie jest na pozycji 0, to wtedy do komórki (409C) jest wpisane 0.

039C Wysłanie znaku na drukarkę. Komórka pamięci (409B) jest licznikiem znaków.
Dla znaków sterujących LF,FF,CR (0A,0C,0D) przyjmuje wartość 0.

03C2 Obsługa wejścia-wyjścia.

03E3 Szybka obsługa klawiatury.

0458 Wypisanie znaku na ekran.

04C0 Kursor na pozycję HOME (0C).

04CE Cofnięcie kursora i skasowanie ostatniego znaku z ekranu — BACKSPACE (08)

04DA Cofnięcie kursora ← (18).
04E7 Przesunięcie kursora w dół ↓ (1A).
04EC Przesunięcie kursora na następną pożycję → (19).
04F1 Przesunięcie kursora do góry ↑ (1B).
04F6 Pisanie znaków na ekranie tylko po parzystych wartościach POS. Parametr w (403D).
0506 Dekodowanie kodów sterujących procedury ekranowej.
0541 Wypisanie znaku na ekran i ewentualnie przesuw całego obrazu o jedną linię w górę (Scroll).
0564 Nowa linia — odpowiada wysłaniu znaków CR i LF, gdzie:
CR(0D) — powrót kursora do początku linii.
LF(0A) — zmiana linii.
0573 Kasowanie wszystkich znaków od pozycji kursora do końca linii (1E).
057C Kasowanie wszystkich znaków od pozycji kursora do końca ekranu (1F).

Adresy funkcji matematycznych i podstawowych operacji arytmetycznych. Funkcje matematyczne operują daną umieszczoną w rejestrze "X", który jest zbudowany z komórek pamięci o następujących adresach:

	INT	STR	SNG	DBL
411D (16669)				LSB
411E (16670)				LSB
411F (16671)				LSB
4120 (16672)				LSB
4121 (16673)	LSB	LSB	LSB	LSB
4122 (16674)	MSB	MSB	LSB	LSB
4123 (16675) Adres wektora			MSB	MSB
4124 (16676) Adres wektora			EXP	EXP

Przy operacjach arytmetycznych jest wykorzystywany dodatkowo rejestr "Y" o podobnych właściwościach, jak "X". Zajmuje on komórki pamięci od adresu 4127(16678) do 412E(16686).

Funkcje matematyczne		Argument	Wynik
0977	X = ABS(X)	INT,SNG,DBL	
15BD	X = ATN(X)	INT,SNG,DBL	SNG
0ADB	X = CDBL(X)	INT,SNG,DBL	DBL
0A7F	X = HL = CINT(X)	INT,SNG,DBL	INT
0AB1	X = CSNG(X)	INT,SNG,DBL	SNG
1541	X = COS(X)	INT,SNG,DBL	SNG
1439	X = EXP(X)	INT,SNG,DBL	SNG
0B26	X = FIX(X)	INT,SNG,DBL	
0B37	X = INT(X)	INT,SNG,DBL	
0809	X = LOG(X)	INT,SNG,DBL	SNG
14C9	X = RND(X)	INT,SNG,DBL	SNG
098A	X = SGN(X)	INT,SNG,DBL	INT
1547	X = SIN(X)	INT,SNG,DBL	SNG
13E7	X = SQR(X)	INT,SNG,DBL	SNG
15A8	X = TAN(X)	INT,SNG,DBL	SNG
14F0	X = RND(0)		SNG

INT — całkowite
SNG — pojedynczej precyzji
DBL — podwójnej precyzji

Operacje arytmetyczne:

0B2D	X = DE + HL	INT
0BC7	X = DE – HL	INT
0BF2	X = DE*HL	INT
2490	X = DE/HL	INT
0716	X = BCDE + X	SNG
0713	X = BCDE – X	SNG
0847	X = BCDE*X	SNG
08A2	X = BCDE/X	SNG
0C77	X = X + Y	DBL
0C70	X = X – Y	DBL
0DA1	X = X*Y	DBL
0DE5	X = X/Y	DBL
093E	X = X*10	SNG
0E4D	X = X*10	DBL

0F0A	X = X*10	SNG,DBL
0897	X = X/10	SNG
0DDC	X = X/10	DBL
0F18	X = X/10	SNG,DBL
0708	X = X + .5	SNG
070B	X = (HL) + X	SNG
0710	X = (HL) − X	SNG
08A0	X = (SP)/X	SNG
097B	X = − X	SNG,DBL,INT

Symbol (SP) oznacza sekwencję:

08A0 POP BC
08A1 POP DE
08A2 ← X = BCDE/X

Symbol (HL) oznacza sekwencję:

LD E, (HL)
LD D, (HL + 1)
LD C, (HL + 2)
LD B, (HL + 3)

W obu tych operacjach ((SP), (HL)) argument typu SNG jest umieszczony w rejestrach BCDE. Rejestr B zawiera wykładnik potęgi (EXP), C zawiera MSB, a D i E — LSB. Wykładnik potęgi jest w postaci nadmiarowej (+ 80 HEX.).

Instrukcje porównania:

0018	CP HL,DE	HL i DE	HL = DE dla Z = 1			
0A0C	CP X,BCDE	SNG	X < BCDE	A = FF	CY = 1	Z = 0
			X = BCDE	A = 00	CY = 0	Z = 1
			X > BCDE	A = 01	CY = 0	Z = 0
0A39	CP HL,DE	INT	HL < DE	A = FF	CY = 1	Z = 0
			HL = DE	A = 00	CY = 0	Z = 0
			HL > DE	A = 01	CY = 0	Z = 0
0A4F	CP X,Y	DBL	X < Y	A = FF	CY = 1	Z = 0
			X = Y	A = 00	CY = 0	Z = 1
			X > Y	A = 01	CY = 0	Z = 0

Operacje pomocnicze do przesyłania argumentów:

09A4	(SP)=X	SNG
09B1	X=BCDE=(HL)	SNG
09B4	X=BCDE	SNG
09BF	BCDE=X	SNG
09C2	BCDE=(HL)	SNG
09CB	(HL)=X	SNG
09F4	X=Y	DBL
09F7	X=(HL)	SNG lub DBL
09FF	(HL)=X	SNG lub DBL
0A9A	X=HL	INT

W komórce o adresie 40AF znajduje się parametr VT i przyjmuje on wartość:

02 — INT
03 — STR
04 — SNG
08 — DBL

W zależności od parametru VT odpowiednio jest wykonywana operacja arytmetyczna.

9

Adresy wybranych komórek systemowych

400C ÷ 400E	Adres przerwania.
4020 ÷ 4021	Adres kursora w pamięci wizji.
4022	Znak z pola kursora.
4028	Maksymalna liczba wierszy na stronę drukarki + 1.
4029	Liczba wyprowadzonych wierszy na stronie.
4090 ÷ 4092	Mantysa mnożnika dla funkcji RND = 0.253514.
409A	Ostatni kod błędu.
409B	Liczba wyprowadzonych znaków w bieżącym wierszu drukarki.
40A0 ÷ 40A1	Początkowy adres pamięci łańcuchów.
40A4 ÷ 40A5	Adres początku programu napisanego w języku Basic.
40A6	Pozycja kursora w wierszu na ekranie dla funkcji POS.
40A7 ÷ 40A8	Wskaźnik bufora wiersza programu.
40AF	Typ VT dla zmiennej w rejestrze X.
40B1 ÷ 40B2	Adres szczytu obszaru dostępnej pamięci − 256.
40B3 ÷ 40B4	Adres najbliższej wolnej komórki pamięci w tabeli łańcuchów. Tabela od 40B5 do 40D2.
40E2 ÷ 40E3	Aktualny numer wiersza dla AUTO.
40E4 ÷ 40E5	Wartość kroku dla AUTO.
40EC ÷ 40ED	Numer ostatnio redagowanego wiersza.
40F0 ÷ 40F1	Numer wiersza ON ERROR GOTO.
40F2	Wskaźnik włączenia ON ERROR GOTO < > 0.

40F5 ÷ 40F6 Numer następnego wiersza dla CONT po BREAK, END, STOP lub ERROR.

40F9 ÷ 40FA Adres początku obszaru zmiennych.

40FB ÷ 40FC Adres obszaru macierzy.

40FD ÷ 40FE Adres początku wolnej pamięci.

40FF ÷ 4100 Adres następnej wartości DATA.

4101 ÷ 411A Obszar dla znacznika typu zmiennych. Kolejne komórki pamięci od adresu 4101 odpowiadają literom — nazwom zmiennych:

4101 — A

4102 — B

4103 — C

.

411A — Z

Podczas definiowania (DEFINT, DEFSTR, DEFSNG, DEFDBL) typu zmiennej, do komórek o podanych adresach są wpisywane kody:

02 — DEFINT

03 — DEFSTR

04 — DEFSNG

08 — DEFDBL

411B Znacznik funkcji śledzenia programu

00 = TROFF, AF = TRON

10

Mapa pamięci mikrokomputera COBRA

0000 ÷ 302F	Język Basic.
3030 ÷ 3FFF	Obszar niewykorzystany pamięci RAM. Może służyć do lokalizacji procedur napisanych w języku maszynowym i wywołanych instrukcją USR(X).
4000 ÷ 42E8	Pamięć RAM dla potrzeb języka Basic.
42E9 ÷ BE8C	Obszar RAM dla programu w języku Basic.
C000 ÷ C7FF	MONITOR COBRA.
F800 ÷ FAFF	RAM monitora ekranowego.

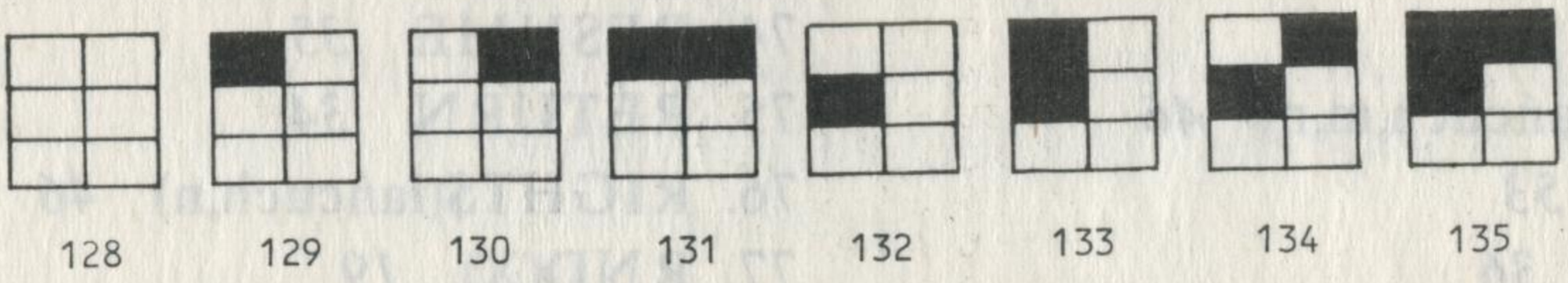
Dodatek 1

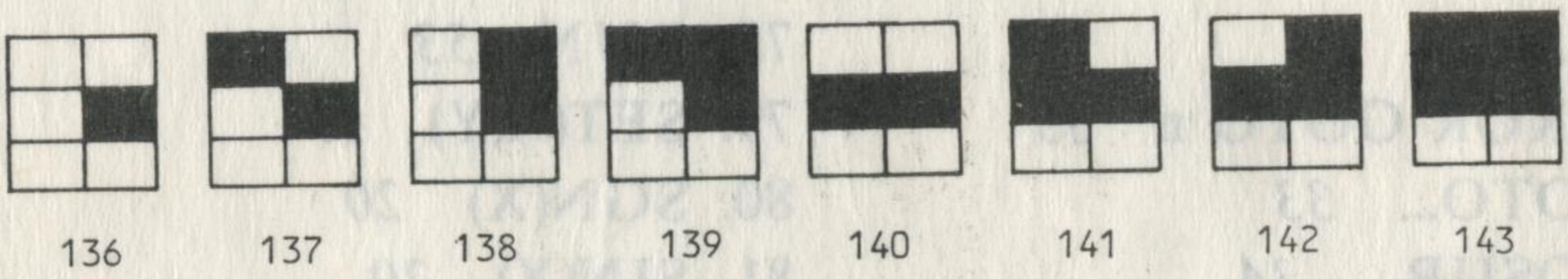
Alfabetyczny spis instrukcji języka Basic-COBRA

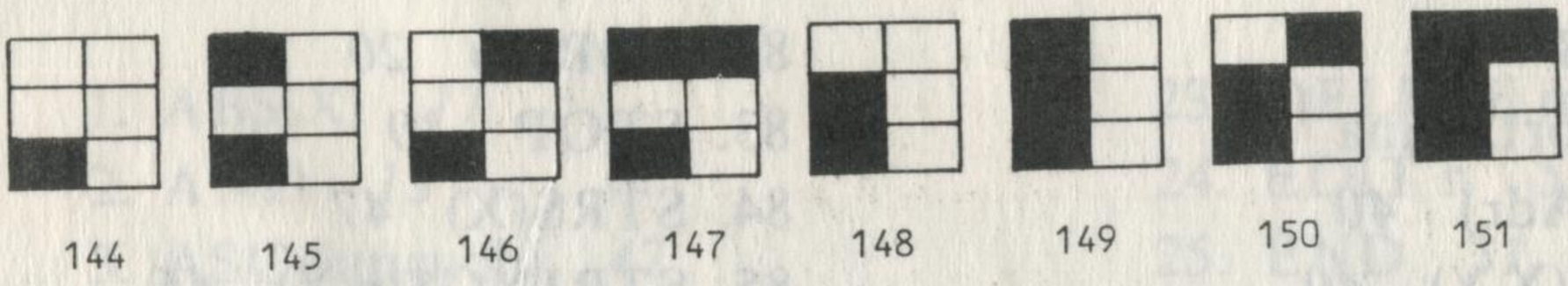
1. ABS(X) *17*
2. AND *13*
3. ASC(łańcuch) *47*
4. ATN(X) *17*
5. AUTOn,k *50*
6. CDBL(X) *17*
7. CHR$(X) *47*
8. CINT(X) *17*
9. CLEAR X *30*
10. CLOAD "nazwa" *51*
11. CLOAD? "nazwa" *51*
12. CLS *38*
13. CONT *52*
14. COS(X) *18*
15. CSAVE "nazwa" *52*
16. CSNG(X) *18*
17. DATA dana *27*
18. DEFDBL lista *31*
19. DEFINT lista *31*
20. DEFSNG lista *31*
21. DEFSTR lista *31*
22. DIM X(x_1, x_2, ...x_n) *32*
23. DELETE $n_1 - n_2$ *52*
24. EDIT n *52*
25. END *31*
26. ERL *38*
27. ERR/2+1 *38*
28. ERRORnumer *31*
29. EXP(X) *18*
30. FIX(X) *19*
31. FOR...TO...STEP *36*
32. FRE(łańcuch) *49*
33. GOSUB n *33*
34. GOTO n *33*
35. IF...THEN...ELSE *37*
36. INKEY$ *48*
37. INP(port) *42*
38. INPUT lista *26*
39. INPUT # −1,lista *27*
40. INT(X) *18*
41. LEFT$(łańcuch,n) *45*
42. LEN(łańcuch) *49*
43. LET zmienna =... *29*
44. LIST $n_1 - n_2$ *52*

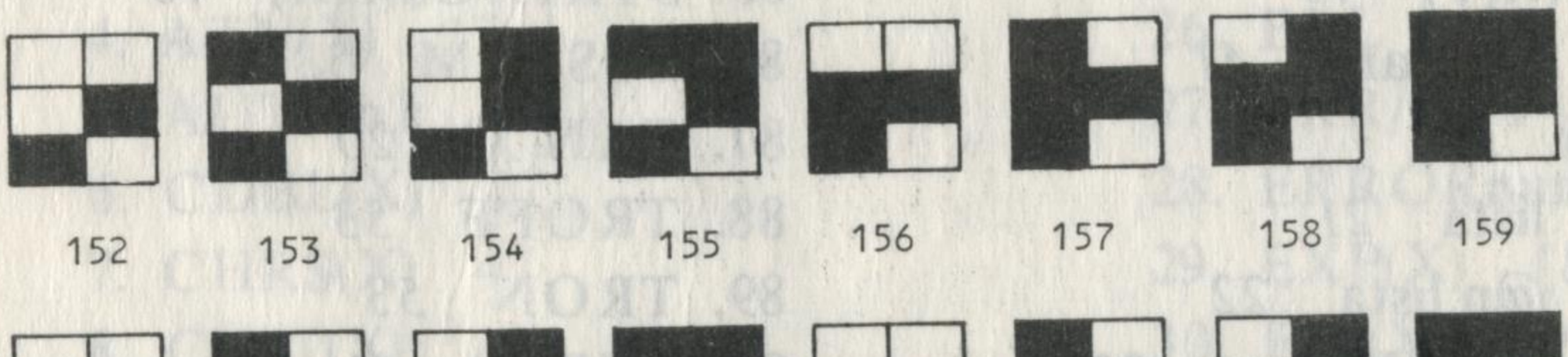
Dodatek 2

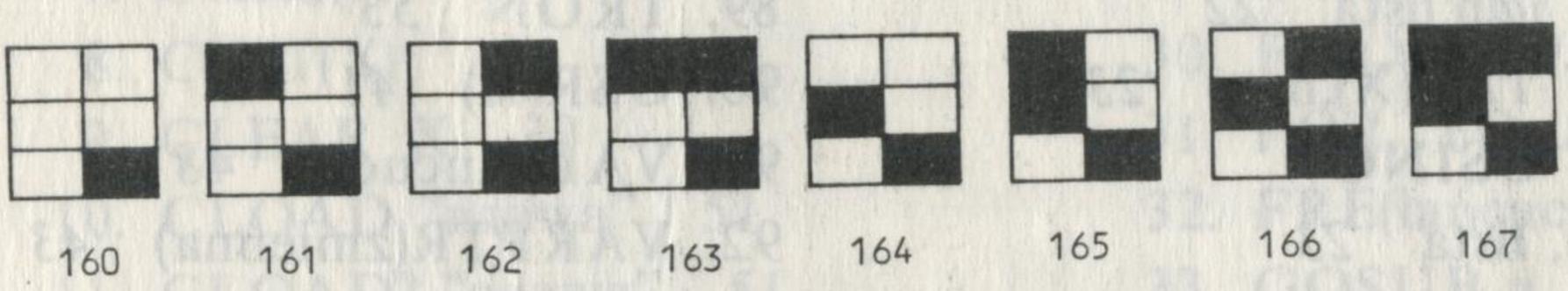
Wykaz znaków semigraficznych

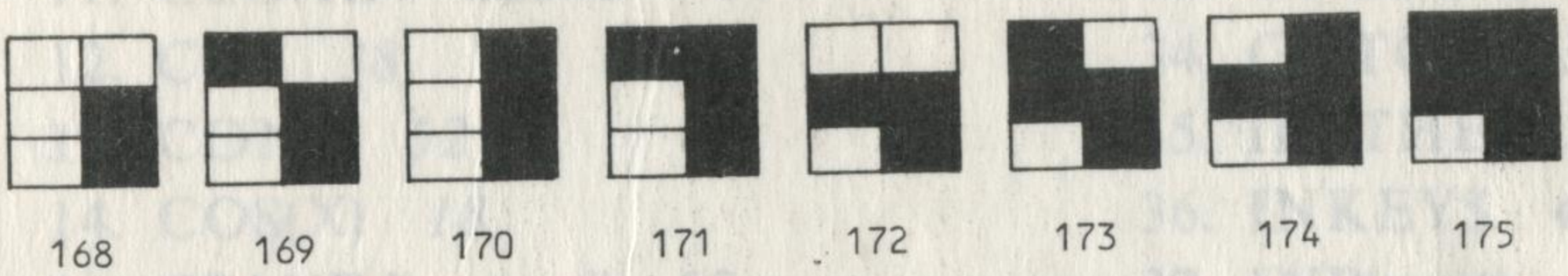


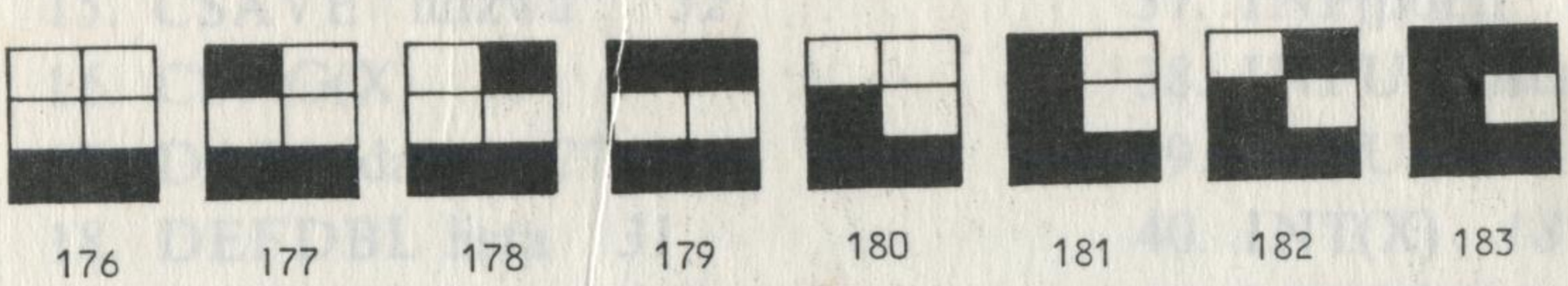


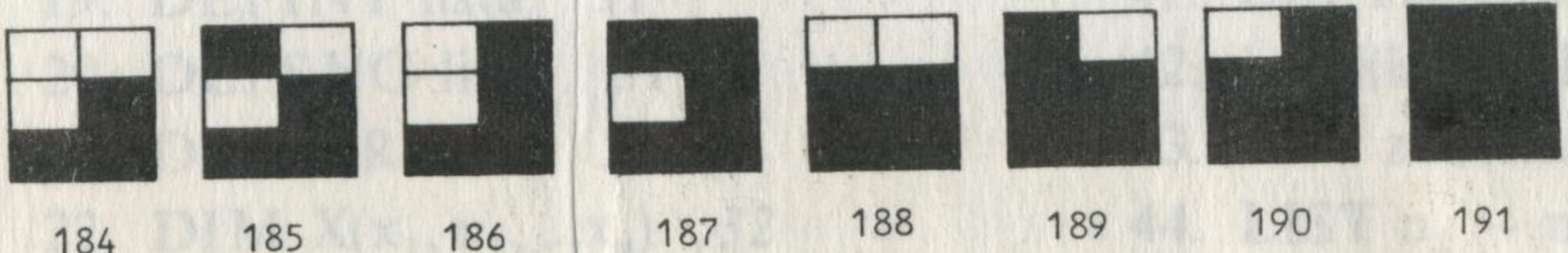












Dodatek 3

Wydruk zawartości generatora znaków

```
D:0,37F
0000  00 00 00 00 00 00 00 00 F0 F0 F0 F0 00 00 00 00
0010  0F 0F 0F 0F 00 00 00 00 FF FF FF FF 00 00 00 00
0020  00 00 00 00 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0
0030  0F 0F 0F 0F F0 F0 F0 F0 FF FF FF FF F0 F0 F0 F0
0040  00 20 40 FF 40 20 00 00 00 04 02 FF 02 04 00 00
0050  10 10 10 10 10 54 28 10 10 38 54 10 10 10 10 10
0060  FF FE FC F8 F0 E0 C0 80 01 03 07 0F 1F 3F 7F FF
0070  80 C0 E0 F0 F8 FC FE FF FF 7F 3F 1F 0F 07 03 01
0080  AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 00 00 00 00
0090  00 00 00 00 AA 55 AA 55 A0 50 A0 50 A0 50 A0 50
00A0  0A 05 0A 05 0A 05 0A 05 AA AA AA FF 00 00 00 00
00B0  00 00 00 FF AA AA AA AA AA AA AA AA AA AA AA AA
00C0  43 25 7A 83 FF 24 18 00 FE F9 24 FC FF 00 00 00
00D0  00 00 FC 02 FF 48 30 00 00 01 07 18 20 7F 05 09
00E0  00 80 E0 18 04 FE 80 40 00 00 00 18 18 00 00 00
00F0  00 00 3C 24 24 3C 00 00 00 7E 42 42 42 42 7E 00
0100  00 00 00 00 00 00 00 00 00 08 08 08 08 08 00 08
0110  00 14 14 14 00 00 00 00 00 14 14 3E 14 3E 14 14
0120  00 08 1E 28 1C 0A 3C 08 00 30 32 04 08 10 26 06
0130  00 10 28 28 10 2A 24 1A 00 08 08 08 00 00 00 00
0140  00 08 10 20 20 20 10 08 00 08 04 02 02 02 04 08
0150  00 08 2A 1C 08 1C 2A 08 00 00 08 08 3E 08 08 00
0160  00 00 00 00 00 08 08 10 00 00 00 00 3E 00 00 00
0170  00 00 00 00 00 00 00 08 00 00 02 04 08 10 20 00
0180  00 1C 22 26 2A 32 22 1C 00 08 18 08 08 08 08 1C
0190  00 1C 22 02 1C 20 20 3E 00 3E 02 04 0C 02 22 1C
01A0  00 04 0C 14 24 3E 04 04 00 3E 20 3C 02 02 22 1C
01B0  00 0E 10 20 3C 22 22 1C 00 3E 02 02 04 08 10 20
01C0  00 1C 22 22 1C 22 22 1C 00 1C 22 22 1E 02 04 38
01D0  00 00 00 08 00 08 00 00 00 00 00 08 00 08 08 10
01E0  00 04 08 10 20 10 08 04 00 00 00 3E 00 3E 00 00
01F0  00 10 08 04 02 04 08 10 00 1C 22 02 0C 08 00 08
0200  00 1C 22 2A 2E 2C 20 1E 00 08 14 22 22 3E 22 22
0210  00 3C 22 22 3C 22 22 3C 00 1C 22 20 20 20 22 1C
0220  00 3C 22 22 22 22 22 3C 00 3E 20 20 3C 20 20 3E
0230  00 3E 20 20 3C 20 20 20 00 1C 22 20 20 26 22 1E
0240  00 22 22 22 3E 22 22 22 00 1C 08 08 08 08 08 1C
0250  00 02 02 02 02 02 22 1C 00 22 24 28 30 28 24 22
0260  00 20 20 20 20 20 20 3E 00 22 36 2A 2A 2A 22 22
0270  00 22 22 32 2A 26 22 22 00 1C 22 22 22 22 22 1C
0280  00 3C 22 22 3C 20 20 20 00 1C 22 22 22 2A 24 1A
0290  00 3C 22 22 3C 28 24 22 00 1C 22 20 1C 02 22 1C
02A0  00 3E 2A 08 08 08 08 08 00 22 22 22 22 22 22 1C
02B0  00 22 22 22 14 14 08 08 00 22 22 22 2A 2A 2A 14
02C0  00 22 22 14 08 14 22 22 00 22 22 14 08 08 08 08
02D0  00 3E 02 04 08 10 20 3E 00 3E 30 30 30 30 30 3E
02E0  00 00 20 10 08 04 02 00 00 3E 06 06 06 06 06 3E
02F0  00 08 1C 3E 08 08 08 08 00 00 00 00 00 00 00 3E
0300  00 10 08 04 00 00 00 00 00 00 00 1C 02 1E 22 1E
0310  00 20 20 3C 22 22 22 3C 00 00 00 1E 20 20 20 1E
0320  00 02 02 1E 22 22 22 1E 00 00 00 1C 22 3E 20 1C
0330  00 04 08 08 1C 08 08 08 00 1E 22 22 22 1E 02 0C
0340  00 20 20 3C 22 22 22 22 00 08 00 18 08 08 08 1C
0350  00 08 00 08 08 08 28 10 00 10 10 12 14 18 14 12
0360  00 18 08 08 08 08 08 1C 00 00 00 34 2A 2A 2A 2A
0370  00 00 00 3C 22 22 22 22 00 00 00 1C 22 22 22 1C

[ASM]:
```

```
D:380,5FF

0380  00 3C 22 22 22 3C 20 20 00 1E 22 22 22 1E 02 02
0390  00 00 00 16 18 10 10 10 00 00 00 1E 20 1C 02 3C
03A0  00 00 08 1C 08 08 08 04 00 00 00 22 22 22 22 1E
03B0  00 00 00 22 22 14 14 08 00 00 00 22 22 2A 2A 14
03C0  00 00 00 22 14 08 14 22 00 22 22 22 22 1E 02 0C
03D0  00 00 00 3E 04 08 10 3E 00 04 08 08 10 08 08 04
03E0  00 08 08 08 08 08 08 08 00 10 08 08 04 08 08 10
03F0  00 00 00 40 00 40 00 00 55 AA 55 AA 55 AA 55 AA
0400  00 00 00 00 00 00 00 00 F0 F0 F0 00 00 00 00 00
0410  0F 0F 0F 00 00 00 00 00 FF FF FF 00 00 00 00 00
0420  00 00 00 F0 F0 00 00 00 F0 F0 F0 F0 F0 00 00 00
0430  0F 0F 0F F0 F0 00 00 00 FF FF FF F0 F0 00 00 00
0440  00 00 00 0F 0F 00 00 00 F0 F0 F0 0F 0F 00 00 00
0450  0F 0F 0F 0F 0F 00 00 00 FF FF FF 0F 0F 00 00 00
0460  00 00 00 FF FF 00 00 00 F0 F0 F0 FF FF 00 00 00
0470  0F 0F 0F FF FF 00 00 00 FF FF FF FF FF 00 00 00
0480  00 00 00 00 00 F0 F0 F0 F0 F0 F0 00 00 F0 F0 F0
0490  0F 0F 0F 00 00 F0 F0 F0 FF FF FF 00 00 F0 F0 F0
04A0  00 00 00 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0
04B0  0F 0F 0F F0 F0 F0 F0 F0 FF FF FF F0 F0 F0 F0 F0
04C0  00 00 00 0F 0F F0 F0 F0 F0 F0 F0 0F 0F F0 F0 F0
04D0  0F 0F 0F 0F 0F F0 F0 F0 FF FF FF 0F 0F F0 F0 F0
04E0  00 00 00 FF FF F0 F0 F0 F0 F0 F0 FF FF F0 F0 F0
04F0  0F 0F 0F FF FF F0 F0 F0 FF FF FF FF FF F0 F0 F0
0500  00 00 00 00 00 0F 0F 0F F0 F0 F0 00 00 0F 0F 0F
0510  0F 0F 0F 00 00 0F 0F 0F FF FF FF 00 00 0F 0F 0F
0520  00 00 00 F0 F0 0F 0F 0F F0 F0 F0 F0 F0 0F 0F 0F
0530  0F 0F 0F F0 F0 0F 0F 0F FF FF FF F0 F0 0F 0F 0F
0540  00 00 00 0F 0F 0F 0F 0F F0 F0 F0 0F 0F 0F 0F 0F
0550  0F 0F 0F 0F 0F 0F 0F 0F FF FF FF 0F 0F 0F 0F 0F
0560  00 00 00 FF FF 0F 0F 0F F0 F0 F0 FF FF 0F 0F 0F
0570  0F 0F 0F FF FF 0F 0F 0F FF FF FF FF FF 0F 0F 0F
0580  00 00 00 00 00 FF FF FF F0 F0 F0 00 00 FF FF FF
0590  0F 0F 0F 00 00 FF FF FF FF FF FF 00 00 FF FF FF
05A0  00 00 00 F0 F0 FF FF FF F0 F0 F0 F0 F0 FF FF FF
05B0  0F 0F 0F F0 F0 FF FF FF FF FF FF F0 F0 FF FF FF
05C0  00 00 00 0F 0F FF FF FF F0 F0 F0 0F 0F FF FF FF
05D0  0F 0F 0F 0F 0F FF FF FF FF FF FF 0F 0F FF FF FF
05E0  00 00 00 FF FF FF FF FF F0 F0 F0 FF FF FF FF FF
05F0  0F 0F 0F FF FF FF FF FF FF FF FF FF FF FF FF FF

[ASM]:
```

Literatura

1. Iszkowski W., Maniecki M.; *Programowanie w języku BASIC*, PWN, Warszawa 1980.
2. MERA-ELZAB, BASIC-MERITUM. Podręcznik programowania i użytkowania.
3. Kämpf K.: Das Colour-Genie ROM-Listing, 1984.
4. Die Mikrocomputer-Zeitschrift, Mai/Juni 1981.
5. Radio Shack, Lewel II Basic Reference Manual.

Cena zł 150,—

ISBN 83-204-0822-9