Mikroračunalniški krožek
LJUDSKA TEHNIKA
Cinkarna Celje

COMMODORE 64
BASIC

Alenka Stepančič
Ivan Kolenko

1. OPERACIJSKI SISTEM IN MNOŽICA ZNAKOV V BASICU NA CBM 64

Operacijski sistem (Operating System) vsebuje ROM (Read Only Memory) čip, in sicer je to kombinacija treh ločenih, vendar med seboj povezanih programskih modulov:

1) BASIC INTERPRETER (Predstavitev BASICA)
2) KERNAL (Jedro)
3) SCREEN EDITOR (Urejevalec izpisa na ekranu)

1) BASIC Interpreter je odgovoren za analizo stavčne sintakse v BASICU ter za predstavitev zahtevanih izračunov ali ureditev podatkov. BASIC interpreter pozna slovar 65 "osnovnih besed" (keywords) ali instrukcij, ki imajo natanko določen pomen. Na razpolago so nam male in velike črke angleške abecede ter številke od 0 - 9 za uporabo pri instrukcijah v BASICU ter imenih spremenljivk. Tudi nekatera ločila in posebni simboli imajo določene pomene v BASIC interpreterju. (Tabela 1)
2) KERNAL upravlja prekinjanje nivoja procesiranja v sistemu. Omogoča nam tudi takojšen vnos (INPUT) ter iznos (OUTPUT) podatkov.
3) SCREEN EDITOR kontrolira izpis na TV ekranu ali monitorju ter ureja programski tekst za listanje v BASICU. Sprejema tudi vhodne signale preko tastature ter določi ali naj jih takoj upošteva ali pa jih pošlje BASIC Interpreterju.

Operacijski sistem nam omogoča dva načina uporabe BASICA:

1) DIREKTEN ALI NEPOSREDEN NAČIN
2) PROGRAMSKI ALI POSREDEN NAČIN

1) Kadar uporabljamo direkten način, BASIC stavki nimajo oštevilčenih vrstic. Izvršijo se takoj, ko pritisnemo na RETURN tipko.
2) Programski način uporabljamo, ko želimo napisati program. Pri tem načinu morajo imeti vsi stavki oštevilčene vrstice. Lahko pišemo več kot en stavek v eni vrstici, toda število stavkov je omejeno, ker lahko programska vrstica vsebuje

TABELA 1

ZNAK	IME IN OPIS
	BLANK (prazno polje) - loči instrukcije ter imena spremenljivk
;	SEMI - COLON (podpičje) - uporablja se za določevanje formata pri izpisu spremenljivk
=	EQUAL SIGN (je enako) - določanje vrednosti ter testiranje relacije
+	PLUS SIGN - aritmetično seštevanje ali koncentriranje stringov
-	MINUS SIGN - aritmetično odštevanje, določanje predznaka (-1)
*	ASTERISK (zvezdica) - aritmetično množenje
/	SLASH (prečna črta) - aritmetično deljenje
↑	UP ARROW (puščica navzgor) - aritmetično potenciranje
(	LEFT PARENTHESIS (levi oklepaj) - določitev vrednosti izraza ali funkcije
)	RIGHT PARENTHESIS (desni oklepaj) - določitev vrednosti izraza ali funkcije
%	PERCENT (odstotek) - določa ime integer spremenljivke
#	NUMBER (število) - pišemo ga pred logičnim imenom datoteke v INPUT/OUTPUT stavkih
$	DOLLAR SIGN (znak za dolar) - določa ime string spremenljivke
,	COMMA (vejica) - uporablja se za določanje formata pri listanju spremenljivk, ločuje tudi parametre za ukazi
.	PERIOD (pika) - decimalna vejica pri realnih spremenljivkah
"	QUOTATNON MARK (narekovaj) - vsebuje string konstante
:	COLON (dvopičje) - ločuje več BASIC stavkov v vrstici
?	QUESTION MARK (vprašaj) - okrajšava za instrukcijo PRINT
<	LESS THEN (manj kot) - uporablja se pri testiranju relacij

ZNAK	IME IN OPIS
>	GREATER THEN (več kot) - uporablja se pri testiranju relacij
π	PI - numerična konstanta 3,141592654

samo 80 znakov (logična ekranska vrstica).

POZOR! Preden začnemo pisati program v BASICU, moramo vedno najprej vtipkati instrukcijo NEW ter pritisniti RETURN tipko.

CBM 64 ima dve množici znakov, ki jih lahko uporabljamo preko tastature ali pa jih izpisujemo s programom.

MNOŽICA 1 vsebuje velike črke abecede in številke od 0 - 9 (izpisujemo jih brez SHIFT tipke). Če pa pritisnemo in držimo SHIFT tipko, lahko uporabimo grafične znake na desni strani tipk. Če pritisnemo in držimo C= tipko, se izpisujejo grafični znaki na levi strani tipk. Kadar držimo SHIFT tipko in pritisnemo na tipke, ki nimajo znakov spredaj, se v večini primerov izpiše znak na vrhu tipke.

V MNOŽICI 2 so na voljo nale črke abecede in števila od 0 - 9 brez pritiska na SHIFT tipko. Velike črke pa lahko izpisujemo s pritiskom na SHIFT tipko. Grafične simbole na desni strani tipke lahko uporabljamo s pritiskom na C= tipko, medtem ko se simboli na vrhu tipk izpišejo, če pritisnemo na SHIFT tipko.

Iz prve množice v drugo in obratno preidemo tako, da pritisnemo hkrati C= in SHIFT tipko.

2. ŠTEVILA IN SPREMENLJIVKE

Konstante so podatki, ki jih uporabljamo v BASIC stavkih. Te vrednosti upošteva BASIC med izvajanjem programskega stavka. CBM BASIC lahko raspozna in uporablja tri tipe konstant:

1) INTEGER števila (celoštevilske konstante)
2) FLOATING-POINT števila (realne konstante)
3) STRINGE (konstante niza znakov)

INTEGER so cela števila, pišemo jih brez decimalne vejice, vendar morajo biti v mejah med - 32768 in + 32767. Znak + lahko pred števili izpustimo. Ničel, ki jih pišemo pred števili, BASIC ne upošteva. Integer konstante se shranijo v spominu računalnika kot dvo byte-na binarna števila.

Primeri: -122
8765
+44
0
-32767

POZOR! Med številkami nikoli ne piši vejic (?SYNTAX ERROR)

FLOATING-POINT konstante (konstante z gibajočo se piko) so realna števila. Uporabljamo in zapisujemo jih na dva načina:

a) SIMPLE (navaden zapis)
b) SCIENTIFIC NOTATION (znanstveni zapis)

Floating-point konstante izpiše BASIC na devet mest natančno. Ta mesta so vrednosti med - 999999999 do + 999999999. Če vnesemo več kot deset mest, bo število zaokroženo glede na vrednost na desetem mestu. Če je ta vrednost več ali enaka 5, bo zaokroženo navzgor, če pa je manjše od 5, bo zaokrožen navzdol.
Floating -point števila zavzemajo 5 bytov spomina in operirajo z desetimi mesti v izračunu, medtem ko se izpišejo na devet mest natančno.

Primeri: 1.23
-0.998877
3.1459
.777777
-333.
.01

Števila, ki so manjša od 0.01 ali večja od 999999999, izpiše BASIC v znanstvenem zapisu. V tem zapisu je konstanta sestavljena iz treh delov:

1) MANTISSA
2) ČRKA E
3) EKSPONENT

Mantissa je navadno realno število. Črka E nam pove, da je število zapisano v eksponentni obliki z osnovo 10. Mantisa in eksponent imata lahko predznak (+ ali -). Eksponent je omejen med - 39 do + 38 in nam pove, za koliko mest moramo premakniti decimalno vejico v številu mantise. Največje število, ki ga v tej obliki lahko napišemo v BASICU je + 1.70141183E+38. Če zapišemo ali izračunamo večje število, se pojavi ?OVERFLOW ERROR. Najmanjše število pa je + 2.93873588E-39. Pri manjših izračunih je rezultat nič. V tem primeru ni sporočila o napaki.

Primeri:	235.988E-3	(.235988)
	2359E 6	(2359000000)
	-7.09E-12	(-.00000000000709)

STRINGI so nizi alfanumeričnih izrazov (črke, števila in simboli). Ko vnašamo stringe preko tastature, so lahko dolgi toliko znakov, kolikor je prostora v vrstici (80 znakov max.). String lahko vsebuje prazne prostore, črke, števila, ločila ter kontrolne znake v kakršnikoli kombinaciji. Med števila lahko postavljamo tudi vejice. Edini znak, ki ga ne moremo vključiti v string je dvojni narekovaj ("). To pa zato, ker ga uporabljamo, da nam označi začetek in konec stringa. String ima lahko tudi nično vrednost - temu rečemo prazen string (ne vsebuje nobenega znaka). Zadnji narekovaj lahko izpustimo, če je string na koncu vrstice, ali če je za znakom dvopičje (:).

Primeri:
"" (prazen string)
"HELLO"
"$ 25,000.00"
"ŠTEVILO ZAPOSLENIH"

POZOR!	Če hočeš vključiti v string narekovaje, uporabi CHR$ (34).

3. SPREMENLJIVKE

Spremenljivke so količine, ki jim lahko priredimo različne vrednosti. Med seboj jih razlikujemo po njihovih imenih. Vrednost spremenljivki priredimo tako, da postane enaka konstanti ali pa je rezultat izračunov v programu. Vrednosti spremenljivk so lahko integer, floating-point števila ali pa stringi. Če uporabimo spremenljivko v programu, preden ji dodelimo določeno vrednost, moramo vedeti, da ji BASIC interpreter dodeli vrednost 0, če je spremenljivka integer ali floating-point. Če je to string spremenljivke pa ji priredi prazen string.

Imena spremenljivk si lahko poljubno izbiramo, vendar pa CBM BASIC razbere le prva dva znaka v imenu. To pomeni, da različne spremenljivke ne smejo imeti enaka prva dva znaka. Imena spremenljivk tudi ne smejo biti enaka instrukcijam v BASICU, prav tako te instrukcije ne smejo biti vključene v njihovo ime. Če se to zgodi, nam BASIC javi ?SINTAX ERROR.

Znaki, s katerimi tvorimo imena spremenljivk, so črke abecede ter števila od 0 - 9. Prvi znak v imenu mora biti črka. Znake, ki določajo vrsto spremenljivke (% in $) pa moramo pisati na koncu imena. Procent (%) določa, da je spremenljivka integer, medtem ko dolarski znak ($) pomeni string spremenljivke. Če ne uporabimo nobenega od teh dveh znakov, bo spremenljivka floating-point.

Primeri BASIC spremenljivk:

```
A$ = "VELIKA ABECEDA"          (string spremenljivke)
MTH$ = "MALA" + A$             (string spremenljivke)
K% = 5                         (integer spremenljivka)
CNT % = CNT% + 1               (integer spremenljivka)
FP = 12.5                      (floating-point spr.)
SUM = FP * CNT%                (floating-point spr.)
```

4. POLJA

Polje(array) je tabela (lista, matrika) podatkov, ki jih določa samo eno ime spremenljivke. Polje je niz med seboj povezanih spremenljivk. Primer takega polja je tabela, katere posamezna števila so elementi polja.

Polje uporabljamo takrat, kadar imamo veliko število med seboj povezanih podatkov (npr. zaporedne meritve). Posameznemu polju lahko priredimo veliko število podatkov tako, da je vsak določen z indeksom polja. Polja imajo lahko eno samo dimenzijo, lahko pa so več dimenzionalna. Elementi polja pa so lahko integer, floating-point ali string spremenljivke. Teoretično je število dimenzij omejeno z 255, medtem ko je število elementov v vsaki dimenziji omejeno s 32767. Praktično pa smo pri dimenzioniranju omejeni s spominom računalnika ter z 80 znaki v logični vrstici. Če ima polje eno dimenzijo ter število elementov ne presega 10 (11 indeksov, 0 - 10), potem bo polje BASIC interpreter avtomatično sprejel, če pa so dimenzije ali število elementov večje, je potrebno polje najprej določiti z inštrukcijo DIM, ki nam definira obliko in velikost polja (DIM (X,Y,Z)).
Spomin, ki ga zahteva tako definirano polje:

```
     5 bytov za ime polja
   + 2 byta za vsako dimenzijo
   + 2 byta za vsak integer element
ali+ 5 bytov za vsak floating-point element
ali+ 3 byte za vsak string element
in + 1 byte za vsak karakter znak v vsakem stringu
```

Števila predpisana dimenzijam morajo biti integer konstante ali spremenljivke ali pa tudi aritmetični izrazi, katerih rezultati so integer števila. Ta števila ločimo z vejico za vsako posamezno dimenzijo. Če določimo polju elemente, ki jih nismo dimenzionirali, nam javi BASIC ?BAD SUBSCRIPT napako.

Nekaj primerov razlag polj:

```
A$(0) = "VELIKI SVET"              (polje stringov)
MTH$(K%) = "JAN"                   (polje stringov)
G2%(X) = 5                         (integer polje)
CNT%(G2%(X)) = CNT%(1) + 2         (integer polje)
FP(12*K%) = 24.8                   (floating-point polje)
SUM(CNT%(1)) = FP K%               (floating-point polje)
```

A(5) = 0 (petemu elementu enodimenzionalnega polja A priredi vrednost 0)

B(5,6) = 0 (elementu v 5. vrstici in 6. stolpcu matrike (polja) B priredi vrednost 0)

C(1,2,3) = 0 (elementu s koordinatami X,Y,Z = 1,2,3 trodimenzionalnega polja C priredi vrednost 0).

5. IZRAZI IN OPERATORJI

Izraze v BASICU formiramo s konstantami, spremenljivkami ali polji. Izraz je lahko samo ena konstanta, preprosta spremenljivka ali pa polje spremenljivk, katerekoli vrste. Lahko pa je tudi kombinacija konstant in spremenljivk, povezanih z aritmetičnimi, relacijskimi ali logičnimi operatorji, tako da predstavlja eno samo vrednost. Izraze lahko razdelimo v dva razreda:

1) ARITMETIČNI IZRAZI
2) STRING IZRAZI

Operatorji so specialni simboli, ki jih BASIC interpreter uporabi za reprezentacijo operacije, ki jo je potrebno izvesti na spremenljivkah ali konstantah. Eden ali več operatorjev, ki jih kombiniramo z eno ali več spremenljivkami ali konstantami predstavljajo izraz.

Aritmetični operatorji

Kadar rešimo aritmetični izraz, nam pove integer ali floating-point vrednost. Aritmetične operatorje uporabljamo za

seštevanje, odštevanje, množenje, deljenje in potenciranje.

Aritmetični operator definira aritmetično operacijo med dvema operandoma na vsaki strani operatorja. Aritmetični operatorji uporabljajo floating-point števila. Integer števila se spremenijo v floating-point, preden se izvrši operacija. Rezultat pa prevede spet nazaj v integer, če je prirejen integer spremenljivki.

SEŠTEVANJE (+):
```
2 + 2
A + B + C
X% + 1
BR + 10E-2
```

ODŠTEVANJE (-):
```
4 - 1
100 - 64
A - B
S5 - 142
```

Znak minus (-) se uporablja tudi kot minus enote. To pomeni, da moramo pred negativnimi števili pisati (-):

```
- 1
-9E4
-B
4-(-2)    enako kot 4 + 2
```

MNOŽENJE (*):
```
100*2
50*0
A*X1
R%*14
```

DELJENJE (/):
```
10/2
6400/4
A/B
4E2/XR
```

POTENCIRANJE (↑): število na desni strani znaka je eksponent

```
2↑2    je enako 2 * 2
3↑3    je enako 3 * 3 * 3
```

AB↑CD
3↑-2 je enako 1/3 * 1/3

Relacijski operatorji

Relacijske operatorje uporabljamo zato, da primerjamo vrednosti dveh operandov, prav tako pa predstavljajo določen aritmetični rezultat. Relacijski operatorji (< , = , > , > =, < =, <>) in logični operatorji (AND, OR, NOT) nam, kadar jih uporabljamo v primerjavi, podajo aritmetično pravilo/nepravilno vrednost izraza . Če je relacija v izrazu pravilna, je ta rezultat - 1, če pa je nepravilna, je rezultat 0.

<	je manjše kot
=	je enako
>	je večje kot
<=	je manjše kvečjemu enako
>=	je večje kvačjemu enako
<>	ni enako

Primeri:
1 = 5 - 4 (vrednost izraza je - 1 (pravilno))
14 > 66 (vrednost izraza je 0 (nepravilno))
15 >= 15 (vrednost izraza je - 1 (pravilno))

Relacijske operatorje lahko tudi uporabljamo za primerjanje stringov. Pri tej primerjavi predpostavimo, da ima abeceda črke urejene po velikosti (A<B<C< ...itd). Stringe primerjamo tako, da ugotovimo relacijo med posameznimi karakterji od leve proti desni.

Primeri:
"A" < "B" (vrednost izraza je -1 (pravilno))
"X" = "YY" (vrednost izraza je 0 (nepravilno))
BB$ <> CC$

Numerične podatke lahko primerjamo samo z drugimi numeričnimi podatki, prav tako to velja za primerjavo stringov, drugače se pojavi napaka ?TYPE MISMATCH.

Na koncu vseh primerjav vedno dobimo integer vrednost, ne glede na vrsto operandov (tudi, če so to stringi). Ta vrednost je lahko - 1 ali 0 ter jo lahko uporabljamo v aritmetičnih izračunih.

Logični operatorji

Logične operatorje (AND, OR in NOT) uporabljamo zato, da oblikujejo pomen relacijskih operatorjev ali pa nam podajo aritmetični rezultat. Logični operatorji lahko podajo rezultat, ki je različen od - 1 in 0, zato je vsak neničelen rezultat pravilen, kadar testiramo pogoj pravilno/nepravilno.

Tabela logičnih operatorjev:

1. Bit-rezultat operatorja AND je 1, samo če sta oba ustrezna bita 1:

   1 AND 1 = 1
   1 AND 0 = 0
   0 AND 1 = 0
   0 AND 0 = 0

2. Bit-rezultat operatorja OR je 1, če je vsaj eden od ustreznih bitov 1:

   1 OR 1 = 1
   0 OR 1 = 1
   1 OR 0 = 1
   0 OR 0 = 0

3. NOT je logični komplement vsakega bita:

   NOT 1 = 0
   NOT 0 = 1

4. Ekskluzivni OR (X OR) je lahko samo del WAIT stavka:

   1 XOR 1 = 0
   1 XOR 0 = 1
   0 XOR 1 = 1
   0 XOR 0 = 0

Logični operatorji se izvršijo šele potem, ko se izvedejo vse aritmetične in relacijske operacije.

Primeri:

```
IF A = 100 AND B = 200 THEN 10
```
(če imata spremenljivki A in B obe hkrati vrednost 100, potem se izvrši ukaz THEN, sicer pa ne)

```
A = 96 AND 32 : PRINT A  (A = 32)
IF A = 100 OR B = 100 THEN 20
```
(če je A ali B ali pa oba hkrati enaka 100, potem se izvrši ukaz THEN, sicer pa ne)

```
A = 64 OR 32 : PRINT A     (A = 96)
IF NOT  X< Y THEN 30
```
(če je X>= Y potem se izvrši ukaz THEN)

```
X = NOT 96 (rezultat je - 97 (komplement glede na
            število 2))
```

6. ZAPOREDJE IZVAJANJA OPERACIJ

Kadar uporabljamo več operacij skupaj, moramo biti pozorni na zaporedje izvajanja, kajti nekatere operacije imajo prednost pred drugimi. Če pa se hočemo zavarovati pred napakami ali pa poudariti prednost, moramo uporabljati oklepaje. V tem primeru se izvršijo najprej operacije v oklepajih. Oklepaje lahko uporabljamo druge v drugih, vendar največ desetkrat. Najprej se izvršijo operacije v notranjih oklepajih.

Primeri:

```
C↑(D+E)/2
((X - C↑(D + E)/2) * 10) + 1
K% = 2 OR (A = B AND M< X)
NOT (D = E)
```

BASIC interpreter bo normalno izvajal operacije tako, da bo najprej izračunal aritmetične izraze, potem relacijske operatorje in na koncu logične operatorje.

HIERARHIJA OPERATORJEV

	OPERATOR	OPIS	PRIMER
1.	↑	Potenciranje	BASE ↑ EXP
2.	-	Negacija (minus enote)	- A
3.	* /	Množenje in deljenje od leve proti desni	AB * CD EF * GH
4.	+ -	Seštevanje in odštevanje od leve proti desni	CNT + 2 JK - PQ
5.	> = <	Operatorji relacije od leve proti desni	A < = B
6.	NOT	Logični NOT (komplement števila 2)	NOT K%
7.	AND	Logični AND	JK AND 128
8.	OR	Logični OR	PQ OR 15

7. OPERACIJE MED STRINGI

Pri primerjanju stringov uporabljamo relacijske operatorje (=, <>, <=, >=, <, >), ki jih uporabljamo tudi za števila. Primerjava stringov se izvaja tako, da vzamemo po en karakter od leve proti desni od vsakega stringa in ugotovimo njegovo kodo v množici CBM karakterjev. Če sta ti kodi enaki, potem sta enaka tudi karakterja. Če pa se kodi razlikujeta, potem je karakter z manjšim kodnim številom manjši od karakterja z večjim kodnim številom. Primerjava se ustavi, ko je konec enega od stringov. Če sta stringa do takrat enaka, pa je krajši string manjši od daljšega. Pri primerjanju moramo upoštevati tudi prazne prostore v stringu. Stringe med seboj lahko tudi seštevamo, in sicer z znakom +. Kadar stringe koncentriramo, to storimo tako, da string na desni strani + znaka dodamo stringu na levi strani + znaka, določimo pa tretji string kot rezultat.

Primer:

```
10 A$ = "FILE" : B$ = "NAME"
20 NAM$ = A$ + B$              (Formira se string "FILENAME")
30 RES$ = "NEW "+ A$ + B$      (Formira se string "NEW FILENAME")
```

8. UKAZI IN INSTRUKCIJE CBM BASICA

V tabeli so zapisane vse instrukcije, ki jih uporablja CBM BASIC, kako jih lahko okrajšano zapišemo ter kakšen znak se v tem primeru izpiše na ekranu.

CBM BASIC dovoljuje okrajšavo skoraj vseh instrukcij. Ta okrajšava ne ohrani spomina, ker se vse instrukcije prevedejo z BASIC interpreterjem na enojne ASCII znake. Kadar program listamo, se te besede pojavijo v izpisani obliki. Okrajšavo lahko uporabljamo takrat, kadar želimo vstaviti več programskih stavkov v logično 80-znakovno vrstico, toda ker Screen Editor deluje samo z 80-znakovno vrstico, te vrstice pri listanju ne moremo popraviti, ampak je potrebno v tem primeru zapisati vrstico še enkrat, ali pa vrstico razdeliti v dve vrstici.

Pojasnili bomo tudi BASIC funkcije, ki so vgrajene v BASIC interpreter. Te lahko uporabljamo kot direktne funkcije v BASIC stavkih, ne da bi jih najprej definirali. Imamo dva tipa funkcij v BASICU:

1) numerične funkcije
2) string funkcije

Argumente teh funkcij moramo vedno pisati v oklepajih. Pri tem moramo paziti, da pišemo oklepaj takoj za funkcijskim ukazom, brez praznega prostora med zadnjo črko ukaza in levim oklepajem. Tip argumenta je v večini primerov določen s tipom rezultata.

TABELA CBM INSTRUKCIJ

	INSTRUKCIJA	OKRAJŠAN ZAPIS	EKRAN
1	ABS	A SHIFT B	A
2	AND	A SHIFT N	A
3	ASC	A SHIFT S	A
4	ATN	A SHIFT T	A
5	CHR$	C SHIFT H	C
6	CLOSE	CL SHIFT O	CL
7	CLR	C SHIFT L	C
8	CMD	C SHIFT M	C
9	CONT	C SHIFT O	C
10	COS	ni	COS
11	DATA	D SHIFT A	D
12	DEF	D SHIFT E	D
13	DIM	D SHIFT I	D
14	END	E SHIFT N	E
15	EXP	E SHIFT X	E
16	FN	ni	FN
17	FOR	F SHIFT O	F
18	FRE	F SHIFT R	F
19	GET	G SHIFT E	G
20	GET#	ni	GET#
21	GOSUB	GO SHIFT S	GO
22	GOTO	G SHIFT O	G
23	IF	ni	IF
24	INPUT	ni	INPUT
25	INPUT#	I SHIFT N	I
26	INT	ni	INT
27	LEFT$	LE SHIFT F	LE
28	LEN	ni	LEN
29	LET	L SHIFT E	L
30	LIST	L SHIFT I	L
31	LOAD	L SHIFT O	L

	INSTRUKCIJA	OKRAJŠAN ZAPIS	EKRAN
32	LOG	ni	LOG
33	MID$	M [SHIFT] I	M ◪
34	NEW	ni	NEW
35	NEXT	N [SHIFT] E	N ▭
36	NOT	N [SHIFT] O	N □
37	ON	ni	ON
38	OPEN	O [SHIFT] P	O □
39	OR	ni	OR
40	PEEK	P [SHIFT] E	P ▭
41	POKE	P [SHIFT] O	P □
42	POS	ni	POS
43	PRINT	?	?
44	PRINT#	P [SHIFT] R	P ▭
45	READ	R [SHIFT] E	R ▭
46	REM	ni	REM
47	RESTORE	RE [SHIFT] S	RE ▩
48	RETURN	RE [SHIFT] T	RE ▯
49	RIGHT$	R [SHIFT] I	R ◪
50	RND	R [SHIFT] N	R ◩
51	RUN	R [SHIFT] U	R ◪
52	SAVE	S [SHIFT] A	S ▩
53	SGN	S [SHIFT] G	S ▯
54	SIN	S [SHIFT] I	S ◪
55	SPC(	S [SHIFT] P	S □
56	SQR	S [SHIFT] Q	S ▩
57	STATUS	ST	ST
58	STEP	ST [SHIFT] E	ST ▭
59	STOP	S [SHIFT] T	S ▯
60	STR$	ST [SHIFT] R	ST ▭
61	SYS	S [SHIFT] Y	S ▯
62	TAB(	T [SHIFT] A	T ▩
63	TAN	ni	TAN
64	THEN	T [SHIFT] H	T ▯

	INSTRUKCIJA	OKRAJŠAN ZAPIS	EKRAN
65	TIME	TI	TI
66	TIME$	TI$	TI$
67	TO	ni	TO
68	USR	U SHIFT S	U ▒
69	VAL	V SHIFT A	V ▒
70	VERIFY	V SHIFT E	V ▒
71	WAIT	W SHIFT A	W ▒

OPIS POSAMEZNIH INSTRUKCIJ IN UKAZOV CBM BASICA

ABS

Tip: Funkcija - numerična
Format: ABS (<izraz>)

Delovanje: Izračuna absolutno vrednost izraza v oklepaju. Absolutna vrednost negativnega števila je število, ki je pomnoženo z - 1.

Primeri: 1Ø X = ABS(Y)
1Ø PRINT ABS (X*Y)
1Ø IF X = ABS (X) THEN PRINT "POZITIVNO"

AND

Tip: Operator
Format: <izraz> AND <izraz>

Delovanje: AND uporabljamo kot logični operator v testiranju vrednosti bitov. Prav tako se uporablja v operacijah, ki testirajo pravilnost dveh operandov.

CBM 64 nam omogoča operacijo AND med števili v območju - 32768 do + 32767. Uporaba decimalnih vrednosti in števil, ki niso v tem območju, povzroči ?ILLEGAL QUANTITY ERROR. Če spremenimo števila v dvojiški sistem, nam območje omogoča 16-bitno polje za vsako število. Ustrezajoči biti se seštejejo in formirajo rezultat v istem območju.

Primeri 16-bitnih AND operacij:

		17
		AND 194
		0000 0000 0001 0001
	AND	0000 0000 1100 0010
dvojiško		0000 0000 0000 0000
desetiško		0

```
                                        - 241
                              AND       15359
                         1111 1111 0000 1111
                    AND  0011 1011 1111 1111
dvojiško            ---------------------------
                         0011 1011 0000 1111
desetiško                          15119
```

Kadar testiramo vrednost kot pravilno/nepravilno, računalnik predpostavlja, da je število pravilno tako dolgo, dokler njegova vrednost ni enaka Ø.

Primeri: 5Ø IF X = 7 AND W = 3 THEN GOTO 1Ø: REM PRAVILNO SAMO, ČE STA OBA IZRAZA X = 7 IN W = 3 PRAVILNA
6Ø IF A AND Q = 7 THEN GOTO 1Ø: REM PRAVILNO; ČE A NI ENAK NIČ IN ČE JE Q = 7

ASC

Tip: Funkcija - numerična
Format: ASC(<string>)

Delovanje: ASC funkcija izpiše število med 0 in 255, ki odgovarja CBM ASCII kodi prvega znaka v stringu. Tabela Commodore ASCII znakov je na strani 65 .

Primeri:
```
1Ø PRINT ASC("Z")
2Ø X = ASC("ZEBRA")
3Ø J = ASC(J$)
```

Če nimamo znakov v stringu, javi ?ILLEGAL QUANTITY napako. Da se izognemo tej napaki, uporabimo CHR$(Ø), to pomeni prazen string.

```
3Ø J = ASC(J$ + CHR$(Ø))
```

ATN

Tip: Funkcija - numerična
Format: ATN(<število>)

Delovanje: Ta matematična funkcija nam izpiše arcus tangens števila v oklepaju. Rezultat je kot (v radianih),

katerega tangens je dano število. Rezultat je vedno v mejah - π/2 in +π/2.

Primeri:
```
1Ø PRINT ATN(Ø)
2Ø X = ATN(J) * 18Ø/π : REM PRETVORIMO V STOPINJE
```

CHR$

Tip: Funkcija - string
Format: CHR$ (<število>)

Delovanje: CHR$ funkcija pretvori CBM ASCII kodo v njen znakovni ekvivalent. (Tabela Commodore ASCII kode) Število v oklepaju mora biti med 0 in 255, drugače nam javi ?ILLEGAL QUANTITIY napako.

Primeri:
```
1Ø PRINT CHR$(65) : REM 65 = VELIKA ČRKA A
2Ø A$ = CHR$(13) : REM 13 = RETURN TIPKA
3Ø A = ASC(A$) : A$ = CHR$(A) : REM PRETVARJANJE V
   ASCII KODO IN NAZAJ
```

CLOSE

Tip: Input/Output stavek
Format: CLOSE <številka datoteke>

Delovanje: Ta stavek zapre katerokoli podatkovno datoteko ali dohod na eksterni priključek. Številka datoteke je enaka tisti, s katero smo odpirali datoteko (OPEN stavek).
Kadar delamo s kaseto ali disketo, instrukcija CLOSE poskrbi, da priključek sprejme vse dopolnilne vmesnike; če tega ne storimo, se lahko zgodi, da datoteka ni popolna in je ne moremo prebrati. Operacija CLOSE ni tako potrebna pri drugih priključkih, vendar nam osvobodi spomin za dodatne datoteke.

Primeri:
```
1Ø CLOSE 1
2Ø CLOSE X
3Ø CLOSE 9 * (1 + J)
```

CLR

Tip: Stavek
Format:CLR

Delovanje: Ta stavek nam osvobodi RAM spomin, ki smo ga uporabili, vendar ga ne potrebujemo več. Katerikoli BASIC program v spominu ostane nedotaknjen, vendar pa so vse spremenljivke, polja, GOSUB adrese, FOR ... NEXT zanke zbrisane iz spomina. V primeru, ko imamo odprte datoteke na disketi ali kaseti, jih CLR stavek ne zaključi pravilno. Informacija o datotekah je za računalnik zgubljena, medtem ko disk drive še vedno upošteva, da je datoteka odprta.

Primeri:
```
1Ø X = 25
2Ø CLR
3Ø PRINT X

RUN
Ø
```

CMD

Tip: Input/Output stavek
Format: CMD <številka datoteke> [,string]

Delovanje: Ta stavek nam preklopi primarni output iz ekrana na specifično datoteko. Ta datoteka je lahko na disku, traku, printerju ali na I/O priključke, ki jih imenujemo modeme. Številko datoteke moramo določiti v predhodnem OPEN stavku. String, ki ga določimo, je poslan na datoteko. Kadar deluje CMD stavek, se PRINT in LIST stavki ne izpisujejo na ekranu, ampak pošljejo tekst z istim formatom na datoteko.

Za preusmeritev outputa nazaj na ekran, nam bo PRINT# stavek poslal prazno vrstico na priključek, preden ga zapremo, tako da lahko zaključimo poslane podatke.

Vsaka sistemska napaka (npr. ?SINTAX ERROR) bo povzročila, da se output vrne na ekran.

Primeri:
```
OPEN 4,4: CMD 4, "NASLOV": LIST: REM LISTAJMO PROGRAM
NA PRINTER
PRINT# 4: CLOSE 4: REM ZAKLJUČIMO PISANJE NA PRINTER

1Ø OPEN 1,1,1 "TEST" : REM KREIRAMO SEKVENČNO DATOTEKO
2Ø CMD 1: REM OUTPUT NA TRAK, NE NA EKRAN
3Ø FOR L = 1 TO 1ØØ
4Ø PRINT L: REM NAPIŠE ŠTEVILO NA TRAKU
5Ø NEXT
6Ø PRINT# 1: REM NE PIŠE VEČ NA TRAK
7Ø CLOSE 1: REM PRAVILNO ZAKLJUČI DATOTEKO
```

CONT

Tip: Ukaz

Format: CONT

Delovanje: Ta ukaz začne ponovno izvajati program, ki smo ga zaustavili s STOP ali END stavkom, ali pa če smo pritisnili na RUN/STOP tipko. Program se bo nadaljeval točno na mestu, kjer smo ga prekinili.
Medtem ko je program zaustavljen, lahko pregledamo ali zamenjamo katerokoli spremenljivko ali pa pregledamo program. Pri čiščenju ali pregledu programa postavimo STOP stavek na tako točko v programu, ki nam omogoča pregled spremenljivk in potek programa.

CAN'T CONTINUE napaka se pojavi, kadar se program zaustavi zaradi napake, ali če smo zapisali nepravilno instrukcijo, preden smo s CONT spet hoteli nadaljevati program.

Primeri:
```
1Ø PI = Ø : C = 1
2Ø PI = PI + 4/C + 4/(C + 2)
3Ø PRINT PI
4Ø C = C + 4: GOTO 2Ø
```

Program računa vrednost transcendentnega števila π .
Z RUN program začnemo in čez nekaj časa, ko pritisnemo na RUN/STOP tipko, se izpiše na ekran:

BREAK IN 2Ø (lahko tudi katera druga vrstica,

Sedaj lahko zapišemo PRINT C, da ugotovimo, na kateri koraku se je program zaustavil. S CONT pa lahko nadaljujemo program.

COS

. Funkcija

Format: COS (<število>)

Delovanje: Ta matematična funkcija izračuna cosinus števila v oklepajih; to število je kot v radianih.

Primeri: 1Ø PRINT COS(Ø)
2Ø X = COS (Y*π/18Ø): REM PRETVORI STOPINJE V RADIANE

DATA

Tip: Stavek

Format: DATA <zaporedje konstant>

Delovanje: DATA stavek uporabljamo za shranjevanje informacij v programu samem. Program te informacije uporabi z READ stavkom, ki prebere potrebne konstante iz DATA stavkov.

DATA stavkov program ne izvaja, morajo pa biti napisani, če uporabimo READ stavek. DATA stavke lahko pišemo kjerkoli v programu, najpogosteje pa jih zaradi preglednosti postavimo na konec programa. Vsi DATA stavki so pravzaprav neko zaporedje konstant, ki jih ukaz READ bere od leve proti desni, od programske vrste z najnižjo številko do programske vrste z najvišjo številko. Če READ stavek naleti na podatke, ki ne ustrezajo zahtevanemu tipu (številu, stringu), se pojavi sporočilo o napaki. Kot podatek lahko v DATA stavek vključimo katerikoli znak, vendar pa moramo nekatere pisati v narekovajih. To je vejica (","), dvopičje (":"), prazni prostori (" ") ter nekateri grafični kontrolni znaki.

Primeri: 1Ø DATA 1,1Ø,5,8
2Ø DATA JOHN, PAUL, GEORGE, RINGO
3Ø DATA "V DATA STAVKE LAHKO SHRANIMO PODATKE"
4Ø DATA - 1.7E-9, 3.333

DEF FN

Tip: Stavek

Format: DEF FN <ime> (<spremenljivka>) = <izraz>

Delovanje: Ta stavek uporabljamo za definicijo funkcije, ki jo želimo uporabiti večkrat v programu. Funkcijo lahko sestavimo s katerokoli matematično formulo. To formulo določimo samo enkrat v definicijskem stavku, potem pa jo uporabljamo z imenom funkcije. Funkcijsko ime je FN, ki mu sledi ime katerekoli spremenljivke, le-to je lahko dolgo en ali pa dva znaka, prvi znak je vedno črka, drugi pa je lahko črka ali številka.

Primeri definicije funkcij:

```
1Ø DEF FN A(X) = X + 7
2Ø DEF FN AA(X) = Y * Z
3Ø DEF FN A9(X) = INT (RND(1)* Q + 1)
```

Funkcijo pokličemo kasneje v programu tako, da zapišemo njeno ime s spremenljivko v oklepaju. To funkcijsko ime se uporablja kot katerakoli druga spremenljivka in njena vrednost se avtomatično izračuna.

Primeri uporabe definiranih funkcij:

```
4Ø PRINT FN A(9)
5Ø R = FN AA(9)
6Ø G = 6 + FN A9(1Ø)
```

V vrstici 5Ø nam število 9 v oklepajih ne določa vrednosti funkcije, ker definicija v vrstici 2Ø ne določi spremenljivke, ki je v oklepaju. Rezultat je vedno Y * Z, brez ozira na X, ki je v oklepaju. V ostalih dveh definicijah pa vrednost v oklepaju vpliva na izračun.

DIM

Tip: Stavek

Format: DIM <spremenljivka>(<številka>)[,<spremenljivka>(<številka>)...]

Delovanje: DIM stavek definira polje (matriko) spremenljivk. Omogoča nam, da uporabljamo spremenljivke z indeksi. Indeks nam natanko določi element polja. Najnižja številka indeksa je Ø, medtem ko je najvišja dana z DIM stavkom, ta pa jih ima lahko največ 32767. DIM stavek lahko uporabljamo v programu samo enkrat za isto polje, če hočemo polje dimenzionirati še drugič, se pojavi ?REDIM'D ARRAY napaka. Najboljše je, če predpostavimo in definiramo vsa polja že na začetku programa. Polja so lahko sestavljena iz realnih števil, lahko pa vsebujejo stringe ali integer števila, v tem primeru moramo pisati ustrezni znak za imenom polja ($ ali %). Če polja, ki ga uporabljamo v programu nismo dimenzionirali, mu lahko določimo samo 11 elementov.

Primeri:

```
1Ø DIM A(1ØØ)
2Ø DIM Z(5,7), Y(3,4,5)
3Ø DIM Y7% (Q)
4Ø DIM PH$ (1ØØØ)
5Ø F(4) = 9: REM AVTOMATIČNO PREDPOSTAVI DIM F(1Ø)
```

Primer programa - menjava rezultata na tekmi:

```
1Ø DIM S(1,5), T$(1)
2Ø INPUT "IMENI MOSTEV"; T$(Ø), T$(1)
3Ø FOR Q = 1 TO 5: FOR T = Ø TO 1
4Ø PRINT T$(T), "REZULTAT V PETINI "Q
5Ø INPUT S(T,Q): S(T,Ø) = S(T,Ø) + S(T,Q)
6Ø NEXT T,Q
7Ø PRINT CHR$(147) "REZULTATI"
8Ø PRINT "PETINA"
9Ø FOR Q = 1 TO 5
```

```
1ØØ PRINT TAB(Q * 2 + 9) Q;
11Ø NEXT  : PRINT TAB(15) "TOTAL"
12Ø FOR T = Ø TO 1: PRINT T$(T)
13Ø FOR Q = 1 TO 5
14Ø PRINT TAB(Q * 2 + 9) S(T,Q)
15Ø NEXT: PRINT TAB (15) S(T,Ø)
16Ø NEXT
```

END

Tip: Stavek

Format: END

Delovanje: END stavek konča program, po njem se izpiše na ekranu READY. V program lahko postavimo več END stavkov, ni pa nujno, da vedno postavimo na konec programa END stavek, ker se program izteče z zadnjo programsko vrstico, vendar pa je to priporočljivo. END stavek je podoben STOP stavku, edina razlika je, da STOP stavek izpiše na ekran BREAK IN XX (številka vrstice), medtem ko END izpiše samo READY. Oba stavka pa omogočata nadaljevanje s CONT ukazom.

Primeri:

```
1Ø PRINT "ALI ZELIS KONCATI PROGRAM?"
2Ø INPUT A$
3Ø IF A$ = "DA" THEN END
4Ø REM OSTALI
   PROGRAM
   :
999 END
```

EXP

Tip: Funkcija - numerična

Format: EXP (<število>)

Delovanje: EXP je matematična funkcija, ki nam izračuna potenco števila e (2.71828182), pri tem pa je eksponent število v oklepaju. Vrednost v oklepaju, ki je večja od 88.Ø296919 povzroči ?OVERFLOW napako.

Primeri: 1Ø PRINT EXP(1)
2Ø X = Y ↑ EXP (Z ↑ Q)

FN

Tip: Funkcija - numerična
Format: FN <ime> (<število>)

Delovanje: Ta funkcija se nanaša na prej definirano formulo z DEF FN stavkom, ki določa tudi ime funkcije. Število v oklepaju se vstavi v formulo namesto spremenljivke, tako da se izračuna funkcijska vrednost. Rezultat je vedno numeričen. Če uporabimo FN pred DEF FN stavkom v programu, se pojavi UNDEF'D FUNCTION napaka.

Primeri: 1Ø PRINT FN A(Q)
112Ø J = FN J(7) + FN J(9)
999Ø IF FN B7 (I+1) = 6 THEN END

FOR...TO STEP

Tip: Stavek
Format: FOR <spremenljivka> = <začetna vrednost> TO <končna vrednost> [STEP <korak>]

Delovanje: FOR stavek nam v BASICU dovoljuje, da uporabimo spremenljivko kot števec. Najprej je potrebno določiti naslednje parametre: floating-point spremenljivko, njeno začetno vrednost, končno vrednost ter korak, s katerim šteje.
Primer kratkega BASIC programa, ki šteje od 1 do 1Ø, izpiše vsako število in konča program, ko se zanka zaključi, pri tem pa ne uporabimo FOR stavka:

```
1ØØ L = 1
11Ø PRINT L
12Ø L = L + 1
13Ø IF L<1Ø THEN 11Ø
14Ø END
```

Isti program lahko zapišemo z uporabo FOR zanke:

```
1ØØ FOR L = 1 TO 1Ø
11Ø PRINT L
12Ø NEXT L
13Ø END
```

Kot vidimo iz primera je z uporabo FOR zanke program krajši ter lažje razumljiv.
Kadar izvajamo FOR stavek, se izvrši nekaj operacij. Začetna vrednost postane vrednost spremenljivke števca, ko pridemo nato do NEXT stavka, se vrednost koraka doda vrednosti spremenljivke števca. Če STEP stavka ni, potem je ta korak vedno + 1. Sedaj se vrednost spremenljivke primerja s končno vrednostjo, in če le-ta še ni dosežena, se izvajanje zanke nadaljuje. V primeru, ko vrednost spremenljivke doseže končno vrednost, se zanka zaključi in program se nadaljuje za NEXT stavkom. Če je vrednost koraka pozitivna, mora spremenljivka doseči ali preseči končno vrednost, kadar pa je negativna, pa mora postati enaka ali manjša od končne vrednosti.

Primeri:

```
1ØØ FOR L = 1ØØ TO Ø STEP - 1
1ØØ FOR L = PI TO 6 * PI STEP .Ø1
1ØØ FOR AA = 3 TO 3
```

FRE

Tip: Funkcija
Format: FRE (< spremenljivka >)

Delovanje: FRE funkcija nam pokaže, koliko RAM spomina je še na voljo za program in spremenljivke. Če poskuša program porabiti več spomina, kot ga je še na razpolago, se pojavi OUT OF MEMORY napaka. Število v oklepaju ima lahko kakršnokoli vrednost in se ne uporablja pri računanju.

POZOR! Če je rezultat FRE funkcije negativen, mu je potrebno prišteti 65536, tako da dobimo pravilno število bytov, ki so še na voljo.

Primeri: PRINT FRE (Ø)
1Ø X= (FRE (K) - 1ØØØ) / 7
5Ø IF FRE (Ø) < 1ØØ THEN PRINT "NI DOVOLJ SPOMINA!"

POZOR! Naslednji stavek nam izpiše tekoče število RAM-a:
PRINT FRE (Ø) - (FRE (Ø) < Ø) * 65536

GET

Tip: Stavek
Format: GET < lista spremenljivk>

Delovanje: Ta stavek prebere vsak znak, ki ga vtipkamo preko tastature. Medtem ko tipkamo, so znaki shranjeni na vmesniku tastature, in sicer se shrani deset znakov, naslednji pa se izgubijo. GET stavek nam omogoča, da naredimo prostor naslednjemu znaku. Če GET stavek zahteva, da vtipkamo številko, pa tega ne storimo, javi ?SINTAX ERROR. Zato je najboljše, da z GET stavkom beremo znake kot stringe ter jih šele nato po potrebi spremenimo v numerične podatke.
GET stavek lahko uporabimo takrat, ko bi se radi izognili nekaterim omejitvam v INPUT stavku.

Primeri:
```
1Ø GET A$: IF A$ = "" THEN 1Ø : REM ČAKA V VRSTICI 1Ø,
   DOKLER NE PRITISNEMO KATEREKOLI TIPKE
2Ø GET A$, B$, C$, D$, E$: REM PREBERE 5 ZNAKOV
3Ø GET A, A$
```

GET#

Tip: Input/Output stavek
Format: GET# <številka datoteke> , < lista spremenljivk >

Delovanje: Prebere nam znake enega po enega iz priključka ali datoteke, ki jo določimo. Deluje tako kot GET stavek, le da podatki niso vnešeni preko tastature, ampak preko priključka. Če ne sprejme podatka, je znak, ki ga določi prazen string ali Ø za numerično spremenljivko. Znake, ki ločujejo podatke v datotekah, vejice

in [RETURN] znake (koda 13), pa sprejme brez problema.

Če ga uporabimo z # 3 (ekran), bo GET# stavek prebral znake z ekrana enega za drugim. Vsaka uporaba GET# stavka pomakne kursor za eno mesto na desno. Znak na koncu logične vrstice pa se zamenja s CHR$(13) (tipka [RETURN]).

Primeri:

```
 5 GET# 1, A$
1Ø OPEN 1,3: GET# 1, Z7$
2Ø GET# 1, A, B, C$, D$
```

GOSUB

Tip: Stavek

Format: GOSUB <številka vrstice>

Delovanje: To je specifična oblika GOTO stavka, ki pa se od njega razlikuje po tem, da si zapomni, s katerega mesta v programu je bil narejen skok. Ko v programu pridemo do RETURN stavka (to ni [RETURN] tipka!!), se program vrne na stavek, ki sledi GOSUB stavku, s katerim smo skočili v podprogram. Podprogram napišemo takrat, kadar nek del programa uporabimo večkrat v različnih delih programa. Tako si lahko prihranimo mnogo prostora. V tem pogledu je GOSUB stavek podoben DEF FN.

Primeri: Najprej poglejmo program, ki ne uporablja GOSUB:

```
1ØØ PRINT "TA PROGRAM PISE"
11Ø FOR L = 1 TO 5ØØ: NEXT
12Ø PRINT "POCASI NA EKRAN,"
13Ø FOR L = 1 TO 5ØØ: NEXT
14Ø PRINT "TAKO DA UPORABI PREPROSTO ZANKO"
15Ø FOR L = 1 TO 5ØØ: NEXT
16Ø PRINT "ZA UPOCASNITEV PISANJA!"
17Ø FOR L = 1 TO 5ØØ : NEXT
```

Isti program, kjer uporabimo GOSUB:

```
1ØØ PRINT "TA PROGRAM PISE"
11Ø GOSUB 2ØØ
12Ø PRINT "POCASI NA EKRAN,"
13Ø GOSUB 2ØØ
14Ø PRINT "TAKO DA UPORABI PREPROSTO ZANKO"
15Ø GOSUB 2ØØ
16Ø PRINT "ZA UPOCASNITEV PISANJA!"
17Ø GOSUB 2ØØ
18Ø END
2ØØ FOR L = 1 TO 5ØØ : NEXT
21Ø RETURN
```

Vedno, kadar program izvrši GOSUB, se številka vrstice in pozicije v programu shrani na posebno mesto, ki ga imenujemo "stack" ter lahko zavzame do 256 bytov spomina. To nam omejuje število podatkov, ki jih lahko shranimo v "stack". Število adres podprogramov je torej omejeno, zato je potrebno pazljivo določiti RETURN stavke za vsak GOSUB, drugače se lahko zgodi, da nam odtegne spomin, čeprav ga je še dosti na razpolago.

GOTO

Tip: Stavek

Format: GOTO ⟨številka vrstice⟩ ali GO TO ⟨številka vrstice⟩

Delovanje: Ta stavek omogoča BASIC programu, da izvaja stavke tako kot želimo in ne po vrstnem redu številk vrstic. Stavek GOTO povzroči, da program preskoči na vrstico z dano številko. To številko moramo zapisati za besedo GOTO. Z GOTO stavkom lahko kreiramo zanke, ki se nikdar ne končajo. Najbolj preprost primer take zanke je 1Ø GOTO 1Ø; vrača se v svojo vrstico. Take zanke lahko zaustavimo z RUN/STOP tipko na tastaturi.

Primeri: GOTO 1ØØ
1Ø GO TO 5Ø
2Ø GOTO 999

IF...THEN

Tip: Stavek

Format: IF <izraz> THEN <številka vrstice>
IF <izraz> GOTO <številka vrstice>
IF <izraz> THEN <stavek>

Delovanje: Ta stavek imenujemo v BASICU tudi pogojni skok v programu, ker nam omogoča, da se nadaljevanje programa izvrši glede na vrednost izraza, ki ga testiramo. Za besedo IF napišemo izraz, ki lahko vsebuje spremenljivke, stringe, številke, primerjane in logične operatorje. Besedo THEN moramo zapisati v isti vrstici, sledi pa ji lahko številka vrstice, na kateri želimo nato nadaljevati, ali pa BASIC stavek. Če je izraz nepravilen, program ne upošteva navodila za THEN in se nadaljuje v naslednji vrstici. Če pa je izraz pravilen, se izvrši navodilo za besedo THEN.

Primeri: IF ... GOTO

```
1ØØ INPUT "VPISI STEVILKO "; N
11Ø IF N <= Ø GOTO 2ØØ
12Ø PRINT "KVADRATNI KOREN = " SQR(N)
13Ø GOTO 1ØØ
2ØØ PRINT "STEVILKA MORA BITI POZITIVNA"
21Ø GOTO 1ØØ
```

Ta program nam izpiše kvadratni koren kateregakoli pozitivnega števila. IF stavek tukaj uporabljamo, da preverimo številko, ki jo sprejme INPUT stavek. Če je izraz N <= Ø pravilen, program skoči na vrstico 2ØØ, če pa je nepravilen, se nadaljuje na vrstici 12Ø.

IF ... THEN

```
1ØØ FOR L = 1 TO 1ØØ
```

```
11Ø IF RND (1) < .5 THEN X = X + 1: GOTO 13Ø
12Ø Y = Y + 1
13Ø NEXT L
14Ø PRINT "GRBOV = " X
15Ø PRINT "CIFER = " Y
```

IF stavek v 11Ø vrstici testira naključno število, če je manjše od .5. Kadar je rezultat pravilen, se izvrši vse za THEN; najprej poveča X za 1, nato pa skoči na vrstico 13Ø. Če pa je rezultat nepravilen, program izvrši vrstico 12Ø.

INPUT

Tip: Stavek

Format: INPUT ["<tekst> ";] <lista spremenljivk>

Delovanje: INPUT stavek nam omogoča, da s programom vnašamo podatke v računalnik. Med izvajanjem INPUT stavek zapiše vprašaj (?) na ekran ter postavi kursor na desno stran vprašaja. Sedaj program stoji in čaka na podatke. Ko vtipkamo podatke, moramo vedno pritisniti na RETURN tipko. Za INPUT lahko zapišemo tudi tekst v narekovajih. Ta se izpiše na ekran pred vprašajem za vnos podatkov. Za tekstom moramo vedno napisati podpičje ter imena spremenljivk, ki jim bomo priredili podatke. Če je spremenljivk več, jih moramo ločiti z vejicami.

Primeri:

```
1ØØ INPUT A
11Ø INPUT B,C,D
12Ø INPUT "STEVILO "; E
```

Ko ta program steče, se najprej prikaže vprašaj, nato vtipkamo neko število - to je vrednost spremenljivke A. Če ne vtipkamo števila, ampak kakšen drug znak, se pojavi?REDO FROM START sporočilo, kar pomeni, da je sprejel string, ko je pričakoval numerični podatek. Če v tem primeru pritisnemo samo na RETURN, se vrednost spremenljivke ne spremeni.

Sedaj se v programu pojavi naslednji vprašaj za INPUT na 11Ø vrstici. Če sedaj vtipkamo samo en podatek, se prikažeta dva vprašaja (??), kar pomeni, da program zahteva več podatkov, ki jih vtipkamo ločene z vejicami. Če pa poskušamo vtipkati več podatkov, se pojavi sporočilo ?EXTRA IGNORED, kar pomeni, da podatke ne upošteva. Vrstica 12Ø nam izpiše besedo STEVILO pred vprašajem.
INPUT stavek ne moremo nikoli uporabiti neposredno, ampak samo v BASIC programu.

INPUT

Tip: Input/Output stavek
Format: INPUT# <številka datoteke> , <lista spremenljivk>

Delovanje: To je po navadi najlažja in najhitrejša pot, da preberemo podatke, ki smo jih shranili na disketi ali kaseti. Podatke bere v njihovi celotni dolžini, za razliko od GET# stavka, ki jih bere znak za znakom. Najprej moramo datoteko odpreti z OPEN stavkom, nato pa lahko uporabimo INPUT#.
INPUT# stavek predpostavlja, da je spremenljivke konec, ko prebere CHR$(13) (RETURN kodo), vejico (,), podpičje (;) ali dvopičje (:).
Če je tip spremenljivke numeričen podatek, INPUT# pa sprejme nenumerične znake, se izpiše BAD DATA ERROR. INPUT# lahko prebere stringe dolge do 8Ø znakov, če je string daljši, izpiše STRING TOO LONG napako.
Kadar uporabljamo INPUT# stavek s #3 (ekran), nam prebere celotno logično vrstico ter premakne kursor v naslednjo vrstico.

Primeri:
```
1Ø INPUT# 1,A
2Ø INPUT# 2, A$, B$
```

INT

Tip: Integer funkcija

Format: INT (<izraz>)

Delovanje: Izračuna najmanjši integer vrednost izraza v oklepaju. Če je izraz pozitiven, potem odreže število pri decimalni vejici, če pa je izraz negativen, ga zaokroži na naslednje manjše celo število.

Primeri: PRINT INT (99.4343), INT (-12.34)
99 -13

LEFT$

Tip: String funkcija

Format: LEFT$ (<string>, <integer>)

Delovanje: Funkcija določi string, ki ga sestavljajo znaki levega dela stringa v oklepaju, število teh znakov pa je določeno z integer številko v oklepaju. Ta številka je lahko med Ø in 255. Če je številka večja od dolžine stringa, potem funkcija določi celoten string. Če je številka Ø, pa imamo prazen string.

Primeri:
```
1Ø A$ = "COMMODORE RACUNALNIKI"
2Ø B$ = LEFT$ (A$,9) : PRINT B$
RUN
COMMODORE
```

LEN

Tip: Integer funkcija

Format: LEN(<string>)

Delovanje: Izračuna število znakov v stringu (dolžina stringa); pri tem se štejejo tudi znaki, ki niso vidno zapisani ter prazni prostori.

Primeri:
```
CC$ = "COMMODORE COMPUTER" : PRINT LEN (CC$)
```

LET

Tip: Stavek

Format: [LET] <spremenljivka> = <izraz>

Delovanje: Stavek LET uporabljamo, da priredimo vrednost spremenljivki, ta stavek pa lahko tudi povsem opustimo, ker je že znak = sam po sebi dovolj, da se priredi izraz imenu spremenljivke.

Primeri: 1Ø LET D = 12 (isto kot D = 12)
2Ø LET E$ = "ABC"

LIST

Tip: Ukaz

Format: LIST [[<prva vrstica>] - [<zadnja vrstica>]]

Delovanje: Ukaz LIST nam omogoča, da pregledamo vrstice BASIC programa, ki ga imamo trenutno v spominu računalnika. Omogoča nam, da uporabimo SCREEN EDITOR, s katerim lahko hitro in enostavno popravljamo in dopolnjujemo programe. Če želimo prelistati program na ekranu, si lahko pomagamo s tipko CTRL, ki bo upočasnila listanje, ko pa želimo listanje ustaviti, pritisnemo na RUN/STOP tipko.
Če LIST ukazu ne dodamo številke vrstice, nam izlista celotni program. Če določimo samo prvo vrstico in dodamo pomišljaj, nam izlista vse vrstice z višjimi programskimi številkami. Če zapišemo pomišljaj in številko vrstice, nam izpiše program od začetka pa do dane vrstice. Če pa določimo prvo vrstico in zadnjo vrstico ter njuni številki ločimo s pomišljajem, nam izpiše del programa med tema dvema vrsticama.

Primeri:
```
LIST          (izpiše program v spominu)
LIST 5ØØ      (izpiše vrstico 5ØØ)
LIST 15Ø -    (izpiše vse vrstice od 15Ø do konca
               programa)
LIST - 1ØØØ   (izpiše program od začetka do 1ØØØ
               vrstice)
```

```
1270 REM ***************************
1280 REM *   NEW COLOUR            *
1290 REM ***************************
1300 CMOB 7,2
1310 PRINT "␣"
1320 REM ***************************
1330 REM * HIGH-RES.-MOB NO.5     *
1340 REM * FROM BLOCK 14          *
1350 REM ***************************
1360 MOB SET 5,14,6,0,0
1370 REM ***************************
1380 REM * HIGH-RES.-MOB NO.4     *
1390 REM * FROM BLOCK 13          *
1400 REM ***************************
1410 MOB SET 4,13,9,5,0
1420 REM ***************************
1430 REM * THE SAME POSITION      *
1440 REM *   FOR MOB NO.4 & 5     *
1450 REM * EXP. FOR MOB NO.5 IN   *
1460 REM *     X & Y WAY          *
1470 REM ***************************
1480 Y=150:Z=50
1490 FOR X=30 TO 280 STEP 4
1500 Y=Y-2:Z=Z+2
1510 IF Y<60 THEN Y=60
1520 IF Z>140 THEN Z=140
1530 FOR DY=1 TO 10
1540 RLOCMOB 5,X,Y+DY*4,3,0
1550 RLOCMOB 4,X,Z+DY*2,0,1
1560 NEXT DY
1570 NEXT X
1580 REM ***************************
1590 REM *  DELETE MOB             *
1600 REM ***************************
1610 PRINT "␣MOB DELETE (Y/N)"
1620 FETCH "YN",1,ANT$
1630 IF ANT$="N" THEN 1670
1640 FOR L=0TO7
1650 :  MOB OFF L
1655 PAUSE 2
1660 NEXT L
1670 END
READY.
```

10. IZDELAVA LASTNIH ZNAKOV

Simon`s Basic nam omogoča izdelavo svojih znakov. To pomeni, da lahko znake, ki so nam na vodljo preko tastature zamenjamo s svojimi in tako pišemo texte s svojimi črkami (Č,Š,Ž...).

10.1. MEM

Format: MEM

Namen: Premestitev standardnega seta znakov iz ROM-a v RAM.

Delovanje: Vsi znaki, (črke, številke, grafični simboli), ki so nam dostopni preko tastature CBM-64 se nahajajo v ROM-u (t.j. memoriji v katero ne moremo pisati). Če želimo kakšne znak spremeniti moramo najprej standardni set znakov prenesti v RAM, t.j. v memorijo v katero lahko pišemo. Hkrati moramo standardni set znakov blokirati. To pomeni, da računalnik ob pritisku na določeno tipko ne bo izpisal znaka, ki se nahaja v ROM-u, ampak znak kakršnega smo definirali in se nahaja v RAM-u. Ukaz MEM naredi torej naslednje:

- prenese standardni set znakov iz ROM-a v RAM na lokacijo $EØØØ t.j. takoj pod KERNAL ROM tako, da ne zavzemajo memorijo namenjeno Basicu.
- prenese ekransko memorijo na lokacijo $CCØØ
  POZOR: Če imamo na tej lokaciji kakšen strojni program se ta ukaz ne sme uporabljati!
- omeji MOB bloke na 192 - 255

Ko se želimo vrniti na standardni set znakov pritisnemo RUN/STOP in RESTORE.

10.2. DESIGN

Format: DESIGN 2,$EØØØ + CH*8

Namen: Menjati standardni set znakov.

Delovanje: Ta ukaz menja standardni znak katerega določa spremenljivka CH z znakom, ki smo ga izdelali sami. Vrednost CH najdemo v ASCII tabeli.
Vsak znak je sestavljen iz matrice 8 X 8 bitov.

Primer: Glej primer v naslednji točki 10.3.!

./.

10.3. Master space (znak štev. 64)

Format: @.....

Namen: Določiti obliko novega znaka.

Delovanje: S tem ukazom lahko dokaj enostavno izdelamo nov znak. Ker se vsak znak sestoji iz matrice 8 X 8 bitov je potrebno 8 vrstic v katerij je prvi znak @ nato pa 8 točk. Če namesto točke vpišemo črko B, pomeni, da bo ta točka na ekranu osvetljena (enako kot pri izdelavi MOB-a).

Primer:

```
100 REM MENJAVA ČRKE A Z NOVIM ZNAKOM
110 MEM
120 DESIGN 2,$EØØØ+1*8
130 @BBBBBB.
140 @BB...B.
150 @ BB....
160 @..BB...
170 @..BB....
180 @.BB.....
190 @.BBBBBB.
200 @........
```

11. PROGRAMIRANJE GLASBE

Na računalniku CBM-64 lahko programiramo tudi glasbo, saj računalnik razpolaga s tremi med samo neodvisnimi generatorji tona. Te tri generatorje lahko uporabimo vsakega zase ali pa vse tri skupaj in tako dobimo možnost za programiranje zahtevnejših glasbenih del. Vsak generator je možno uporabiti v eni od štirih valnih oblik tona in v razponu osmih oktav. Vlane oblike tona so sledeče: trikotna oblika, žagasta oblika, štirikotna oblika tona in šum. Na ta način je možno posnemati različne instrumente. V osnovnem Basicu moramo za delo s temi generatorji uporabljati ukaz POKE, v Simon`s Basicu pa nam je na voljo nekaj ukazov, ki nam olajšajo delo z njimi. Če želimo na CBM-u slišati kakšen ton moramo določiti naslednje spremenljivke:

- glasnost
- valno obliko
- glas
- noto
- obliko ovojnice (envelope)

./.

11.1. VOL

Format: VOL N

Namen: Nastavljanje jakosti

Delovanje: Ukaz določi jakost vseh treh generatorjev tona. Spremenljivka N lahko ima vrednost med Ø in 15.

N = Ø generatorji tona so izklopljeni
N =15 maksimalna jakost

Primer: Glej primer na koncu poglavja!

11.2. WAVE

Format: S, BINARNO ŠTEVILO

Namen: Določanje valne oblike.

Delovanje: S tem ukazom določamo karakteristiko zvoka za posamezen generator. Spremenljivka S nam pove kateri generator želimo nastaviti:

S = 1 določamo parametre za 1. generator
S = 2 določamo parametre za 2. generator
S = 3 določamo parametre za 3. generator

Druga spremenljivka je binarno število, ki odreja karakteristike izbranemu generatorju. Vrstni red bitov je naslednji: 7 6 5 4 3 2 1 Ø in imajo naslednji pomen:

BIT	POMEN
Ø	postavlja GATE BIT (ga ni potrebno post.)
1	sinhronizacija
2	ring modulacija
3	test bit
4	valna oblika je trikotna
5	valna oblika je žagasta
6	valna oblika je štirikotna
7	šum

Podrobna obrazložitev posameznih bitov

Bit Ø - GATE BIT

Ta bit vključuje generator tona. Kadar ga postavimo na (1) se incializira ATTACK-DECAY-SUSTAIN ciklus. Če pa ga postavimo na (Ø) se incializira RELEASE ciklus. S tem bitom upravlja sistem sam, ko izvaja ukaz PLAY.

BIT 1 - SINHRONIZACIJA

Ta bit nam omogoča sinhronizacijo enega generatorja z drugim po naslednjih zakonitostih:

./.

Bit postavljen	Funkcija
v glasu 1	sinhronizira osnovo frekv.glasa 1 z glasom 3
v glasu 2	sinhronizira osnovo frekv.glasa 2 z glasom 1
v glasu 3	sinhronizira osnovo frekv.glasa 3 z glasom 2

Če sinhroniziramo glas z spremeljivo frekvenco z glasom fiksne frekvence dobimo kompleksne zvoke. Najboljše rezultate dobimo če fiksno frekvenco izberemo nižje od spremenljive.

BIT 2 - RING MODULACIJA

Z ring modulacijo menjamo trikotno valno obliko z drugo, ki nastane s kombiniranjem izbranega glasu z drugim glasom. Tudi tu je priporočljivo, kot pri sinhronizaciji, vzdrževati konstantno frekvenco enega glasu, frekvenco drugega pa spreminjati. Katere glasove lahko kombiniramo prikazuje naslednja tabela:

Glas (mora biti trikotne oblike)	Modulacijski signal
1	glas 1 z glasom 3
2	glas 2 z glasom 1
3	glas 3 z glasom 2

Pri tem nastajajo neharmonični tonski nihaji, s katerimi se lahko dosegajo posebni zvočni efekti (zvonec, gong...)

BIT 3 - TEST BIT

Običajno se ne uporablja. S postavljanjem na (1) nato pa na (Ø) se vrši "RESET" (incializacija generatorja zvoka).

BIT 4 - TRIKOTNA VALNA OBLIKA

Vključuje generator trikotne valne oblike. Ta oblika ima mnogo harmoničnih nadtonov.

BIT 5 - ŽAGASTA VLANA OBLIKA

Vključuje generator žagaste valne oblike. Ta valna oblika ima mnogo ravnih in neravnih harmoničnih nadtonov.

BIT 6 - PRAVOKOTNA VALNA OBLIKA

Vključuje generator pravokotne valne oblike pri katerem se delež nadtonov odreja s širono nihaja (pulsa).

./.

BIT 7 - ŠUM

Vključuje generator šuma. Šum je odvisen od frekvence oscilatorja.

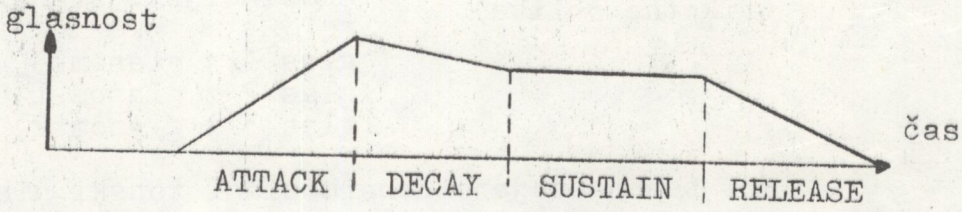
Primer: Postavi parametre za prvi glas
1ØØ WAVE 1,ØØØ1ØØØØ

11.3. ENVELOPE

Format: ENVELOPE Glas, Attack, Decay, Sustain, Release

Namen: Nastavitev tonske krivulje

Delovanje: Za vsak glas moramo nastaviti tonsko krivuljo, ki določa kako bo kakšen ton - nota odigran. Določiti je potrebno štiri parametre in sicer ATTACK, DECAY, SUSTAIN in RELEASE. Vrednosti se gibljejo za vse štiri parametre od Ø do 15. Kaj predstavlja posamezen parameter je vidno iz diagrama.



Primer: 30 ENVELOPE 1,8,8,0

11.4. MUSIC

Format: MUSIC n, "note" ali MUSIC n,ime stringa

Namen: Določiti zaporedje not, ki jih želimo slišati.

Delovanje: Z ukazom MUSIC določimo vse parametre za neko melodijo. Spremenljivka "n" ima lahko vrednost od 1 do 255 in določa trajanje tonov. Naslednji parameter odreja dolžino in višino ene ali večih not. V stringu mora biti prvi parameter SHIFT CLR/HOME in številka od 1 do 3, ki incializira enega od treh generatorjev. Nato sledi takt, ki ga določimo za vsako noto (šestnajstinka, osminka ...) in to z funkcijskimi tipkami F1 - F8 in končno ime note, ki ga označimo s črkami C - B (uporablja se B in ne H!).

./.

ZNAKI, KI OZNAČUJEJO TRAJANJE NOTE

Znak	Funkcija
SHIFT CLR/HOMEn	n (1-3) določa glas (generator)
F1	naslednja nota traja 1/16 polne note
F2	" " " 1/8 " "
F3	" " " 1/4 " "
F4	" " " 1/2 " "
F5	" " " 1/1 " "
F6	" " " 2/1 " "
F7	" " " 4/1 " "
F8	" " " 8/1 " "

Vsaka enota je sestavljena iz črke (C-B) in številke oktave (0-7). Za poltona povišano noto dobimo tako, da skupaj s črko, ki označuje noto pritisnemo SHIFT. Na koncu vsakega notnega stringa je potrebno vpisati SHIFT CLR/HOME in G. Ta kombinacija starta RELEASE odsek zadnje note. Pavza se označi s črko Z (brez št. oktave)

11.5. PLAY

Format: PLAY n

Namen: Igranje tonov po določenih parametrih.

Delovanje: Ta ukaz prične igrati melodijo, ki smo jo skomponirali. Spremenljivka n določa kaj se bo zgodilo z nadaljnjim potekom programa.

n = Ø konec glasbe
n = 1 igraj! Program čaka, da se odigra cela melodija
n = 2 igraj! Program nadaljuje delo.

Primer:

```
1340 VOL 10
1350 WAVE 1,00010000
1360 ENVELOPE 1,12,8,8,0
1370 S$="■1Z■"
1380 A1$="G3■F3■E3■E3■E3■D3■E3■F3■D3■"
1390 A2$="A3■G3■F3■F3■F3■E3■F3■G3■E3■"
1400 A3$="A3■G3■B3■A3■G3■F3■E3■"
1410 E$="■G"
1420 MUSIC 6,S$+A1$+A2$+A1$+A3$+E$
1430 PLAY 2
```

12. UKAZI ZA LIGHT-PEN, JOYSTICK IN PADDLE

Simon`s Basic nam omogoča, da s štirimi ukazi lahko kontroliramo pozicije zunanjih priključkov (položaj potenciometra, igralne palice ali svetlobnega peresa).

12.1. PENX

Format: SX-PENX

Namen: Ugotoviti položaj svetlobnega peresa (X-os)

Delovanje: Po izvršitvi ukaza dobimo v spremenljivki SX vrednost med Ø in 320, ki pomeni položaj LIGHT-PEN-a relativno od levega roba ekrana. Ukaz se mora izvesti vedno pred ukazom PENY.

Primer:
```
100 LX = PENX
```

12.2. PENY

Format: SY = PENY

Namen: Ugotoviti položaj svetlobnega peresa (Y-os)

Delovanje: V spremenljivki SY dobimo po izvršitvi ukaza vrednost med Ø in 200, ki pomeni položaj LIGHT-PEN-a relativno od zgornjega roba ekrana. Pred tem ukazom se mora izvršiti ukaz PENX.

Primer:
```
110 LY = PENY
```

12.3. POT

Format:
```
PØ = POT(Ø)
Pl = POT(1)
```

Namen: Ugotoviti položaj paddle-a (potenciometra)

Delovanje: Ukaz nam omogoča, da v spremeljivki PØ ali Pl dobimo vrednost med Ø in 255, ki pomeni položaj potenciometra. Ker sta lahko priključena dva paddle-a nam številka v oklepaju pove kateri paddle želimo kontrolirati.

Primer:
```
100 PØ = POT(Ø)
```

12.4. JOY

Format: JP = JOY

Namen: Ugotoviti položaj ročice na joystick-u.

./.

Delovanje: V spremenljivki JP dobimo kodo za položaj v katerem se nahaja ročica jostick-a. Kontroliramo lahko samo port-2!

Kode položaja so naslednje:

```
8 1 2
7 0 3
6 5 4
```

Če je pritisnjena tipka "FIRE" dobimo kodo 128.

Primer: JP = JOY

13. STRUKTURIRANO PROGRAMIRANJE

V standardnem Basicu nimamo možnosti napisati nobenega podprograma, ki bi ga lahko potem po potrebi klicali. Ukaza GOTO in GOSUP naredita program zelo nepregleden in nerazumljiv. V Simon`s Basicu je ta problem rešen z nekaterimi ukazi za strukturirano programiranje.

UKAZI ZA KONTROLO TOKA PROGRAMA

13.1. IF ... THEN ... ELSE

Format: IF (pogoj) THEN (ukazi): ELSE: (ukazi)

Namen: V eni vrsti testirati pogoj in izvršiti navedene ukaze.

Delovanje: Ukaz je podoben IF ... THEN ukazu iz standardnega Basica, razlika je v tem, da lahko dodamo dodatne ukaze, ki se bodo izvršili če pogoj ni izpolnjen.

POZOR: Ukaz ELSE moramo vpisati med dvopičji!

Primer:

```
1ØØ PRINT"ALI PONOVIM POSTOPEK (D/N)"
110 GET A$: IF A$ ="" THEN GOTO 110
120 IF A$ ="D" THEN END: ELSE: GOTO 10
```

13.2. REPEAT ... UNTIL

Format: REPEAT (ukazi) UNTIL (pogoj)

Namen: Ukaz omogoča ponavljanje več ukazov

Delovanje: S tem ukazom omogočimo ponavljanje več ukazov dokler se ne izpolni določen pogoj. REPEAT označuje začetek ukazov, ki se morajo ponavljati, UNTIL pa vsebuje pogoj.

./.

Primer:

```
100 REM  IZPISATI ČRKE OD A DO G
110 A=65 : REM CHR$ KODA ZA ČRKO A
120 REPEAT
130 : PRINT CHR$ (A);
140 : A=A+1
150 UNTIL A>71
160 END
```

13.3. RCOMP

Format: RCOMP (ukazi če je pogoj izpolnjen): ELSE (če ni izpolnjen)

Namen: Nadaljevati IF .. THEN .. ELSE v naslednji vrsti.

Delovanje: RCOMP uporabljamo takrat ko želimo razširiti IF stavek v naslednjo vrsto in pri tem ne želimo ponovno pisati isti IF .. THEN ukaz.

Primer:

```
100 INPUT A
110 IF A=0 THEN PRINT +A = 0" : ELSE : PRINT
    "A<> 0"
120 RCOMP: PRINT "(NIČ)" : ELSE: PRINT "(NI NIČ)"
    (RCOMP je uporabljen namesto IF A=0 THEN)
```

13.4. LOOP .. EXIT IF .. END LOOP

Format: LOOP (niz ukazov) EXIT IF (pogoj) END LOOP

Namen: Ponavljati del programa dokler ni izpolnjen določen pogoj.

Delovanje: Ta instrukcija se uporablja, kadar je potrebno ponoviti del programa večkra, število ponovitev pa ni znano.

Primer:

```
100 LOOP              :REM PRIČETEK ZANKE
110 : GET A$          :REM PREBERI TASTATURO
120 : EXIT IF A$
      =""             :REM ČE JE SPACE KONČAJ ZANKO
130 : PRINT A$        :REM IZPIŠI A$
140 : END LOOP        :REM KONEC ZANKE
150 : PRINT  "NADALJEVANJE"
```

PROCEDURE

Za olajšanje pisanja strukturiranih programov nam Simon`s Basic omogoča pisanje t.i. podprogramov in procedur. To so programi, ki jih kličemo z imenom in ne s številko vrstice kot je to v standardnem Basic-u z ukazom GOSUB.

13.5. PROC

Format: PROC labela

Namen: Definirati simbolični naslov (labelo)

./.

Delovanje: Z ukazom PROC dajemo naslednjemu ukazu simbolično ime. Vsi znaki, ki sledijo ukazu se smatrajo za ime, tako, da se v isti vrsti ne sme pisati drugih ukazov.

Primer:
```
100 PROC TEST
```

13.6. END PROC

Format: END PROC

Namen: Označiti konec podprograma.

Delovanje: END PROC označuje konec podprograma, ki smo ga pričeli z ukazom PROC. Če smo program poklicali z ukazom EXEC (13.8) takrat ukaz deluje kot RETURN.

Primer:
```
100 PROC TEST
110 PRINT "TO JE PODPROGRAM TEST"
120 END PROC
```

13.7. CALL

Format: CALL labela

Namen: Skoči na navedeni simbolični naslov.

Delovanje: Ukaz služi namesto GOTO ukaza takrat ko želimo skok na simbolični naslov in ne na število ukaza.

Primer:
```
100 IF A$ = "STOP" THEN CALL ENDPROG
```

13.8. EXEC

Format: EXEC labela

Namen: Klic podprograma s simboličnim imenom.

Delovanje: Ukaz EXEC deluje enako kot ukaz GOSUP, vendar za razliko od GOSUB ni potrebno vedeti številke ukaza. Ko je klican podprogram končal delo se kontrola delovanja vrne na prvi ukaz, ki sledi ukazu EXEC.

Primer: Pokliči podprogram TEST
```
100 PRINT "KLIČEMO TEST"
110 EXEC TEST
120 PRINT "PODPROGRAM TEST JE KONČAN"
```

Primer: Skoči na del programa, ki se imenuje TEST
```
100 PRINT "SKOČIMO NA TEST"
110 CALL TEST
120 REM SEM SE NE BOMO VEČ VRNILI
```

./.

Primer: Malo bolj kompliciran program, ki včita nekaj imen s tastature dokler ne vtipkamo "STOP". Takrat se imena dodajo v datoteko in program konča.

```
50 I=1
100 PROC START
110 EXEC VHOD
120 IF A$="STOP" THEN CALL KONEC
130 IF A$="VPIS" THEN EXEC VPIS
140 B$ (I)=A$: I=I+1
150 CALL START
160 PROC VHOD          (procedura za vpis)
170 INPUT A$
180 END PROC
190 PROC VPIS          (proced.za izpis doslej
200 FOR J=1 TO I       vpisanih imen na ekran)
210 : PRINT B$(I)
220 NEXT
230 END PROC
240 PROC KONEC         (proced.dodaja imena v dat.)
250 OPEN 2,8,2,"imena,S,A"
260 FOR J=1 TO I
270 : PRINT#2,B$(I)
280 NEXT
290 END
```

14. SPREMENLJIVKE V BASICU

V standardnem Basic-u so vse spremenljivke globalne, kar pomeni, da je kjerkoli definirana spremenljivka od takrat dalje znana celemu programu do konca. To je včasih neugodno, ker moramo paziti, da v podprogramu ne zamenjamo kakšne spremenljivke, ki jo potrebujemo samo v glavnem programu. Simon`s Basic uvaja t.i. lokalne spremenljivke. To so spremenljivke, ki obstajajo samo v določenem primeru in jih lahko potem enostavno zbrišemo. Takšen način dela tudi pospešuje izvajanje podprogramov. Kajti če je manj nazivov spremenljivk jih Basic hitreje pregleda in najde zahtevano.

14.1.LOCAL

Format: LOCAL SPR1, SPR2,.. SPRn

Namen: Določiti lokalne spremenljivke.

Delovanje: Ukaz naredi novo kopijo navedenih spremenljivk, tako, da vrednosti v spremeljivkah z istim imenom uporabljenih kje drugje v programu ne spremeni.

./.

Primer: Glej primer za 14.2.

14.2.GLOBAL

Format: GLOBAL

Namen: Vrniti vrednosti spremeljivk, ki so jih imele pred izvršitvijo ukaza LOCAL.

Delovanje: Ko ne potrebujemo več lokalnih spremenljivk, moramo spremenljivkam vrniti vrednosti, ki so jih imele pred izvršitvijo ukaza LOCAL.

Primer:

```
100 A=10 : B%=10:A$="AA"
110 PRINT A,B%,A$                    -izpis : 10  10  AA
120 EXEC PODPR
130 PRINT A,B%,A$                    -izpis : 20  20  CC
140 GLOBAL
150 PRINT A,B%,A$                    -izpis : 10  20  AA
160 CALL KONEC
170 PROC PODPR
180 LOCAL A,A$                       -B% ni lokal!
190 A=20: B%=20: A$="CC"
200 PRINT A,B%,A$                    -izpis : 20  20  CC
210 END PROC
220 PROC KONEC
230 PRINT "K O N E C"
240 END
```

LIST 15Ø - 1ØØØ (izpiše del programa med 15Ø do 1ØØØ)

```
1Ø PRINT "TO JE VRSTICA 1Ø"
2Ø LIST
3Ø PRINT "TO JE VRSTICA 3Ø"
```

LOAD

Tip: Ukaz

Format: LOAD ["<ime datoteke>"] [, <priključek>] [, <adresa>]

Delovanje: Stavek LOAD prebere vsebino programske datoteke iz traku ali diskete v spomin računalnika; številko priključka lahko izpustimo, takrat bo računalnik avtomatično izvedel polnjenje iz kasetofona, torej priključka s številko 1. Diskovna enota ima številko 8. LOAD ukaz zapre vse odprte datoteke, in če ga uporabljamo kot direkten ukaz, se izvrši tudi CLR (clear), preden začne s polnjenjem. Če uporabimo LOAD v programu samem, lahko tako zvežemo več programov skupaj; v tem primeru namreč ne čisti spomina.

V primeru, ko ime datoteke, ki jo želimo prebrati, ne obstaja, javi ?FILE NOT FOUND napako. Kadar polnimo programe iz kasete, lahko izpustimo ime programa; v tem primeru prebere prvi program na traku. CBM 64 nam izbriše ekran vedno kadar deluje ukaz LOAD in pritisnemo tipko PLAY na kasetofonu. Ko se ekran spet vključi, izpiše FOUND "ime programa" . Sedaj pritisnemo na [C=] , [CTRL], [←] ali [SPACE] tipko in program se začne polniti na spominsko lokacijo 2Ø48, razen če uporabimo drugo adreso v LOAD stavku.

Primeri:

```
LOAD          (Prebere program iz traku)
LOAD A$       (Uporabi ime, ki je v A$ za iskanje)
LOAD "*",8    (Prebere prvi program na disketi)
LOAD "", 1,1  (Išče prvi program na traku in ga
                napolni v isti del spomina, s
```

```
katerega je bil shranjen.)
LOAD "STAR TREK"  (Prebere program iz traku)
PRESS PLAY ON TAPE
FOUND STAR TREK
LOADING
READY.

LOAD "FUN",8      (Prebere program iz diskete)
SEARCHING FOR FUN
LOADING
READY.

LOAD "GAME ONE",8,1 (Prebere program na lokacijo, s
                     katere je bil shranjen na disketo)
SEARCHING FOR GAME ONE
LOADING
READY.
```

LOG

Tip: Floating-point funkcija
Format: LOG (<število>)

Delovanje: Izračuna naravni logaritem (logaritem z osnovo e) argumenta. Če je vrednost argumenta Ø ali negativno število, se javi ?ILLEGAL QUANTITY napaka.

Primeri:

```
PRINT LOG (45/7)
1.86Ø75234

1Ø NUM = LOG (ARG) / LOG (1Ø)
(Izračuna desetiški logaritem ARG)
```

MID$

Tip: String funkcija
Format: MID$ (<string> , <število - 1> [, <število -2>])

Delovanje: Funkcija MID$ določi string, ki ga vzame iz nekega večjega stringa. Začetno pozicijo definira s številom - 1, medtem ko je dolžina definirana s številom - 2. Oba ta numerična argumenta imata lahko vrednosti med Ø in 255.

Če je vrednost števila - 1 večja od dolžine stringa ali če je vrednost števila - 2 nič, nam MID$ določi prazen string. Če izpustimo število - 2, potem odreže preostali del stringa.

Primeri:
```
1Ø A$ = "GOOD"
2Ø B$ = "MORNING EVENING AFTERNOON"
3Ø PRINT A$ + MID$ (B$, 8, 8)
GOOD EVENING
```

NEW

Tip: Ukaz

Format: NEW

Delovanje: Ukaz NEW uporabljamo za čiščenje tekočega programa in spremenljivk iz spomina. Vedno, preden želimo vtipkati nov program, moramo uporabiti instrukcijo NEW kot direkten ukaz.

POZOR! Če ne zbrišemo starega programa iz spomina in vtipkamo novega, lahko to povzroči, da se programske vrstice obeh programov med seboj pomešajo.

Primeri:
```
NEW      (Izbriše program in vse spremenljivke)
1Ø NEW   (Ustavi program, potem ko ga tudi zbriše)
```

NEXT

Tip: Stavek

Format: NEXT [<števec>][, <števec>].....

Delovanje: Stavek NEXT se uporablja v kombinaciji s FOR stavkom, in sicer zaključi FOR ... NEXT zanko. Števec imenujemo spremenljivko v FOR stavku, ki začne zanko. Za več različnih FOR stavkov lahko uporabimo samo en NEXT stavek, ki mu dodamo imena števcev po vrstnem redu, ločene z vejicami. Zanko, ki smo jo zadnjo začeli, moramo najprej končati. Eno v drugi lahko zapisujemo do 9 zank naenkrat. Vedno,

ko zanka doseže NEXT stavek, števec naraste za vrednost koraka (STEP) v zanki; v primeru ko je zadnja vrednost (zanke) že dosežena, pa NEXT zaključi zanko in program se nadaljuje z naslednjim stavkom.

Primeri:
```
1Ø FOR J = 1 TO 5: FOR K = 10 TO 2Ø: FOR N = 5 TO
   - 5 STEP - 1
2Ø NEXT N, K, J
```
(Zaključujemo zanke, ki so druga v drugi)

```
1Ø FOR L = 1 TO 1ØØ
2Ø FOR M = 1 TO 1Ø
3Ø NEXT M
4Ø NEXT L
```
(Zanke ne smejo križati druga drugo)

```
1Ø FOR A = 1 TO 1Ø
2Ø FOR B = 1 TO 2Ø
3Ø NEXT
4Ø NEXT
```
(Imena spremenljivk v NEXT stavkih ne potrebujemo.)

NOT

Tip: Logični operator
Format: NOT <izraz>

Delovanje: NOT je logični komplement vrednosti vsakega bita ter nam izračuna dvojiški komplement. Če uporabljamo floating-point števila, se operandi najprej pretvorijo v integer, decimalni del je v tem primeru izgubljen. Operator NOT uporabljamo tudi v primerjavah pri testiranju pravilne/nepravilne vrednosti izraza.

Primeri:
```
1Ø IF NOT AA = BB AND NOT (BB = CC) THEN ...

NN% = NOT 96: PRINT NN%
- 97
```

POZOR! Za izračun vrednosti NOT uporabljamo izraz X = (-(X+1)). (Dvojiški komplement kateregakoli integer števila je komplement bita plus 1.)

ON

Tip: Stavek

Format: ON <spremenljivka> GOTO / GOSUB <številka vrstice> [, <številka vrstice>] ...

Delovanje: Stavek ON uporabljamo za programski skok na različne vrstice v odvisnosti od vrednosti spremenljivke. Ta vrednost je lahko med Ø in številom danih programskih vrstic. Če ni integer, se decimalni del odreže. Na primer, če je vrednost spremenljivke 3, to pomeni, da bo GOTO stavek izvedel skok na številko programske vrstice, ki je zapisana tretja za GOTO. Če je vrednost spremenljivke negativna, se pojavi napaka ?ILLEGAL QUANTITY. Če je vrednost Ø ali večja od števila programskih vrstic, program preskoči ON stavek in nadaljuje v naslednji vrstici.
ON stavek lahko uporabljamo kot več pogojnih skokov, v določenem primeru nadomesti več IF ... THEN stavkov.

Primeri:
```
ON - (A=7) - 2 * (A=3) - 3 * (A  3) - 4 * (A  7)
GOTO 4ØØ, 9ØØ, 1ØØØ, 1ØØ

ON X GOTO 1ØØ, 13Ø, 18Ø, 22Ø
ON X + 3 GOSUB 9ØØØ, 2Ø, 9ØØØ

1ØØ ON NUM GOTO 15Ø, 3ØØ, 32Ø, 39Ø
5ØØ ON SUM/2 + 1 GOSUB 5Ø, 8Ø, 2Ø
```

OPEN

Tip: Input/Output stavek

Format: OPEN <številka datoteke> [, <priključek] [, <adresa>] ["<ime datoteke> [, <tip>] [, <mod>]"]

Delovanje: Stavek OPEN odpre kanal za input ali output na zunanji priključek. Vedno ne potrebujemo vseh določil za OPEN stavek, največkrat sta pomembni dve kodi:
a) logična številka datoteke
b) številka priključka.
Logična številka datoteke je številka, ki jo uporabljamo s stavki OPEN, CLOSE, CMD, GET#, INPUT# in

PRINT# vzporedno z imenom datoteke. To je lahko število med 1 in 127.
Vsak zunanji priključek (printer, disk-drive, kasetofon) v sistemu ima svojo številko, na katero odgovori. Številka priključka se uporablja zato, da določimo, na katerem priključku odpiramo datoteko. Če to številko izpustimo, bo računalnik poslal podatke na priključek s številko 1, to pomeni na kasetofon.
Tudi ime datoteke lahko izpustimo, toda pozneje v programu ne smemo klicati datoteke z imenom, če ji tega nismo dali v OPEN stavku. Če shranjujemo podatke na kaseto, bo računalnik predpostavil, da je adresa Ø, kadar izpustimo številko adrese v READ stavku. Adresa, ki ima vrednost 1 odpre datoteke na kasetofonu za pisanje. Ko zaključimo pisanje, moramo tej adresi dati vrednost 2. To preprečuje, da beremo podatke po koncu datoteke, kar lahko povzroči ?DEVICE NOT PRESENT napako. Pri diskovnih datotekah so na razpolago številke adres med 2 in 14, druge številke imajo posebne pomene v DOS ukazih. Če uporabljamo disk-drive, moramo določiti številko adrese.
Ime datoteke je string dolg največ 16 znakov ter ga lahko tudi izpustimo, kadar uporabljamo kasetofon ali printer. Če izpustimo ime datoteke, bo to avtomatično programska datoteka, razen če ni dan mod. Sekvenčne datoteke odpremo za branje z mod = R (reading), razen če ne določimo, da bomo datoteko pisali mod = W (writing). Relativne (REL) in sekvenčne datoteke lahko uporabljamo le na disketi. Če poskušamo formirati datoteko preden smo jo odprli, nam javi ?FILE NOT OPEN napako. Če hočemo odpirati datoteko za branje, ki je ni na disketi, nam javi ?FILE NOT FOUND napako. Če odpremo datoteko z imenom, ki že obstaja na disketi, se izpiše sporočilo FILE EXISTS. Pri datotekah, ki jih odpremo na kaseti, pa teh sporočil ni; datoteko lahko napišemo preko podatkov, ki smo jih že shranili, zato moramo biti pri formiranju datotek na kaseti previdni.

Primeri:
```
1Ø OPEN 2,8,4, "DISK-OUTPUT, SEQ, W"
   (Odpre sekvenčno datoteko z imenom DISK-OUTPUT
    na disketi.)
1Ø OPEN 1,1,2, "TAPE-WRITE"
   (Ukaz za konec datoteke na kasetofonu.)
1Ø OPEN 5Ø,Ø     (Vhod za tastaturo.)
1Ø OPEN 12,3     (Izhod na ekran)
1Ø OPEN 13Ø,4    (Izhod za printer.)
1Ø OPEN 1,1,Ø,"NAME"  (Bere s kasetofona.)
1Ø OPEN 1,1,1,"NAME"  (Piše na kasetofonu.)
1Ø OPEN 1,2,Ø,CHR$(1Ø) (Odpre kanal na priključek
                        RS-232)
1Ø OPEN 1,4,Ø,"STRING" (Pošlje grafiko na printer.)
1Ø OPEN 1,4,7,"STRING" (Pošlje črke na printer.)
1Ø OPEN 1,5,7,"STRING" (Pošlje črke na printer preko
                        priključka 5.)
1Ø OPEN 1,8,15,"COMMAND" (Pošlje ukaz na disk.)
```

OR

Tip: Logični operator

Format: <operand> OR <operand>

Delovanje: Logični operator OR združuje dva izraza in ugotovi njegovo vrednost (pravilno/nepravilno). Če ga uporabljamo pri računanju, nam OR določi bit rezultat 1, če je le eden izmed operandov 1. To pomeni, da je rezultat dveh operandov vedno integer (glej AND).

Primeri:
```
1ØØ IF (AA=BB) OR (XX=2Ø) THEN ...
    (Uporaba OR za primerjavo dveh izrazov.)
KK% = 64 OR 32: PRINT KK%
96
```

```
                                64
                             OR 32
                   0000 0000 0100 0000
              OR   0000 0000 0010 0000
dvojiško      0000 0000 0110 0000
desetiško                       96
```

PEEK

Tip: Integer funkcija

Format: PEEK (<število>)

Delovanje: Funkcija PEEK priredi integer vrednosti med Ø in 255, ki jih prebere iz spominskih lokacij. Številka v oklepaju je spominska lokacija, to število mora biti med Ø in 65535. Če ni v tem območju, nam BASIC javi napako ?ILLEGAL QUANTITY.

Primeri:
```
1Ø PRINT PEEK (53280) AND 15
   (Vrednost barve ozadja ekrana.)
 5 A% = PEEK (45) + PEEK (46) * 256
   (Vrednost adrese tabele BASIC spremenljivk.)
```

POKE

Tip: Stavek

Format: POKE <lokacija>, <vrednost>

Delovanje: POKE stavek uporabljamo zato, da zapišemo eno bytno binarno vrednost na dano spominsko lokacijo ali input/output register. Številka lokacije je lahko rezultat aritemtičnega izraza, ki mora biti v območju med Ø in 65535. Binarna vrednost pa mora biti med Ø in 255. V primeru, če vrednosti niso v tem območju, javi ?ILLEGAL QUANTITY napako.
Stavek POKE in PEEK uporabljamo za shranjevanje podatkov, kontrolo grafičnega zapisa ali generatorja zvoka in polnjenje podprogramov v strojnem jeziku. Parametre operacijskega sistema lahko menjujemo, jih nadomeščamo ali preverjamo s pomočjo PEEK funkcije in POKE stavka.

Primeri:
```
POKE 1Ø24,1      (Postavi "A" na pozicijo 1 ekrana.)
POKE 2Ø4Ø, PTR   (Grafičnemu znaku določi kazalec.)
1Ø POKE RED,32
2Ø POKE 36879,8
3Ø POKE A,B
```

POS

Tip: Integer funkcija
Format: POS (<argument>)

Delovanje: Funkcija POS nam pove tekočo pozicijo kursorja (Ø - 79) v logični 8Ø znakovni vrstici. Ker ima COMMODORE 64 ekran s 4Ø stolpci, pomeni pozicija med 4Ø - 79 drugo vrstico logične vrstice. Argument funkcije ni pomemben.

Primeri: 1ØØØ IF POS(Ø) > 38 THEN PRINT CHR$(13)

PRINT

Tip: Stavek
Format: PRINT [<spremenljivka>] [<,/;> <spremenljivke>] ...

Delovanje: PRINT stavek uporabljamo za pisanje podatkov na ekran. Če zapišemo tudi CMD, PRINT stavek piše na priključek, ki ga določimo. Lista spremenljivk v PRINT stavku so lahko kakršnikoli izrazi. Če za PRINT ne napišemo ničesar, pomeni to prazno vrstico. Pozicija izpisa vsakega izraza je podana z določitvijo točke izpisa na izpisni listi. Izpis lahko določamo s praznim prostorom, vejico ali podpičjem. 8Ø-znakovna logična vrstica je razdeljena na 8 1Ø-znakovnih izpisov. Če ločimo izraze z vejico, to pomeni, da bo vsak naslednji izraz zapisan v naslednji 1Ø-znakovni oddelek. Če hočemo, da se izrazi izpisujejo neposredno drug za drugim, jih pišemo ločene s podpičji. Pri tem pravilu poznamo dve izjemi:

1) Pri izpisu numeričnega podatka izpiše za njim še prazen prostor.
2) Pozitivna števila izpiše s praznim prostorom pred številom.

Če postavimo vejico ali podpičje na konec izpisne liste, naslednji PRINT stavek v programu nadaljuje izpis v isti vrstici. Če ne napišemo ločila na

koncu PRINT stavka, BASIC zaključi izpis s carriage returnom in line feed; naslednji PRINT stavek piše v naslednji vrstici. Če je izpisna lista PRINT stavka daljša od 4Ø stolpcev, začne pisati v naslednjo vrstico.
V BASICU se PRINT stavek uporablja na mnogo različnih načinov pisanja na ekran.

Primeri: 1)
```
5 X = 5
1Ø PRINT -5 * X, X - 5, X + 5, X  5
-25          Ø          1Ø          3125
```

2)
```
5 X = 9
1Ø PRINT X; "NA KVADRAT JE "; X * X; "IN "
2Ø PRINT X; "NA TRETJO JE "; X  3

9 NA KVADRAT JE 81 IN 9 NA TRETJO JE 729
```

3)
```
9Ø AA$ = "ALFA":BB$ = "BETA": CC$ = "GAMA":
     DD$ = "DELTA": EE$ = "EPSILON"
1ØØ PRINT AA$BB$; CC$ DD$,EE$

ALFABETAGAMA DELTA          EPSILON
```

Pisanje v narekovajih

Ko enkrat napišemo narekovaj (SHIFT 2), tipke za kontrolo kursorja ne delujejo več; če nanje pritisnemo , izpišejo obratne znake, ki stojijo za njimi. Na ta način lahko uporabimo kontrolne tipke kursorja tudi v programu, če jih zapišemo v PRINT stavku. Edino INST/DEL tipka deluje normalno, tudi če jo pritisnemo za narekovajem.

1. Premik kursorja

Kontrolne tipke kursorja, ki jih lahko vstavimo v program s PRINT stavkom so:

TIPKA		IZPIS V NAREKOVAJIH
	CLR/HOME	S
SHIFT	CLR/HOME	▽
	↑ CRSR ↓	Q
SHIFT	↑ CRSR ↓	▢

TIPKA	IZPIS V NAREKOVAJIH
⇦CRSR⇨	]
SHIFT ⇦CRSR⇨	[

Če hočemo zapisati besedo HELLO diagonalno na ekran, uporabimo naslednji stavek:

PRINT " CLR/HOME H ↑CRSR↓ E ↑CRSR↓ L ↑CRSR↓ L ↑CRSR↓ O"

Ta stavek se izpiše na ekran kot:

PRINT " S H Q E Q L Q L Q O"

2. Obratni znaki

Če pritisnemo v narekovajih na tipki CTRL in 9 hkrati, se izpiše znak R, to pomeni, da se bodo vsi naslednji znaki izpisali v inverzni obliki (podobno kot negativ na sliki). Če želimo spet nazaj v normalen izpis, pritisnemo na CTRL Ø, ki izpiše znak ▭ ali pa pritisnemo na RETURN tipko. Tudi, če končamo PRINT stavek brez vejice ali podpičja, naslednji PRINT stavki spet pišejo v normalni obliki.

3. Kontrolne tipke za barve

Če uporabimo CTRL ali C= tipko s katerokoli izmed 8 tipk, ki določajo barve, v narekovajih, se izpišejo nekateri obratni znaki. V PRINT stavku lahko z njimi spreminjamo barvo izpisa.

TIPKA	BARVA	IZPIS V NAREKOVAJIH
CTRL 1	črna	▢
CTRL 2	bela	E
CTRL 3	rdeča	E
CTRL 4	modro-zelena	◹
CTRL 5	vijolična	▯
CTRL 6	zelena	↑
CTRL 7	modra	←
CTRL 8	rumena	▯
C= 1	oranžna	▣
C= 2	rjava	◪

TIPKA	BARVA	IZPIS V NAREKOVAJIH
C= 3	svetlo rdeča	
C= 4	siva 1	
C= 5	siva 2	
C= 6	svetlo zelena	
C= 7	svetlo modra	
C= 8	siva 3	

Če želimo izpisati besedo COMMODORE v rdečem in 64 v belem, uporabimo:

PRINT "CTRL 3 COMMODORE CTRL 2 64"

PRINT#

Tip: Input/Output stavek

Format: PRINT# <številka datoteke> [<spremenljivka>]
[<, / ;> < spremenljivke>] ...

Delovanje: Stavek PRINT# uporabljamo za zapis podatkov v logično datoteko. Uporabiti moramo isto številko, ki smo jo zapisali pri odpiranju datoteke. Izrazi, ki jih zapisujemo so lahko kakršnegakoli tipa. Ločila med podatki se uporabljajo enako kot pri PRINT stavku razen v dveh primerih. Če uporabljamo PRINT# z datotekami na traku, ima vejica isti učinek kot podpičje. Tudi če ne uporabljamo nobenih ločil med podatki, si sledijo v istem vrstnem redu; numerični podatki se zapisujejo s spaceom na koncu, in če so pozitivni tudi na začetku. Če nobeno ločilo ne zaključi liste podatkov, se zaključi sama s carriage return in line feed. Najboljši in najlažji način, da napišemo več spremenljivk na datoteko na trak ali disk je ta, da stringu priredimo CHR$(13) (RETURN) in uporabljamo ta string med spremenljivkami, ki jih pišemo na datoteko.

Primeri:1)
```
1Ø OPEN 1,1,1,"TAPE FILE"
2Ø R$ = CHR$(13)
```

```
3Ø PRINT  1,1;R$;2;R$;3;R$;4;R$;5
4Ø PRINT  1,6
5Ø PRINT  1,7
   (Če pišemo namesto CHR$(13)  CHR$(44), bo med
    spremenljivkami vejica, če pa pišemo CHR$(59)
    bo med spremenljivkami podpičje.)
2) 1Ø CO$ = CHR$(44) : CR$ = CHR$(13)
   2Ø PRINT  1, "AAA" CO$ "BBB" , "CCC"; "DDD"; "EEE"
      CR$ "FFF" CR$:
   3Ø INPUT# 1,A$,BCDE$, F$

   AAA,BBB              CCCDDDEEE
   (carriage return)
   FFF (carriage return)
3)  5 CR$ = CHR$(13)
   1Ø PRINT# 2, "AAA"; CR$; "BBB"
   2Ø PRINT# 2, "CCC";
   3Ø INPUT# 2, A$, B$, DUM$, C$

   (1Ø praznih mest)   AAA
   BBB
   (1Ø praznih mest)   CCC
```

READ

Tip: Stavek

Format: READ <spremenljivka> [, <spremenljivka>]...

Delovanje: READ stavek uporabljamo, da v programu dodeli konstante iz DATA stavka spremenljivkam. Paziti moramo, da se tipi spremenljivk ujemajo s podatki, sicer javi ?SINTAX ERROR. Konstante v DATA stavkih ločimo z vejicami. Z enim samim READ stavkom lahko preberemo več DATA stavkov, ali pa se lahko več READ stavkov nanaša ne en sam DATA stavek v programu. Če hočemo prebrati več podatkov kot jih je v DATA stavkih, BASIC javi ?OUT OF DATA ERROR. Če pa je podatkov več, kot jih zahteva READ stavek, bo naslednji READ stavek v

programu začel brati podatke tam, kjer se je branje končalo (glej RESTORE).

POZOR! Napaka ?SINTAX ERROR se pojavi v vrstici, kjer je DATA stavek in ne v READ stavku.

Primeri:

```
11Ø READ A,B,C$
12Ø DATA 1,2,HELLO

1ØØ FOR X = 1 TO 1Ø : READ A(X): NEXT
 :
2ØØ DATA 3.Ø8, 5.19, 3.12,3.98,4.24
21Ø DATA 5.Ø8, 5.55, 4.ØØ,3.16,3.37

1 READ MESTO$, DRZAVA$, PST
:
5 DATA LJUBLJANA, JUGOSLAVIJA, 61ØØØ
```

REM

Tip: Stavek

Format: REM [<opomba>]

Delovanje: Stavek REM uporabljamo za večje razumevanje programa med listanjem. Če pišemo dolg program, je dobro, da si vsake toliko časa zabeležimo v programu opombo, ki nam pove, kaj program v nekem delu izvaja. Za REM lahko pišemo katerikoli izraz, vključno z dvopičjem in BASIC instrukcijami. Stavek REM namreč povzroči, da BASIC ne prebere ničesar, kar je zapisano v isti vrstici. Ti stavki se izpisujejo edino pri listanju programa.

Primeri:

```
1Ø REM IZRACUN POVPRECNE VREDNOSTI
2Ø FOR X = 1 TO 2Ø : REM ZANKA ZA 2Ø VREDNOSTI
3Ø SUM = SUM + V(X) : NEXT
4Ø A = SUM/2Ø
```

RESTORE

Tip: Stavek

Format: RESTORE

Delovanje: BASIC vsebuje notranji kazalec za naslednji podatek, ki ga beremo z READ v DATA stavkih. Ta kazalec lahko prenesemo na prvi DATA stavek v programu, če uporabimo RESTORE. RESTORE lahko zapišemo kjerkoli v programu pred ponovnim branjem podatkov.

Primeri:

```
1ØØ FOR X = 1 TO 1Ø: READ A(X): NEXT
2ØØ RESTORE
3ØØ FOR Y = 1 TO 1Ø : READ B(Y): NEXT
 :
4ØØØ DATA 3.Ø8,5.19,3.12,3.98,4.24
41ØØ DATA 5.Ø8,5.55,4.ØØ,3.16,3.37
```

(Program napolni dve polji z identičnimi podatki.)

```
1Ø DATA 1,2,3,4
2Ø DATA 5,6,7,8
3Ø FOR L = 1 TO 8
4Ø READ A: PRINT A
5Ø NEXT
6Ø RESTORE
7Ø FOR L = 1 TO 8
8Ø READ A: PRINT A
9Ø NEXT
```

RETURN

Tip: Stavek

Format: RETURN

Delovanje: S stavkom RETURN zaključimo podprogram, ki smo ga poklicali z GOSUB. To pomeni, da se program nadaljuje na naslednjem programskem stavku za GOSUB. Če uporabljamo podprogram v podprogramu, moramo paziti, da ima vsak GOSUB vsaj en RETURN stavek.

Primer:
```
1Ø PRINT "TO JE PROGRAM"
2Ø GOSUB 1ØØØ
3Ø PRINT "PROGRAM SE NADALJUJE"
4Ø GOSUB 1ØØØ
5Ø PRINT "SE SMO V PROGRAMU"
6Ø END
1ØØØ PRINT "TO JE PODPROGRAM": RETURN
```

RIGHT$

Tip: String funkcije
Format: RIGHT$(<string> , <število>)

Delovanje: Funkcija RIGHT$ določi string, ki ga vzame iz desnega dela string argumenta. Dolžina tega stringa je definirana s številom argumenta in je lahko integer vrednost med Ø in 255. Če je vrednost Ø, potem je rezultat prazen string; če pa je ta vrednost večja od dolžine string argumenta, RIGHT$ postane celoten string argument.

Primeri:
```
1Ø MS6$ = "COMMODORE COMPUTERS"
2Ø PRINT RIGHT$ (MS6$,9)

RUN
COMPUTERS
```

RND

Tip: Floating-point funkcija
Format: RND(<število>)

Delovanje: RND funkcijo imenujemo tudi generator naključnih števil (random). Določi nam naključno vrednost med Ø.Ø in 1.Ø, tako da kreira začetno vrednost niza naključnih števil. Argument je lahko po vrednosti katerokoli število, pomemben je le njegov predznak. Če je argument pozitivno število, lahko pokličemo nazaj niz naključnih števil z isto začetno vrednostjo.

Če izberemo za argument ničlo, nam RND določi naključno vrednost direktno iz sistemske vgrajene ure. Negativni argument pa povzroči, da funkcija določi z vsakim izvajanjem novo začetno vrednost.

Primeri:
```
1ØØ PRINT INT (RND(Ø) * 5Ø)
```
(Izračuna nam naključne integer vrednosti med 1 in 49.)
```
1ØØ X = INT (RND(1) * 6 + INT (RND(1) * 6 + 2
```
(Simulira dve kocki.)
```
1ØØ X= INT (RND(1) * 1ØØØ) + 2
```
(Naključna integer števila med 1 - 1ØØØ.)
```
1ØØ X = INT (RND(1) * 15Ø) + 1ØØ
```
(Naključna števila od 1ØØ - 249.)
```
1ØØ X = RND(1) * (U - L) + L
```
(Naključna števila med zgornjo (U) in spodnjo (L) mejo.)

RUN

Tip: Ukaz

Format: RUN [<številka vrstice>]

Delovanje: S sistemskim ukazom RUN se program, ki ga imamo zapisanega v spominu računalnika začne izvajati. Pred tem pa izvede tudi operacijo čiščenja spomina(CLR). Če se hočemo temu izogniti, lahko uporabimo CONT ali GOTO za izvedbo programa. Če je številka vrstice za RUN določena, se začne program izvajati s to programsko vrstico, v vsakem drugem primeru pa RUN začne s prvo programsko vrstico. Ukaz RUN lahko uporabimo tudi v programu samem. Izvajanje programa se ustavi in BASIC se vrne v direkten način, ko pridemo do STOP ali END stavka, ali pa ko se izvrši zadnja programska vrstica.

Primeri:
```
RUN       (Začne izvajati program v prvi vrstici.)
RUN 5ØØ   (Začne izvajati program v vrstici 5ØØ.)
```

RUN X (Začne v vrstici X, če pa le-te ni v programu, se izpiše UNDEF'D STATEMENT napaka.)

SAVE

Tip: Ukaz

Format: SAVE ["<ime datoteke>"][,<številka priključka>] [,<adresa>]

Delovanje: Z ukazom SAVE shranimo program, ki je trenutno v spominu računalnika na kaseto ali disketo. S tem ukazom shranjujemo le programske datoteke. Če izpustimo številko priključka, CBM 64 avtomatično shrani program na kaseto (1). Če je številka priključka 8, nam izpiše program na disketo. SAVE stavek lahko uporabimo tudi v programu samem.
Programi na trakovih se avtomatično shranjujejo dvakrat, tako da se izognemo napakam pri branju programa iz kasete. Kadar shranjujemo program na trak, lahko izpustimo ime programa, kakor tudi adreso. Vendar pa je lažje najti na traku program, ki smo mu dodelili ime, kajti če program nima imena, ko ga shranjujemo, ga ne moremo priklicati z LOAD stavkom in imenom.
Če je številka adrese 1, to povzroči, da se napolni program na začetno adreso, namesto na lokacijo 2Ø48, kot je normalno. Številka adrese 2 nam označi konec traku po shranjevanju programa, medtem ko je 3 kombinacija obeh funkcij.
Ko shranjujemo programe na disketo, moramo obvezno pisati ime programa za SAVE.

Primeri:

```
SAVE              (Piše na trak brez imena.)
SAVE "ALFA",1     (Shrani na trak program z imenom ALFA.)
SAVE"ALFA",1,2    (Shrani na trak program ALFA in določi
                   konec traku.)
SAVE"DISK",8      (Shrani program na disketo.)
SAVE A$           (Shrani na trak program z imenom A$.)
1Ø SAVE "HI"      (Shrani na trak HI in nadaljuje z
```

naslednjo programsko vrstico.)
SAVE "ME",1,3 (Shrani na isto spominsko lokacijo
in označi konec traku.)

SGN

Tip: Integer funkcija
Format: SGN (<število>)

Delovanje: Funkcija SGN izračuna integer vrednost odvisno od predznaka argumenta. Če je le-ta pozitiven, je rezultat 1, če je Ø, je Ø, če pa je negativen, je rezultat - 1.

Primer: 9Ø ON SGN (DV) + 2 GOTO 1ØØ, 2ØØ, 3ØØ
(Preskoči na 1ØØ, če je DV negativen, na 2ØØ, če je DV = Ø in na 3ØØ, če je DV pozitiven.)

SPC

Tip: String funkcija
Format: SPC (<število>)

Delovanje: Funkcijo SPC uporabljamo za formatiranje podatkov na ekranu ali v logični vrstici. Izpiše se število space znakov, ki so dani z argumentom v oklepaju. Za ekran in datoteke na traku mora biti vrednost argumenta med Ø in 255, medtem ko za disk samo do 254.

Primeri:
```
1Ø PRINT "OD TU"
2Ø PRINT SPC(5) "IN" SPC(14) "TAM"

RUN
OD TU     IN              TAM
```

SQR

Tip: Floating-point funkcija
Format: SQR (<število>)

Delovanje: SQR nam izračuna vrednost kvadratnega korena argumenta. Vrednost argumenta ne sme biti negativna,

sicer BASIC javi ?ILLEGAL QUANTITY napako.

Primeri: FOR J = 2 TO 5: PRINT J * 5, SQR(J * 5): NEXT

```
1Ø          3.16227766
15          3.87298335
2Ø          4.47213595
25          5
```

STATUS

Tip: Integer funkcija

Format: STATUS

Delovanje: Izpiše nam status zadnjih input/output operacij, ki smo jih izvršili na odprti datoteki. STATUS lahko preberemo iz kateregakoli zunanjega priključka. Beseda STATUS je sistem definiranih spremenljivk, v katere nam KERNAL izpiše input/output operacije. Tabela STATUS kod za trak, printer, disk in priključek RS-232 je naslednja:

ST bit pozicija	ST numerična vrednost	Branje s kasete	Serijski R/W	VERIFY + LOAD
Ø	1		ne piše	
1	2		ne bere	
2	4	kratki blok		kratki blok
3	8	dolgi blok		dolgi blok
4	16	neodpravlj. napake v branju		MISMATCH napaka
5	32	napaka pri preverjanju		napaka pri preverjanju
6	64	konec datoteke	input/ output napaka	
7	- 128	konec traku	ni priključ.	konec traku

Primeri:
```
1Ø OPEN 1,4: OPEN 2,8,4, "MASTER FILE, SEQ, W"
2Ø GOSUB 1ØØ: REM PREVERI STATUS
3Ø INPUT# 2, A$, B, C
4Ø IF STATUS AND 64 THEN 8Ø : REM KONEC DATOTEKE
5Ø GOSUB 1ØØ : REM PREVERI STATUS
6Ø PRINT# 1, A$, B; C
7Ø GOTO 2Ø
8Ø CLOSE 1: CLOSE 2
9Ø GOSUB 1ØØ : END
1ØØ IF ST > Ø THEN 9ØØØ: REM INPUT/OUTPUT ERROR
11Ø RETURN
```

STEP

Tip: Stavek

Format: [STEP < izraz >]

Delovanje: STEP mora biti del FOR ... NEXT zanke, če je korak v zanki različen od + 1. Za korak v zanki lahko uporabimo katerokoli realno število; če uporabimo Ø, bo to neskončna zanka.

POZOR! Vrednosti koraka ne moremo spremeniti, ko je program še v zanki.

Primeri:
```
25 FOR XX = Z TO 2Ø STEP 2
   (Zanka se ponovi desetkrat.)
35 FOR ZZ = Ø TO -2Ø
```

STOP

Tip: Stavek

Format: STOP

Delovanje: Stavek STOP uporabljamo za ustavitev izvajanja programa. Isti učinek je, če med izvajanjem programa pritisnemo na tipko RUN/STOP. Izpiše se ?BREAK IN LINE XXX, ki mu sledi READY. XXX je številka vrstice, kjer smo zapisali stavek STOP. Vse odprte datoteke ostanejo odprte, tudi vrednosti spremenljivk se ne spremenijo. Program lahko spet startamo s CONT ali

GOTO stavkom.

Primeri:
```
1Ø INPUT# 1,AA,BB,CC
2Ø IF AA = BB AND BB = CC THEN STOP
3Ø STOP
```

Če je spremenljivka AA enaka BB in CC potem izpiše:

BREAK IN LINE 2Ø

BREAK IN LINE 3Ø izpiše za katerikoli drugi podatek.

STR$

Tip: String funkcija
Format: STR$ (<število>)

Delovanje: S STR$ funkcijo lahko priredimo string numerični vrednosti argumenta.

Primeri:
```
1ØØ FLT = 1.5E4: ALFA$ = STR$(FLT)
11Ø PRINT FLT, ALFA$
 15ØØØ          15ØØØ
```

SYS

Tip: Stavek
Format: SYS <spominska lokacija>

Delovanje: Najenostavnejši način, da uporabimo strojni program v BASIC programu je s stavkom SYS. Strojni program se začne na spominski lokaciji, ki jo zapišemo za SYS. Sistemski ukaz SYS uporabimo lahko direktno v programu, da zamenjamo strojni program v spominu s kontrolo mikroprocesorja. Dana spominska lokacija je zapisana v numeričnem izrazu in je lahko kjerkoli v spominu, RAM-u ali ROM-u. Kadar uporabljamo SYS stavek v programu, moramo strojni program zaključiti z RTS (ReTurn from Subroutine) instrukcijo, tako da lahko nadaljujemo BASIC program, ko se strojni program konča.

Primeri: SYS 64738 (Skoči na sistemski topli start v ROM-u.)
1Ø POKE 44ØØ,96 : SYS 44ØØ
(Skoči na strojno lokacijo 44ØØ in se takoj vrne.)

TAB

Tip: String funkcija
Format: TAB (<število>)

Delovanje: Funkcija TAB premakne kursor na relativno pozicijo na ekranu, dano s številko argumenta, če začnemo na levi strani tekoče vrstice. Vrednost argumenta je lahko Ø do 255. Funkcija TAB uporabljamo le s PRINT stavkom, s PRINT# stavkom nima nobenega efekta.

Primeri:
```
1ØØ PRINT "IME" TAB(25)"VSOTA": PRINT
11Ø INPUT# 1, IM$, VS$
12Ø PRINT IM$ TAB(25) VS$

IME                       VSOTA
A.KOVAC                   25
```

TAN

Tip: Floating-point funkcija
Format: TAN (<število>)

Delovanje: Izračuna tangens argumenta v radianih. Če je funkcijska vrednost prevelika, nam BASIC izpiše ?DIVISION BY ZERO ERROR.

Primeri:
```
1Ø XX = .785398163: YY = TAN(XX): PRINT YY
1
```

TIME

Tip: Numerična funkcija
Format: TI

Delovanje: Funkcija TI nam prebere intervalni čas, to je čas,

ki ga imenujemo čas sistemske ure ali "jiffy clock". Uro inicializiramo (priredimo ji vrednost Ø), ko vklopimo računalnik, z delovanjem računalnika pa se njena vrednost povečuje.

Primeri:
```
1Ø PRINT TI/6Ø "SEKUND JE MINILO OD VKLOPA
   RACUNALNIKA"
```

TIME$

Tip: String funkcija
Format: TI$

Delovanje: Funkcija TI$ deluje kot prava ura, dokler je sistem vključen. Za določitev tega časa uporabimo "jiffy clock", TI$ pa nam izpiše string šestih znakov v urah, minutah in sekundah. Tudi TI$ lahko priredimo določeno vrednost, potem pa nam izpisuje natančen čas.

Primeri:
```
1 TI$ = "ØØØØØØ" : FOR J = 1 TO 1ØØØØ : NEXT : PRINT TI$
ØØØØ11
```

USR

Tip: Floating-point funkcija
Format: USR (<število>)

Delovanje: S funkcijo USR lahko skočimo iz BASIC programa na podprogram v strojnem jeziku (User callable machine language SubRoutine), ki ima startne adrese na spominskih lokacijah 785-786. Začetno adreso moramo določiti pred uporabo USR funkcije s POKE stavkom za lokacije 785-786. Če tega ne naredimo, se izpiše ?ILLEGAL QUANTITY napaka.
Vrednost številskega argumenta je shranjena v floating-point akomulatorju, če začnemo na lokaciji 97. Rezultat USR funkcije pa je vrednost, ki se konča tam, kjer podprogram spet preskoči v BASIC.

Primeri:
```
1Ø B = T * SIN(Y)
2Ø C = USR(B/2)
3Ø D = USR(B/3)
```

VAL

Tip: Numerična funkcija

Format: VAL (<string>)

Delovanje: VAL funkcija nam izpiše numerično vrednost string argumenta. Če prvi znak v stringu ni + ali - ali številka, potem je funkcijska vrednost Ø. VAL funkcija bere vrednosti stringa, dokler ni končan ali pa dokler ne naleti na prvi znak, ki ni številka (razen decimalne pike in črke E za potenco).

Primeri:
```
1Ø INPUT# 1, IM$, PT$
2Ø IF VAL (PT$) = 61ØØØ THEN PRINT IM$ TAB(25) "JE IZ
   LJUBLJANE"
```

VERIFY

Tip: Ukaz

Format: VERIFY ["<ime datoteke>"][, <priključek>]

Delovanje: Ukaz VERIFY uporabljamo v direktnem ali programskem načinu zato, da primerjamo BASIC program na traku ali disku s programom v spominu računalnika. Po navadi VERIFY uporabljamo po ukazu SAVE, da se prepričamo, če smo program pravilno shranili na disketi ali kaseti.

Če izpustimo številko priključka, program predpostavlja, da preverjamo zapis na traku (1). Če izpustimo ime programa, potem preveri prvi program, ki ga najde na traku po ukazu VERIFY, za programe na disketi pa velja, da je potrebno zapisati za VERIFY vse programe in številko priključka 8. Če VERIFY ugotovi razliko med programoma, ki ju primerjamo, se izpiše ?VERIFY ERROR.

Ime programa lahko zapišemo v narekovajih ali pa kot

string spremenljivke. VERIFY uporabljamo tudi za to, da na traku označimo mesto, kjer bomo zapisali nov program, tako da ne pišemo preko zapisa na traku.

Primeri:
```
VERIFY          (Preveri prvi program na traku.)
PRESS PLAY ON TAPE
OK
SEARCHING
FOUND <FILE NAME>
VERIFYING

9ØØØ SAVE "ME",8:
9Ø1Ø VERIFY "ME",8
(Preveri zapis na disketi.)
```

WAIT

Tip: Stavek

Format: WAIT <lokacija>, <maska -1>[, <maska -2>]

Delovanje: WAIT stavek povzroči, da se izvajanje programa zaustavi, dokler dana spominska adresa ne prepozna določene bit kombinacije. Z WAIT lahko ustavimo program, dokler se ne zgodi nekaj zunaj spomina računalnika. Podatki, ki jih uporabljamo v WAIT stavku, so lahko katerikoli numerični izrazi, toda vedno jih pretvori v integer vrednosti.
Večina programerjev tega stavka nikoli ne uporablja. WAIT stavek ustavi program, dokler se bit na specifični lokaciji ne spremeni na specifičen način. Uporaben je samo za nekatere input/output operacije.

Primeri:
```
WAIT 1,32,32
```
Počaka, dokler ne pritisnemo tipke na kasetofonu, potem pa nadaljuje program.

SIN

Tip: Floating-point funkcija

Format: SIN (< število >)

Delovanje: Funkcija SIN nam izračuna sinus argumenta v radianih.

Primeri:

```
AA = SIN (1.5) : PRINT AA
 .997494987
```

10. TASTATURA RAČUNALNIKA CBM 64

Operacijski sistem ima 10-znakovni vmesnik tastature, ki se uporablja za shranjevanje znakov, ki prihajajo preko tipkovnice v računalnik, preden jih sprejme procesor. To nam omogoča hitrejši dotok podatkov in lepši zapis. Ta vmesnik nam omogoča tudi odgovore na INPUT in GET stavke v programu; pomeni pa tudi to, da lahko popravljamo napačno vnešene znake. Po navadi uporabljamo za popravljanje tipko [← CRSR] ali [INST/DEL], ter nato ponovno vtipkamo znak.

Tastatura je pravzaprav množica preklopov, ki jih lahko zapišemo v matriki z 8 stolpci in 8 vrsticami. Biti od Ø do 7 spominske lokacije 56320 ustrezajo stolpcem, medtem ko biti od Ø do 7 spominske lokacije 56321 ustrezajo vrsticam. Matrika nam omogoča 64 vrednosti. Če pa uporabljamo tipko [RVS], [CTRL] ali [C=] ali držimo [SHIFT] tipko ter vtipkamo drugi znak, pridobimo še dodatne vrednosti (obratne ali inverzne vrednosti).

11. SCREEN EDITOR

Screen Editor nam omogoča lažje popravljanje in preverjanje programskega teksta. Ko izlistamo del teksta na ekran, uporabimo lahko tipke za kontrolo kursorja, da se pomikamo po ekranu in popravljamo posamezne vrstice ali zamenjamo znake. Ko popravimo določeno vrstico in pritisnemo na tipko [RETURN] kjerkoli v vrstici, Screen Editor prebere celotno popravljeno 80-znakovno logično vrstico. Tekst nato pošlje BASIC interpreterju, da ga shrani v programu. Če želimo kopirati programske vrstice, lahko to storimo preprosto tako, da spremenimo številko vrstice in pritisnemo na [RETURN].

Če uporabljamo skrajšan zapis BASIC instrukcij, lahko to pomeni, da programska vrstica preseže 80-znakov pri listanju. Znaki, ki jih piše preko meje logične vrstice, so pri popravljanju vrstice izgubljeni, ker bo Screen Editor prebral le dve vrstici. Zato tudi INPUT stavka ne moremo uporabljati za več kot 80-znakov.

ASCII - TABELA

American Standard Code for Information Interchange

Znak	Izpis	Znak	Izpis
0		31	BLU
1		32	SPACE
2		33	!
3		34	"
4		35	#
5	WHT	36	$
6		37	%
7		38	&
8	SHIFT C= OFF	39	'
9	SHIFT C= ON	40	(
10		41	)
11		42	*
12		43	+
13	RETURN	44	,
14	Preklop na	45	-
	male črke	46	.
15		47	/
16		48	Ø
17	↑CRSR	49	1
18	RVS ON	50	2
19	CLR/HOME	51	3
20	INST/DEL	52	4
21		53	5
22		54	6
23		55	7
24		56	8
25		57	9
26		58	:
27		59	;
28	RED	60	←
29	CRSR ⇒	61	=
30	GRN	62	→

Znak	Izpis	Znak	Izpis
63	?	98	
64	@	99	
65	A	100	
66	B	101	
67	C	102	
68	D	103	
69	E	104	
70	F	105	
71	G	106	
72	H	107	
73	I	108	
74	J	109	
75	K	110	
76	L	111	
77	M	112	
78	N	113	
79	O	114	
80	P	115	
81	Q	116	
82	R	117	
83	S	118	
84	T	119	
85	U	120	
86	V	121	
87	W	122	
88	X	123	
89	Y	124	
90	Z	125	
91	[	126	
92	£	127	
93	]	128	
94	↑	129	oranžna
95	←	130	
96		131	
97		132	

Znak	Izpis	Znak	Izpis
133	f1	163	
134	f3	164	
135	f5	165	
136	f7	166	
137	f2	167	
138	f4	168	
139	f6	169	
140	f8	170	
141	SHIFT RETURN	171	
142	Preklop na	172	
	velike črke	173	
143		174	
144	BLK	175	
145	CRSR ⇩	176	
146	RVS OFF	177	
147	CLR/HOME	178	
148	INST/DEL	179	
149	rjava	180	
150	svetlo rdeča	181	
151	siva 1	182	
152	siva 2	183	
153	svetlo zelena	184	
154	svetlo modra	185	
155	siva 3	186	
156	PUR	187	
157	⇦ CRSR	188	
158	YEL	189	
159	CYN	190	
160	SPACE	191	
161			
162			

GRAFIKA V SIMON`S BASICU

1. U V O D

V tem poglavju si bomo ogledali grafične ukaze Simon`s Basica. Najprej si bomo ogledali zgradbo ekrana in pojasnili razliko med High-resolution grafiko in Multi-colour grafiko. Ker so možnosti za uporabo barv enake kot pri normalnem Basicu se ob njih ne bomo posebej ustavljali, ampak se bomo teh možnosti samo ponovno spomnili. Dalje bomo v tem poglavju spoznali vse grafične ukaze, katerih delovanje si bomo ogledali na nekaterih primerih.

2. ZGRADBA EKRANA

Kadar delamo z ekransko grafiko, se ekran C64 razdeli v matrico velikosti 320 X 200 točk. Vsaka točka ima X in Y koordinato. Levi zgornji vogal ekrana ima koordinate 0,0, desni spodnji vogal pa ima koordinate 200,320. High-resolution grafika razdeli ekran na matrico 320 X 200 točk, Multi-colour grafika pa ekran razdeli na matrico 160 X 200 točk. Vidimo torej, da je razmak med dvema točkama v X- smeri v multi-colour grafiki dvakrat večji kot v high-resolution grafiki.

3. BARVE NA COMMODORE 64

Kot smo že spoznali v prejšnjih poglavjih je možno na CBM64 definirati 16 različnih barv. V matrici 8 x 8 točk, lahko izberemo do tri različne barve. Barve imajo v Simon`s Basicu enake šifre kot v standardnem Basicu in sicer:

0 - črna	8 - oranžna
1 - bela	9 - svetlo oranžna
2 - rdeča	10 - svetlo rdeča
3 - modro zelena	11 - siva 1
4 - vijoličasta	12 - siva 2
5 - zelena	13 - svetlo zelena
6 - modra	14 - svetlo modra
7 - rumena	15 - siva 3

./.

4. TIP TOČKE

Vsi grafični ukazi v Simon`s Basicu imajo skupno karakteristiko Tip točke, ki bo v vseh ukazih označena s črko "T". Tip točke nam pove na kakšen način bo neka točka prikazana na ekranu. Tip točke je označen s celo enomestno številko, ki ima lahko naslednje vrednosti:

A) HIGH-RESOLUTION MODE

tip	pomen (funkcija)
0	briše točko (na ekranu ne bo prikazana)
1	prikaže točko
2	invertira točko (če točka na ekranu obstaja jo briše, če ne obstaja jo prikaže)

B) MULTI-COLOUR MODE

0	briše točko (na ekranu ne bo prikazana)
1	prikaže točko v barvi 1 (MULTI/LOW COL)
2	" " " 2 "
3	" " " 3 "
4	invertira točko - točka v barvi ozadja bo prikazana v barvi 3 - točka v barvi 1 bo prikazana v barvi 2 - točka v barvi 2 bo prikazana v barvi 1 - točka v barvi 3 bo prikazana v barvi ozadja

5. GRAFIČNI UKAZI

5.1. HIRES

Format: HIRES BZ,BO

Namen: Določevanje barve znakov in ozadja

Delovanje: Ukaz HIRES vključi grafiko visoke ločljivosti. Vsaka točka se lahko posebej vnese v matrico 320 x 200 točk. Spremenljivka BZ določa barvo točke, spremenljivka BO pa barvo ozadja.

Primer: HIRES 0,1

5.2. NRM

Format NRM

Namen: Povratek na normalni ekran

Delovanje: S tem ukazom se izključi grafika visoke ločljivosti in se vrnemo na normalni ekran 40 x 25 znakov.

Primer: NRM

./.

5.3. REC

Format: REC X, Y, X1, Y1, T

Namen: Risanje pravokotnikov

Delovanje: Ukaz REC riše pravokotnik na ekranu. Prvi dve spremeljivki X in Y odrejata koordinate levega vogala pravokotnika (gornjega). Spremenljivki X1 in Y1 pa določata dolžino stranic.

Primer:
```
100 HIRES 2,1
110 FOR X = 5 TO 125 STEP 5
120 : REC X,1.3*X, 320-5/2*X, 120-9*X/13,1
130 NEXT X
1000 GO TO 1000
```

5.4. MULTI

Format: HIRES BZ,BO: MULTI C1,C2,C3

Namen: Vključitev MULTI COLOUR moda in določevanje treh barv za znake

Delovanje: Multi colour grafika se vključi, če ukazu HIRES sledi ukaz MULTI. V tem primeru ima točka na ekranu dvojno širino v primerjavi z točko v HIRES grafiki. Matrica ekrana ima sedaj velikost (160 x 200 točk). Spremeljivke C1, C2, C3 določajo tri barve, katere želimo pridružiti znakom. Te barve določamo s pomočjo tipa točke za posamezen ukaz.

Primer:
```
100 HIRES 1,11: MULTI 12,8,6
110 FOR X=5 TO 60 STEP 1
120: F= F+2
130: IF F=5 THEN F=0
140: REC X,1.9*X, 169-5/2*X, 120-9*X/13,F
150: NEXT X
1000 GO TO 1000
```

5.5. LOW COL

Format: LOW COL C1, C2, C3

Namen: Menjava barv

Delovanje: Ukaz LOW COL nam omogoča uporabo naslednjih treh barv. Spremenljivke C1, C2, C3 določajo te barve enako kot pri ukazu MULTI.

OPOZORILO: Vedno moramo navesti vse tri spremenljivke!

./.

Primer:
```
100 HIRES 0,2: MULTI 3, 4, 9
110 FOR X=5 TO 53 STEP 1
120: F=F+1
130: LOW COL F,F+2, F+4
140: IF F=5 THEN F=0
150: REC X+X*1.5 X+2*X, 120-2.0*X, 190-3.5*X,F
160 NEXT X
1000 GO TO 1000
```

5.6. HI COL

Format: HI COL

Namen: Menjava barv

Delovanje: S tem ukazom ponovno vrnemo barve, ki so bile postavljene z ukazom MULTI.

Primer:
```
100 z=5
110 HIRES 0,2 : MULTI 10, 6, 8
120 REPEAT
130: FOR X=1 TO 3
140: REC Z,Z,Z,Z,X : Z=Z+31
150: LOW COL 1, 12, 0
160: REC Z,Z,Z,Z,X : Z=Z-21
170: HI COL
180: NEXT X : Z=Z-29
190  UNTIL Z>14
200 GO TO 200
```

5.7. PLOT

Format: PLOT X,Y,T

Namen: Narisati točko na ekranu

Delovanje: Z ukazom PLOT narišemo točko na ekranu. Koordinate točke določimo s spremenljivkami X in Y. Spremenljivka T pa določa barvo točke.

Primer:
```
100 HIRES 0,1
110 FOR X=0 TO 320 STEP 0.5
120: Y=100+SIN (X/34)*90
130: PLOT X,Y,1
140: PLOT X,100,1
150 NEXT X
160 GO TO 160
```

5.8. TEST

Format: SPREMENLJIVKA=TEST (X,Y)

Namen: Ugotoviti ali se neka točka nahaja na ekranu

Delovanje: Z ukazom TEST se informiramo ali se neka točka nahaja na ekranu. Spremenljivki X in Y določata koordinati test-točke na ekranu. Če na tem mestu točka obstaja dobimo kot vrednost spremenljivke 1, če točke ni je vrednost spremenljivke 0.

./.

Primer:

```
100 I=1
110 PRINT " "
120 HIRES 0,1
130 FOR X=0 TO 320 STEP 0.5
140: Y=100-COS (X/33)*90
150: PLOT X,Y,1
160: S=TEST (X,100)
170: IF S=1 THEN GOSUB 1000
180: PLOT X,101,1
190 NEXT X
200 PAUSE 30
210 END
1000 REM +++++++ PODPROGRAM +++++++
1010 N(I)=INT (X)
1020 IF N(I)=N(I-1) THEN RETURN
1030 PRINT " TOČKA NE OBSTAJA NA X="; N(I)
1040 I=I+1
1050 RETURN
```

5.9. LINE

Format: LINE X,Y,X1, Y1,T

Namen: Risanje črte

Delovanje: Z ukazom LINE rišemo črte, katere začetek označujeta spremenljivki X in Y, konec pa spremenljivki X1 in Y1. Spremenljivka T določa barvo točke.

Primer:

```
100 HIRES 1,6
110 FOR X=0 TO 320 STEP 8
120: M=100/320*X
130: LINE 320-X,M,X,100+M1
140 NEXT X
150 GO TO 150
```

5.10. CIRCLE

Format: CIRCLE X,Y,X1,Y1,T

Namen: Risanje krožnih oblik

Delovanje: Z ukazom CIRCLE rišemo krožne oblike na ekranu. Spremenljivki X in Y nam določata središče lika, X1 in Y1 pa določata radijusa. Vidimo torej, da nam ta ukaz v osnovi riše elipso. Če želimo narisati krožnico, potem moramo upoštevati naslednje popravke radijev:

HIRES grafika	X1 = 1.15*Y1
MULTI grafika	X1 = 0.575*Y1
TISKALNIK	X1 = Y1

./.

Primer:

```
100 HIRES 1,6
110 FOR Y=70 TO 100 STEP 3
120: X=1.15*X
130: CIRCLE 160,     Y,X,Y,1
140: CIRCLE 160,200- Y,X,Y,1
150: CIRCLE 45+X,100,  X,Y,1
160: CIRCLE 275-X,100  X,Y,1
170 NEXT X
200 GO TO 200
```

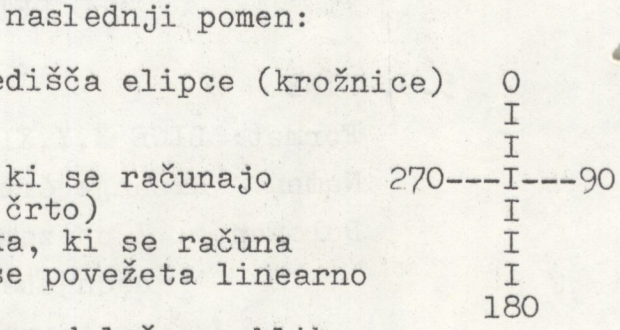
5.11. ARC

Format: ARC X,Y,SA,EA,I,X1,Y1,T

Namen: Risanje lokov

Delovanje: Z ukazom ARC rišemo loke katerih oblike določajo spremenljivke, ki jih moramo navesti. Te spremenljivke imajo naslednji pomen:

X,Y = koordinate središča elipce (krožnice)
SA = začetni kot
EA = končni kot
I = razmak kotov, ki se računajo (I=1 za polno črto)
SA+I= naslednja točka, ki se računa ti dve točki se povežeta linearno
X1 = X-radij
Y1 = Y-radij, z njima določamo obliko krivulje, katere lok rišemo.



Primer:

```
100 HIRES 1,0 : MULTI 6,1,1
110 REM ++++++++++ KROŽNICA ++++++++++
120 FOR X=8 TO 64 STEP 8
130: CIRCLE 80,100,x,100,3
140 NEXT X
150 CIRCLE 80,100,72,100,1
160 REM ++++++++++ DVE ČRTI ++++++++++
170 LINE 80,0,80,200,3
180 LINE 10,100,150,100,1
190 REM ++++++++++ ŠTIRJE LOKI ++++++++++
200 ARC 80,0,152,207,1,1oo,30,1
210 ARC 80,200,333,28,1,100,30,1
220 ARC 80,0,0,142,217,1,100,60,1
230 ARC 80,200,323,38,1,100,60,1
300 GO TO 300
```

5.12. ANGL

Format: ANGL X,Y,SA,X1,Y1,T

Namen: Risanje radija poljubnega lika

Delovanje: Z ukazom ANGL lahko v lik vrišemo radij. Lik katerega radij želimo narisati ni potrebno, da je viden. Spremenljivke imajo naslednji pomen:

X,Y = koordinate zamišljenega lika (krožnica)
SA = kot pod katerim bo radij narisan
X1,Y1 = polosi elipse (krožnice) katere radij želimo narisati

./.

Primer:

```
100 HIRES 2,1
110 FOR X=X TO 358 STEP 3
120: ANGL 220, 125, X,X/1.4,X/3.8,1
130 NEXT X
200 GO TO 200
```

5.13. PAINT

Format: PAINT X,Y,T

Namen: Barvanje določenega dela ekrana

Delovanje: Z ukazom PAINT lahko pobarvamo določen omejen del ekrana. Spremenljivki X in Y določata točko, ki se nahaja znotraj površine, ki jo želimo obarvati. Če željena površina ni strogo omejena bo pobarvan cel ekran.

Primer:

```
100 HIRES 0,11 : MULTI 5,4,6
110 CIRCLE 80,100,48,78,1
120 ANGL 80,100,120,48,78,1
130 ANGL 80,100,160,48,78,1
140 ANGL 80,100,220,48,78,1
150 ANGL 80,100,330,48,78,1
160 PAINT 90,35,1
170 PAINT 60,60,2
180 PAINT 90,120,3
190 LOW COL 7,4,6
200 PAINT 80,110,1
300 GO TO 300
```

5.14. BLOCK

Format: BLOCK X,Y,X1,Y1,T

Namen: Pobarvati pravokotnik

Delovanje: Z ukazom BLOCK rišemo pravokotnik v določeni barvi. Enako bi lahko dosegli z ukazoma REC in PAINT vendar je ukaz BLOCK hitrejši. Pomen spremenljivk je enak kot pri ukazu REC.

Primer:

```
100 HIRES 2,2 : MULTI 2,1,5
110 X=16
120 REPEAT
130 : FOR F=1 TO 3
140 : DY=RND (1) *140
150 : BLOCK X,10+DY,X+10,160,F
160 : X=X+8
170 : NEXT F
180 UNTIL X > 120
200 GO TO 200
```

5.15. DRAW

Format: DRAW "NMNN...9",X,Y,T

N amen: Risanje geometrijskega lika

./.

Delovanje: Z ukazom DRAW lahko narišemo poljuben geometrijski lik. Lik se riše podobno kot na papirju. S pomočjo številk (n) določamo smer po kateri se bo gibal "svinčnik" in ali bo puščal za sabo sled ali ne.

(N)	pomen
0	neviden pomik v desno
1	neviden pomik gor
2	neviden pomik dol
3	neviden pomik v levo
4 = 2	
5	viden pomik v desno
6	viden pomik gor
7	viden pomik dol
8	viden pomik v levo
9	konec lika

Številke vnašamo direktno v narekovajih ali pa jih prej definiramo kot string. Spremenljivki X in Y določata začetni koordinati. Željeni lik se na ekranu prikaže z ukazom ROT.

Primer:

```
1000 HIRES 2,1
110  A$ = "55555555555555555555555555"
120  D$ = "88888888888888888888888888"
130  B$ = "65566566686686888888878878"
140  E$ = "78877877757757557555565565"
160  F$ = "56556556556556556556556659"
170  C1$= "555555555555559"
180  L$ = D$+E$+F$
190  R$ = A$+B$+C$+C1$
200  FOR U=0 TO 7 STEP 2
210 : ROT U,1
220 : DRAW R$, 160, 100,1
230 : DRAW L$, 160, 100, 1
240 : DRAW R$, 160, 100, 0
250 : DRAW L$, 160, 100, 0
260   NEXT U
300 GO TO 300
150 C$="878878878878878878878555555"
```

5.15. ROT

Format: ROT R,S

Namen: Rotacija in menjava velikosti lika

Delovanje: Z ukazom ROT lahko geometrijski lik, ki smo ga narisali z ukazom DRAW, rotiramo za določen kot ali pa mu spremenimo velikost. Kot rotacije se določa s spremenljivko R kot sledi:

./.

R	kot rotacije (v stopinjah)
0	0
1	45
2	90
3	135
4	180
5	225
6	270
7	315

Spremenljivka S določa velikost lika in lahko ima vrednost od 1 do 255.

OPOZORILO: Velikost lika je odvisna od vrste grafike (HIR-ES/MULTI)

5.16. CSET

Format: CSET N

Namen: Izbira oblike znakov ali vračanje zadnjega grafičnega ekrana

Delovanje: Z ukazom CSET lahko v programu izbiramo obliko znakov. Spremenljivka N ima naslednji pomen:

CSET 0 - velike črke / grafični mod
CSET 1 - velike črke / male črke
CSET 2 - HIRES grafika - vračanje zadnjega ekrana

Primer: V programu za risanje blokov (5,14) dodaj naslednje ukaze:

```
 90 REM +++++++++ CSET o/1 ++++++++++
200 PAUSE 5: CSET 0
210 PRINT "ČE ŽELIŠ VRNITI ZADNJO GRAFIKO
      PRITISNI TIPKO"
220 GET A$
230 IF A$ = "" THEN 220
240 IF A$ = "Q" THEN END
250 CSET 2 : MULTI 2,1.5.
260 GO TO 200
```

6. TEKST V GRAFIKI

6.1. CHAR

Format: CHAR X,Y, CODE,T,F

Namen: Pisanje znakov na ekran ko je vključena grafika

./.

Delovanje: S tem ukazom lahko na ekranu prikažemo nek znak (črko) tudi takrat ko je vključen grafični mod. Spremeljivke imajo naslednji pomen:

X,Y - koordinate gornjega levega vogala znaka
CODE - kod znaka iz tabele znakov ASCII)
T - tip točke
F - faktor povečanja znaka v Y - smeri

Primer: Programu iz prejšnjega poglavja (5.14 in 5.16) dodaj naslednji ukaz:

195 CHAR X,150,X/8-1,F-1,1

6.2. TEXT

Format: TEXT X,Y,"(CTRL-A) string", T,F,R

Namen: Pisanje niza znakov na ekran ko je vključena grafika

Delovanje: S tem ukazom lahko pišemo na ekran nize znakov v nasprotju s prejšnjim ukazom kjer lahko naenkrat napišemo na ekran en znak. Spremenljivke imajo naslednji pomen:

X,Y - koordinate levega zgornjega vogala niza
Naslednji parameter je niz ali niz spremenljivka.
Prvi znak v nizu odreja vrsto znakov:
(CTRL-A) - velike črke; (CTRL-B) male črke
T - tip točke
F - višina znaka
R - razmak med posameznimi znaki (širina znaka se ne da spreminjati)

7. UPRAVLJANJE Z SLIKO NA EKRANU

Z ukazi razloženimi v tem poglavju se bomo naučili kako se upravlja s sliko na ekranu. S temi ukazi lahko v Simon`s Basicu kontroliramo barve na ekranu, ekran polnimo z željenimi znaki, podvojujemo in premikamo dele ekrana, kakor tudi shranjujemo ekranske podatke na zunanji spomin ali tiskalnik.

7.1. BCKGNDS

Format: BCKGNDS SC, B1, B2, B3

Namen: Menjava barve ozadja znakov

Delovanje: S tem ukazom določimo barvo ozadja znakov in znakov samih. Spremenljivke imajo naslednji pomen:

./.

SA - izbira barve ekrana (iz tabele barv)
B1 - izbira barve znakov vpisanih s SHIFT
B2 - izbira barve znakov vpisanih z RVS/ON
B3 - izbira znakov barve vpisanih s SHIFT in RVS/ON

Primer: BCKGNDS 1,6,7,5

7.2. FLASH

Format: FLASH B,S

Namen: Hitro menjavanje barve ekrana

Delovanje: Z ukazom FLASH lahko hitro menjujemo barvo ekrana in sicer med normalnim ekranom in inverznim. Spremenljivka B določa barvo ekrana, S pa hitrost menjave. Vrednost spremenljivke S je lahko med 0 in 255, vsak korak pa pomeni hitrost 1/16 sekunde. Ukaz FLASH v HIRES in MULTI grafiki ne deluje.

Primer:

```
100 POKE 53280,8 : POKE 53281,6
110 FOR X=16 TO 1 STEP-1
120 : FLASH 7,X
130 : PAUSE 1
140 NEXT X
200 OF
```

7.3. OFF

Format: OFF

Delovanje: Ukaz OFF končuje ukaz FLASH

7.4. BFLASH

Format: BFLASH S,B1,B2

Namen: Kontinuirano spreminjanje barv okvirja ekrana

Delovanje: S tem ukazom dosežemo kontinuirano spreminjanje barv okvirja ekrana. Barvi sta določeni s spremenljivkama B1 in B2. Spremenljivka S pa ima enak pomen kot v ukazu FLASH. Ukaz oz. njegovo delovanje zaključimo z BFLASH 0.

7.5. FCHR

Format: FCHR R,C,W,D,CODE

Namen: Na določen del ekrana vpisati več enakih znakov

Delovanje: Z ukazom lahko na del ekrana vpišemo več enakih znakov. Spremenljivki R (0-24) in C (o-39) nam povesta koordinati prvega znaka,

./.

spremenljivka W nam določa število znakov po stolpcih, spremenljivka D pa število znakov po vrstah. Spremenljivka CODE ima enak pomen kot v ukazu CHAR.

Primer: glej primer za ukaz FCOL

7.6. FCOL

Format: FCOL R,C,W,D,B

Namen: Določevanje barve znakov na določenem delu ekrana

Delovanje: S tem ukazom določimo barvo znakov na določenem delu ekrana. Spremenljivke R,C,W,D, imajo enak pomen kot pri ukazu FCHR. Spremenljivka B pa določa barvo.

Primer:

```
100 POKE 53280,8: POKE 53281,8
110 PRINT "(SHIFT/CLR/HOME)"
120 FCHR 10,10,10,10,10
130 FOR X=10 TO 15 STEP 5
140 : FOR Y=10 TO 15 STEP 5
150 : FCOL X,Y,5,5,F
160 : F=F+1
170 : NEXT Y
180 NEXT X
```

7.7. FILL

Format: FILL R,C,W,D, CODE,B

Namen: Na določen del ekrana vpisati več enakih znakov v določeni barvi.

Delovanje: Ukaz FILL je pravzaprav sinteza ukazov FCHR in FCOL, zato spremenljivk ne bomo posebej razlagali.

Primer:

```
100 POKE 53280,8 : POKE 53281,8
110 PRINT "(SHIFT/CLR/HOME)"
120 FOR X=10 TO 15
130 : FILL X,2*X,5,5,X,X;
140 NEXT X
```

7.8. MOVE

Format: MOVE R,C,W,D,DR,DC

Namen: Podvojevanje (kopiranje) dela ekrana.

Delovanje: Z ukazom MOVE lahko prenašamo (kopiramo) del ekrana na drugo mesto. Prve štiri spreMenljivke imajo enak pomen kot v prejšnjih ukazih. Spremenljivki DR in DC pa pomenita začetni koordinati področja kamor hočemo prenesti del ekrana.

./.

Primer:

```
100 POKE 53280,8 : POKE 53281,8
110 PRINT "(SHIFT/CLR/HOME)"
120 FILL 0,0,5,5,1,1,
130 MOVE 0,0,5,5,0,35
140 MOVE 0,0,5,5,19,0
150ØMOVE 0,0,5,5,19,35
```

7.9. INV

Format: INV R,C,W,D

Namen: Invertiranje dela ekrana

Delovanje: Ta ukaz invertira del ekrana, ki ga določimo s spremenljivkami R,C,W,D. Pomen spremenljivk je enak kot pri prejšnjih ukazih.

Primer: V programu za ukaz MOVE dodaj naslednja dva ukaza:

```
160 PAUSE 3
170 INV 2,2,26,4
180 GO TO 160
```

7.10. SCROLL

Format: SmerW SL,SC,EL,EC
SmerB SL,SC,EL,EC

Namen: Premikanje dela ekrana

Delovanje: Simon`s Basic omogoča pomikanje dela ekrana v štirih različnih smereh. Ime ukaza (Smer) določa smer pomika slike in je lahko:

UP - pomik gor, DOWN - pomik dol,
LEFT- pomik v leto, RIGHT - pomik v desno

Naslednja spremenljivka v imenu ukaza pa določa tip pomika:

B - pomik slike brez kroženja, to pomeni, da se slika ali njen del, ki izgine iz ekrana ne pojavi na nasprotni strani.

W - pomik slike z kroženjem, slika ki izgine iz ekrana se pojavi na nasprotni strani.

Spremenljivke SL,SC,EL,EC imajo naslednji pomen:

SL,SC - prva vrsta, stolpec
EL,EC - zadnja vrsta, stolpec

Primer:

```
100 PRINT "(SHIFT/CLR/HOME)"
110 FOR X=1 TO 39
120 : Y=INT (10*SIN(X/3.14159)+12
130 : PRINT AT (X,Y) "*"
140 NEXT X
150 LEFTW 0,0,20,25 : RIGHTW 0,20,20,25
160 GO TO 150
```

./.

7.11. SCRSV

Format: SCRSV 2,8,2, "ime, S,W" - shranjevanje na disketo
SCRSV 1,1,1, "ime, S,W" - shranjevanje na kaseto

Namen: Shranjevanje slike nizke ločljivosti (normalen ekran)

Delovanje: S tem ukazom lahko shranimo na kaseto ali disketo podatke, ki jih vidimo na ekranu. Shranjujemo lahko samo podatke (sliko), ki je narejena v normalnem text modu. Slik, ki so narejene v HIRES ali MULTI modu ne moremo shranjevati s tem ukazom.

Primer:

```
100 PRINT "(SHIFT/CLR/HOME)"
110 FILL 6,10,20,4,160,2)
120 FILL 10,10,20,4,160,1
130 FILL 14,10,20,4,160,6
140 SCRSV 1,1,1,"SLIKA 1,S,W"
150 END
```

7.12. SCRLD

Format: SCRLD 2,8,2, "ime" - polnjenje slike in diskete
SCRLD 1,1,1, "ime" - polnjenje slike iz kasete

Namen: Polnjenje slike nizke ločljivosti

Delovanje: S tem ukazom lahko sliko, ki smo jo shranili z ukazom SCRSV ponovno prikažemo na ekranu.

Primer: SCRLD 1,1,1, "SLIKA 1"

7.13. COPY

Format: COPY

Namen: Shranjevanje slike visoke ločljivosti na papir

Delovanje: S tem ukazom lahko sliko, ki je bila narejena v HIRES ali MULTI modu shranimo na papir. Seveda potrebujemo zato tiskalnik, ki lahko piše grafične znake, oz. tako imenovani grafični tiskalnik.

Primer: COPY

7.14. HRDCPY

Format: HRDCPY

Namen: Shranjevanje slike nizke ločljivosti na papir

Delovanje: Ta ukaz nam omogoči, da sliko, ki je narejena v normalnem text modu prenesemo na papir.

Primer: HRDCPY

./.

8. S P R I T E

Na CBM 64 lahko generiramo tudi male sličice velikosti 24*21 bitov, ki jih imenujemo SPRITE. Teh sličic je lahko naenkrat na ekranu največ 8 in jih lahko poljubno premikamo po ekranu. Poleg SPRITE-ov lahko na CBM 64 generiramo tudi svoj set znakov. Seveda lahko obe možnosti uporabljamo tudi v normalnem (standardnem) Basicu, vendar nam je delo v Simon`s Basicu bistveno olajšano. V Simon`s Basicu se SPRITE imenuje MOB, kar je kratica za Movable Object Block. Z uporabo sprite-ov v grafičnem programiranju razbremenjujemo procesor, ker se z njimi ukvarja VIC-II chip.

8.1. DESIGN

Format: DESIGN C,AD
DESIGN C,SA+GC

Namen: Določiti prostor v memoriji, kjer želimo SPRITE shraniti.

Delovanje: S tem ukazom rezerviramo v memoriji prostor za en sprite. S spremenljivko C določimo ali je sprite v HIRES grafiki (o), ali je v MULTI grafiki (1). Spremenljivka AD nam določa področje v memoriji (adress), kjer se nahajajo podatki za sprite. Ta naslov se izračuna iz številke bloka pomnoženega z 64 in sicer zato, ker vsak sprite zavzema 64 bytov memorije.

Številka bloka	Naslov memorije (dec.)
13 - 15	832 - 1023
32 - 63	2048 - 4095
128 -255	8192 -16383

Če želimo sprite prikazati v HIRES modu, moramo naslovu memorije prišteti konstanto 49152. Če v programu uporabljamo ukaz MEM, takrat se lahko uporablja samo blok 192 - 255.

OPOZORILO: Shranimo lahko toliko spritov, kolikor imamo na razpolago spomina, vendar lahko na ekranu prikažemo največ 8 spritov naenkrat. Seveda pa lahko v programu meddelom zamenjamo en sprite z drugim na istem naslovu.

./.

Primer: Glej program na koncu poglavja!

8.2. Master space (znak štev. 64)

Format: @.....

Namen: določiti obliko MOB-a

Delovanje: Kot smo povedali v uvodu je MOB matrica velika 24*21 točk. Če želimo MOB v večih barvah potem je matrica velika 12*21 točk. Vsaka definicija MOB-a je sestavljena iz 21 vrstic, v katerih je odvisno od HIRES ali MULTI MOB-a 24 ali 12 točk, namesto katerih lahko tipkamo črke, ki nam povejo kakšne barve bo točka. Če pustimo točko (ne vtipkamo črke) bo točka na ekranu v barvi ozadja, torej nevidna.

MOB - Visoke ločljivosti

Črka B	Barva: barva navedena v MOB SET ukazu.

MOB - Večbarvni (MULTI-COLOR)

Črka	Barva
B	Barva 1 (C1) iz ukaza CMOB
C	Barva iz ukaza MOB SET
D	Barva 2 (C2) iz ukaza CMOB

Primer: Glej program na koncu poglavja!

8.3. CMOB

Format: CMOB C1, C2

Namen: Določiti barve za večbarvne MOB-e.

De-lovanje: S tem ukazom določimo obe dodatni barvi za vse večbarvne MOB-e. Spremenljivki C1 in C2 imata lahko vrednost od Ø do 15 kar ustreza številkam za barve iz poglavja 3.

Primer: Glej program na koncu poglavja!

8.4. MOB SET

Format: MOB SET MB,BLK,COL,PR,RES

Namen: Določiti parametre MOB-a.

Delovanje: S tem ukazom določimo parametre MOB-a. Spremenljivke pa imajo naslednji pomen:

MB - Številka MOB-a(Ø-7). Kadar je na ekranu več MOB-ov imajo tisti z nižjo številko višjo prioriteto. To pomeni, če se MOB-i prekrivajo bo viden tisti, ki ima nižjo številko.

./.

BLK - spremenljivka določa v katerem bloku je MOB definiran (glej 8.1.)

COL - spremenljivka določa barvo posameznega MOB-a in ima lahko vrednost od Ø do 15, kar ustreza številkam za barve. (glej tudi 8.2. in 8.3.)

PR - določanje prioritete MOB-a
Ø - MOB se premika pred ekranskimi znaki (črkami)
1 - MOB se premika za ekranskimi znaki

RES - Ø MOB visoke resokucije (ločljivosti)
1 večbarvni MOB

Primer: Glej program na koncu poglavja!

8.5. MMOB

Format: MMOB MN,X1,Y1,X2,Y2,EXP,SP

Namen: Postavljanje ali premikanje MOB-a

Delovanje: S tem ukazom prikažemo MOB na ekranu, omogoča pa nam tudi njegovo premikanje. Spremenljivke imajo naslednji pomen:

MN - določa kateri MOB želimo prikazati
X1,Y1 - koordinate kamor se MOB postavi na začetku
X2,Y2 - koordinate kamor se MOB giblje
EXP - velikost MOB-a: Ø - normalna velikost
1 - 2 x povečan po X-osi
2 - 2 x po-večan po Y-osi
3 - 2 x povečan po X in Y osi
SP - hitrostMOB-a: 1 - najhitreje
255 - najpočasneje

Primer: Glej program na koncu poglavja!

8.6. RLOCMOB

Format: RLOCMOB MN,X,Y,EXP,SP

Namen: Premikanje MOB-a po ekranu.

Delovanje: S tem ukazom premikamo po ekranu MOB, ki je že na ekranu. Spremenljivki X in Y nam določata cilj, ostale spremenljivke pa so opisane v prejšnjem poglavju.

8.7. DETECT

Format: DETECT N

Namen: Pripraviti Simon`s Basic na testiranje trka dveh MOB-ov

Delovanje: V spominu računalnika se nahaja en register (mem.lokacija), kamor se beleži trenutek ko sta

./.

se dva MOB-a in črka na ekranu dotaknila. Ta register uporablja ukaz CHECK zato ga moramo najprej pripraviti za uporabo. Ukaz DETECT moramo uporabiti dvakrat in sicer se po prvem ukazu register briše, z drugim pa omogočimo, da sprejme novo vrednost. Če je spremeljivka N = Ø pomeni, da želimo testirati trk dveh MOB-ov, če pa je N = 1 pomeni, da želimo testirati trk črke in MOB-a.

Primer: Glej program na koncu poglavja!

8.8. CHECK

Format: IF CHECK (mn1, mn2)=Ø THEN ukaz.
IF CHECK (Ø)=Ø THEN ukaz.

Namen: Testirati če je prišlo do trka.

Delovanje: S tem ukazom testiramo če je prišlo do trka. Spremenljivki mn1 in mn2 povesta katera dva MOB-a želimo testirati. Če pa v oklepaju napišemo Ø pomeni, da želimo testirati trk kateregakoli MOB-a z znakom na ekranu. Če je do trka prišlo se izvaja ukaz, ki je napisan za THEN.

Primer: Program služi kot primer za vse ukaze tega poglavja!

```
110 REM ****************************
120 REM * EXAMPLE:                  *
130 REM * MOB'S WITH SIMON'S BASIC *
140 REM ****************************
150 REM * CREATE ONE                *
160 REM * HIGH-RESOLUTION-MOB       *
170 REM * IN BLOCK 13               *
180 REM ****************************
190 PRINT "␣"
200 DESIGN 0,13*64
210 @.........B.B..........
220 @..........BB.BB.......
230 @............BB..B.....
240 @............BBBBB.....
250 @..........BBBB.BBB....
260 @.........BBBBB....B....
270 @.........BBBBB.........
280 @.......BBBBBBBBBB.....
290 @......BBBBBBBB..BB.....
300 @....BBBBBBBBBB........
310 @...BBBBBB..BB.........
320 @..BB.BBBBBBB..........
330 @..BB.BBBBBBBB.........
340 @.BB...BBBBBBB.........
350 @.BB.....BBBBBB........
360 @BB.........BBBB.......
370 @BB.......BBBB.........
380 @........BBB...........
390 @.......BB.............
400 @......BBBBBBB.........
410 @.......BBBBBBB........
420 REM ****************************
430 REM * CREATE ONE                *
440 REM * HIGH-RESOLUTION-MOB       *
450 REM * IN BLOCK 14               *
460 REM ****************************
470 DESIGN 0,14*64
480 @............BB........
490 @............BBBB.BBB...
500 @..............BBBBBBBB..
510 @................BBBB.BB.
520 @................BBBBBBBBB
530 @..............BBBBBB....
540 @............BBBBBB......
550 @..........BBBBBBBBBBBB.
560 @.........BBBBBBBBBBBBBB
570 @.......BBBBBBBBBBB...BB
580 @......BBBBBBBBBBBB.....
590 @......BBBBBBB..BBBB.....
600 @.....BBBBBBBBB.BBB.....
610 @....BBBBBBBBBB.BBBB....
620 @....BBBBBBBBB.BBBB.....
630 @...BBBB.BBBBBBBBB......
640 @..BBBB...BBBBBBB.......
650 @BBBBB....BBBB..........
660 @BBB.....BBBB...BBB.....
670 @........BBBBBBBBBB.....
680 @.........BBBBBBBBB.....
```

```
690 REM ******************************
700 REM * CREATE ONE                  *
710 REM * MULTI-COLOUR-MOB            *
720 REM * IN BLOCK 15                 *
730 REM ******************************
740 DESIGN 1,15*64
750 @.....BB.....
760 @..BBBBCCCB..
770 @.DBBBCCCCCB.
780 @.BBBCCCCCCC.
790 @BBBBCCCDDCCB
800 @BBBBCCCDDCCB
810 @.BBBCCCCCCC.
820 @.BBBBCCCCCB.
830 @..BBBBCCCB..
840 @.D...BB...D.
850 @............
860 @..D......D..
870 @............
880 @...D....D...
890 @............
900 @....D..D....
910 @....CCCC....
920 @....CCCC....
930 @....CCCC....
940 @...DCCCCD...
950 @...DDDDDD...
960 REM ***************************
970 REM * MULTI-COLOUR-MOB NO.1 *
980 REM * EXPANSION ON Y-WAY     *
990 REM * POS   0/0-300/205      *
1000 REM ***************************
1010 CMOB 7,2                      \
1020 MOB SET 1,15,8,0,1
1030 MMOB 1,  0,  0,300,205,2,200
1040 REM ***************************
1050 REM * MULTI-COLOUR-MOB NO.2 *
1060 REM * EXPANSION ON X&Y WAY   *
1070 REM * POS 200/200-300/50     *
1080 REM ***************************
1090 CMOB 0,14
1100 MOB SET 2,15,6,1,1
1110 MMOB 2,200,200,300,50,3,150
1120 REM ***************************
1130 REM * MULTI-COLOUR-MOB NO.7 *
1140 REM * EXPANSION ON X-WAY     *
1150 REM * POS 100/0-300/150      *
1160 REM ***************************
1170 CMOB 8,5
1180 MOB SET 7,15,13,0,1
1190 MMOB 7, 10,150,300,175,0, 50
1200 REM ***************************
1210 REM * MULTI-COLOUR-MOB NO.6 *
1220 REM * WITHOUT EXPANSION      *
1230 REM * POS  10/130-300/175    *
1240 REM ***************************
1250 CMOB 3,4
1260 MOB SET 6,15,0,1,1
```