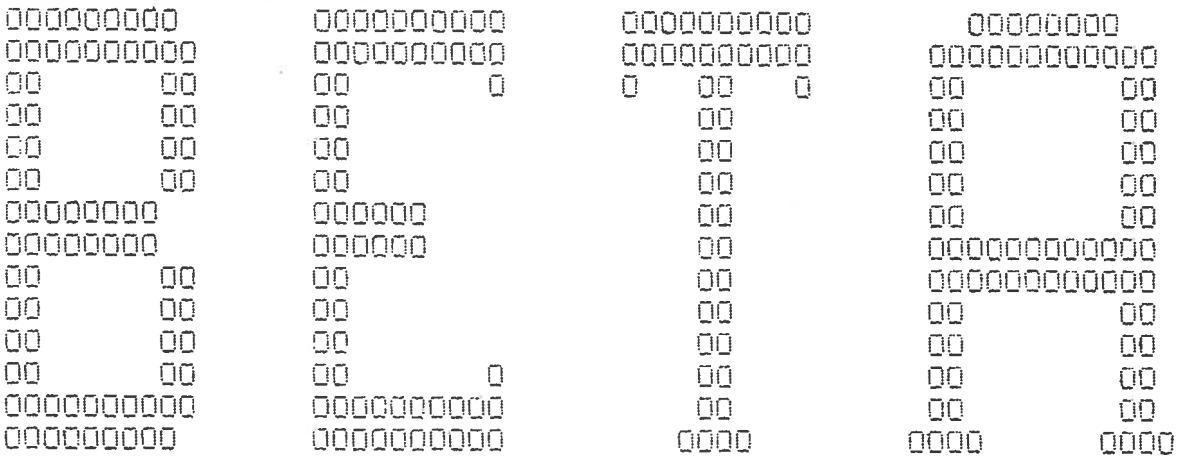

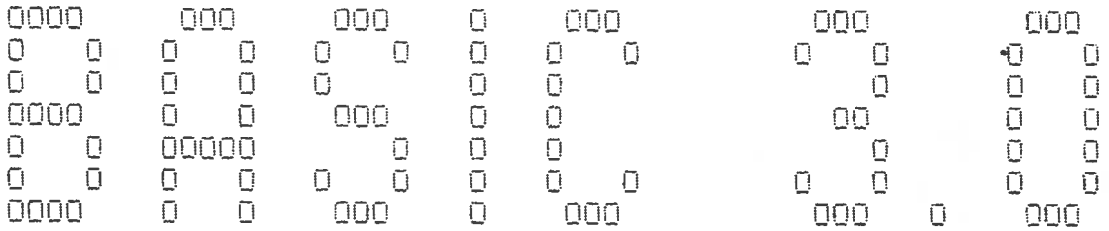

&

BETABASIC 1.0
BETABASIC 1.9
SIGMABASIC 1.10T
SIGMABASIC 1.12T

Ú V O D

Tento přehled se týká hlavně BetaBasicu 3.0, nicméně se zde dovíte základní věci i o verzích předchozích, t.j. BB 1.0, BB 1.8 i verzi Sinsoft 1.10T označovanou jako SigmaBasic.

Vycházel jsem z textu zaznamenaného A.Mikusem v roce 1996, z dalších materiálů neznámých autorů i z vlastní, asi dvouleté zkušenosti z práce s BetaBasicem.

Díky BB03 má Spectrum nejlepší osmibitový BASIC. Pokud jste zvyklý na normální BASIC Spectra budete překvapen možnostmi, které máte k dispozici. Musíte ovšem počítat s tím, že jako obvykle zaplatíte za zvýšené pohodlí omezením v oblasti paměti.

Verze 1.0 vám ubírá asi 5,5 kB, V 1.8 asi 9.5 kB, V 1.10T 12 kB a konečně 3.0 kolem 18 kB.

Pro práci v těchto rozšířeních není třeba zvlášť velké školení. Nejprve si přečtěte poznámky o editaci a potom prostudujte v libovolném pořadí popis jednotlivých příkazů.

Každý BB nahrajete obvyklým LOAD "". BASICová část obvykle obsahuje řádky 0,1,2. V řádku 0 jsou obsaženy definice rozšiřujících fukcí BB, přesněji řečeno odkazy na strojový kód, který zajišťuje jejich provedení.

Natažený BASIC odstartuje na řádku 2, nastaví RAMTOP na potřebnou hodnotu (BB03 na 47070), Natáhne se a spustí strojový kód. Potom se vymažou řádky 1 a 2 a BB03 je připraven k práci.

P O Z N Á M K Y

Rádek 0 obsahuje definice funkcí BB03. Tento řádek musí být částí každého programu, který tyto funce používá. Pokud máte programy, které nebyly vytvořeny v BB, je nutné je natahovat pomocí MERGE, protože LOAD by řádek 0 zrušilo. Je změněna činnost NEW, NEW nyní maže celý program kromě řádku 0, oblast proměnných kromě zvláštního případu proměnných se jmény xrg, yrg, xos a yos. O nich si povíme později. Programy, které napíšete v BB, můžete natahovat přes LOAD, protože ty už řádek 0 obsahují.

Činnost BB je signalizována prodlouženým pípáním klávesnice. Lze samozřejmě vypnout pomocí POKE 23609,0. Dalším příznakem práce je invertovaný řádkový kurzor.

VYTVOŘENÍ KOPIE

Řádek 1 je v podstatě ukládací rutina, která uloží na pásek (nebo drajv) nejprve BASICový program a hned za ním následuje strojový kód.

Bezpečnou kopii lze vytvořit následovně:

1. Natáhnout BASICový program pomocí MERGE
2. CLEAR rt: LOAD ""CODE
3. Pokud chcete kopii na drajv, musíte řádek 1 odpovídajícím způsobem změnit.
4. Po RUN se uloží program i kód.

Pozn. 1 Strojový kód BB03 se nedá ověřit pomocí VERIFY, pokud je BB03 v činnosti. Pokud to chcete přesto udělat, je třeba jej vypnout pomocí RANDOMIZE USR 59904.
Pozn. 2 Jinak lze samozřejmně provést kopii libovolným kopírovacím programem.

Manuál konstatuje, že programy napsané v normálním Sinclair BASICu běží po BB rychleji. Je to způsobeno tím, že v cyklech a skocích jsou použity přímo adresy a ne čísla řádků. Dodal bych, že toto zrychlení není příliš podstatné.

EDITACE

K dispozici jsou nové možnosti : EDIT, KEYWORDS, LIST FORMAT, CSIZE, JOIN, SPLIT.

Starý editor Spectra je nadále k dispozici, ale můžete použít i další. Podrobně jsou popsány dále v seznamu příkazů.

Řádkový kurzor:

- je zobrazen inverzně
- jeho ovládání kurzorovými tlačítky je rychlejší. Pokud dojde k tomu, že jeho pozice nesouhlasí - stisknete <ENTER>.

Editace řádku:

Klávesa <0> funguje jako příkaz EDIT. Stisknete-li ho po <ENTER>, příkaz EDIT se vypíše do vstupního řádku. <0>, číslo řádku, <ENTER> má za následek výpis řádku do editační zóny. Pokud neuvedete číslo řádku je předhozen k editaci tzv. běžný řádek (current line), t.j. řádek označený inverzním kurzorem.

EDIT lze také napsat jako <CS>+<9> (GRAPHICS), <CS>+<5>. Původní EDIT (<CS>+<1>) funguje samozřejmě také.

Pohyb kurzoru v řádku:

V editovaném řádku lze pohybovat kurzorem všemi kurzorovými klávesami - tedy i nahoru a dolů, což editaci značně zrychluje. Kromě toho dáte-li po <EDIT> hned <CS>+<7> tedy 'nahoru', skočí kurzor hned na konec editovaného řádku.

Kurzorové režimy:

Za normálních okolností je možné vložit klíčové slovo jen v režimu 'K'. Text pak v režimech 'L' nebo 'C'. V BB03 si můžete kurzorový režim přepínat. Jste-li v režimu 'K' dostanete se do režimu 'C/L' stiskem <SPACE>. Naopak z 'L/C' se do režimu 'K' dostanete současným stiskem <SS> (Symbol Shift) + <ENTER>. Že je to výhodné, poznáte v praxi.

KEYWORDS:
========

Tento příkaz (kromě jiného) ovlivňuje i režimy vkládání. Můžete příkazy a názvy funkcí vkládat buď ve formě u Spectra obvyklé t.j. klíčovými slovy, nebo po jednotlivých znacích, jak je zvykem u většiny jiných počítačů. Po načtení BB je aktivní režim KEYWORDS 3, který akceptuje oba způsoby.

Pro jednoklávesové vstupy nových klíčových slov v režimu 'K' je třeba přejít do režimu 'G' a pak stisknout odpovídající klávesu s doplňkovým příkazem.

Název funkce lze napsat standardně, t.j. FN + písmeno + závorky s argumentem (y), nebo vypsat ' ','F','N',' ' ..., nebo je možné vypsat přímo název funkce BB, např.: BIN$(123).

Test vstupu:

Po stisku <ENTER> je vložený řádek kontrolován na syntaktickou správnost. Je-li nalezena chyba, ozve se BEEP. Jeho délka je určena hodnotou na adrese 23609.

Při vkládání programu ve formě textových řetězců, je vhodné vkládat text malými písmeny. BB rozezná příkazy a po převedení na klíčová slova je zobrazí velkými znaky, ostatní znaky zůstanou malé.

LIST FORMAT:
===========

Příkazem lze ovlivňovat formát listingu. Struktura programu tím jasně vynikne, program je čitelnější a snadněji se objeví případné chyby.

CSIZE:
=====

Příkazem lze měnit velikost znaků tištěných na obrazovku.

JOIN a SPLIT
============

JOIN spojí dva řádky programu v jeden.
SPLIT rozdělí jeden řádek na dva.

Vložení CR do programového řádku:

V libovolném místě je možné vložit do vstupního řádku programu nebo do INPUT řádku znak CHR$ 13=CR a to pomocí <CS> + <ENTER>. Rádek tím není ukončen, ale dojde k přechodu na nový řádek.

P R O C E D U R Y A P A R A M E T R Y

Nová klíčová slova:

DEF PROC, PROC, LOCAL, DEFAULT, REF, READ LINE, LIST PROC,

Funkce:

ITEM()

Tato kapitola vás seznámí s možnostmi použití procedur. Napsanou proceduru je možné pomocí MERGE podle potřeby kdykoliv nahrát z knihovny, kterou si tímto způsobem můžete vytvořit.

V programu voláme proceduru přímo jménem, za které se napíší případné parametry. Procedury uložené v paměti je možné volat i v přímém režimu, což vlastně umožňuje rozšířit množinu příkazů BB. Každá procedura by měla být jen tak velká, aby byla snadno pochopitelná.

Procedura je vlastně podprogram, který začíná klíčovým slovem DEF PROC jméno a popř. seznam parametrů, které jsou jí při jejím volání předány. Na konci procedury je klíčové slovo END PROC. Interpreter proceduru najde podle jejího jména bez ohledu na to, na kterém místě v programu je umístěna. A nezáleží na tom, zda program již běžel nebo ne.

Příklad: Vytvoření ekvivalentu příkazu DIR, používaného operačními systémy DOS a CP/M.

```
100 DEF PROC DIR
110   CAT 1
120 END PROC
```

Vložíte-li RUN, nic se nestane. Procedura nebyla volaná. Chování je podobné jako u DEF FN.

Vložte: 10 dir a pak RUN. Vypíše se obsah drajvu. Napište DIR a příkaz se znovu provede. Spectrum má prakticky o příkaz víc.

Jméno procedury musí začínat písmenem a končit znakem 'mezera', ':', 'REF', 'DATA' nebo 'ENTER'.

Jméno může být tvořeno znaky velké i malé abecedy (velikost písmen je ignorována), číslicemi, potržením. Je možno používat i jiné znaky, ne však znaky uživatelské grafiky.

Procedura musí začínat slovem DEF PROC a končit END PROC. END PROC smí být v proceduře pouze jednou.

Příklad:
```
100 DEF PROC dir cislo
110   CAT cislo
120 END PROC
```

Proměnná 'cislo' je nazývána formální parametr. Přes formální parametry se při volání procedury předávají do procedury skutečné hodnoty. Při volání procedury pak lze použít následující tvary:

```
dir 1
```
nebo
```
LET x=1: dir x
```
nebo
```
dir <libovolný numerický výraz>
```

Tento způsob je označován jako předání parametrů hodnotou, protože procedura převezme pouze hodnotu.

Formální parametr při definici procedury musí být proměnná. Jde však o proměnnou lokální a platí jen uvnitř dané procedury. Po jejím ukončení ji BB nezná, neexistuje a také nezabírá paměť. Proměnná použitá jinde v těle procedury je systému známá, je globální. Aby byla také lokální je nutné použít příkazu

LOCAL. Příklad:

```
100 DEF PROC nazdar xkrat
110   FOR n=1 TO xkrat
120     PRINT "NAZDAR"
130   NEXT n
140 END PROC
```

(Pozn.: tento a všechny další výpisy programů v tomto textu jsou ve formátu příkazu LIST FORMAT 2.)

Pokud napíšete: nazdar 4

procedura se provede. Pak napište PRINT n, xkrat. Hodnota n se vypíše, ale xkrat ne. Dostanete chybovou zprávu '2 Variable not found'.

Dobří programátoři zajišťují, aby všechny proměnné používané uvnitř procedury byly lokální. Tedy 'n' by mělo být lokální. Zabezpečíte to vložením řádku:

```
105   LOCAL n
```

Pokud by někde v programu existovala globální proměnná se jménem 'n', nemá na ni volání procedury žádný vliv.

Příklad:

```
100 DEF PROC box x,y,sirka,delka
110   PLOT x,y
      DRAW sirka,0
120   DRAW 0,-delka
      DRAW -sirka,0
130   DRAW 0,delka
140 END PROC
```

Příkaz BOX 100,100,10,40 nakreslí obdélník s levým rohem v bodě 100,100 se šířkou 10 a délkou 40.

Pokud některý z parametrů vynecháte, dojde k chybě. Lze ovšem použít příkaz DEFAULT. Např.:

```
105   DEFAULT delka=sirka
```

Nyní můžete zavolat BOX 100,100,50 a nakreslí se čtverec.

Příkaz DEFAULT nastaví hodnotu parametru v případě, že nebyla nastavena při volání procedury.

S E Z N A M Y P A R A M E T R Ů

Za voláním procedury může být žádný nebo několik parametrů. Může ovšem nastat případ, že předem nevíme kolik bude mít daná procedura parametrů. V tomto případě použijeme místo formálních parametrů klíčové slovo DATA a uvnitř procedury parametry přebíráme pomocí READ. Potřebné informace o parametrech nám dodá funkce ITEM(), která může vracet následující hodnoty:

```
0  - Není k dispozici žádný parametr
1  - Další parametr je řetězec
2  - Další parametr je číselný
```

Příklady:

```
100 DEF PROC KILL DATA
110   DO UNTIL ITEM()=0
120     READ a$
130     ERASE 1,a$
140   LOOP
150 END PROC
```

Procedura vymaže na drajvu soubory, jejichž názvy tvoří parametr ve formě seznamu. Jednotlivé položky musí být odděleny čárkami. Volání procedury může mít tedy tvar:

```
KILL "soubor 1"
KILL "soubor 1","soubor 2", ... , "soubor n"
```

Pokud změníte řádek 120 na:

```
120     READ LINE a$
```

pak je možné i volání:

```
KILL file 1
KILL file 1,file 2, ...  file n
```

Příklad:

```
10 DEF PROC tisk DATA
20   LOCAL a$,a
30   DO WHILE ITEM()<>0
40     IF ITEM()=1 THEN
         READ a$
         PRINT PAPER 6;a$
       ELSE
         IF ITEM()=2 THEN
           READ a
           PRINT INVERSE 1;a
50   LOOP
60 END PROC
70 let a1=1000, a2=7777
80 tisk 1,2,"tri",a1,"ctyri",5,"sest",a2,7
90 tisk "Klokani jsou mazani",10
```

Procedura vytiskne libovolný počet parametrů libovolného druhu.

R E K U R Z E

V definici procedury je možné volat opět tutéž proceduru. Procedura volá sama sebe. Příklad:

```
100 DEF PROC diamant x,y,velikost,rozdil
      DEFAULT rozdil=15
      PLOT x,y-velikost
      DRAW -velikost,velikost
      DRAW velikost,velikost
      DRAW velikost,-velikost
      DRAW -velikost,-velikost
110   IF velikost>4 THEN
        diamant x,y+velikost,velikost-rozdil
        diamant x,y-velikost,velikost-rozdil
        diamant x-velikost,y,velikost-rozdil
        diamant x+velikost,y,velikost-rozdil
120 END PROC
```

A můžete si vyzkoušet napsat diamant 128,88,40.

S T R U K T U R O V A N É P R O G R A M O V Á N Í

Nové prostředky:

a) Procedury: DEF PROC, END PROC, LOCAL, REF, DEFAULT, ITEM
b) DO, LOOP, EXIT IF, WHILE, UNTIL funguje jako REPEAT a WHILE v Pascalu, ale jsou mnohem pružnější.
c) IF - THEN - ELSE
d) LIST FORMAT - vypisuje programy ve strukturovaném tvaru.

ZJEDNODUŠENÝ PŘÍSTUP NA PERIFERNÍ ZAŘÍZENÍ

Nové prostředky:

DEFAULT (SAVE/LOAD), SAVE, MERGE, MOVE, funkce LENGTH.

Při práci s drajvem je možné vynechat *m a lze tedy použít příkaz:

LOAD 1;"PROGRAM"

Pokud je ještě DEFAULT=m1, stačí pro uložení programu na Microdrive i příkaz SAVE "Program".

SAVE a VERIFY umožňují práci i s částí programu. Lze samostatně uložit i oblast proměnných.

MERGE může z Microdrive nahrát i samostartující programy.

MOVE je schopné přenést kromě datových souborů i programy, CODE a pole.

Funkce LENGTH vrací kromě rozměrů i adresu pole, takže je ožné manipulovat s polem pomocí LOAD "jméno" CODE.

PŘÍSTUP K DATŮM - POLE, TŘÍDĚNÍ, PROHLEDÁVÁNÍ

Nové prostředky:

JOIN, COPY, DELETE, SORT, EDIT proměnná, SAVE DATA, USING

Prohledávací funkce:

INARRAY, INSTRING

Další funkce:

LENGTH, CHAR$, NUMBER, EOF, SHIFT$, USING$

JOIN a COPY umožňují pole a nebo jejich části spojovat a mísit. Rozměry pole je možné měnit bez ztráty dat.

DELETE vymaže pole nebo jeho část.

SORT pole třídí.

INARRAY, INSTRING velmi rychle prohledávají pole a řetězce.

Pomocí EDIT je možné měnit obsah proměnných.

Pomocí SAVE DATA lze uložit proměnné na pásek nebo drajv.

USING a USING$ formátují výstup dat.

Funkce EOF informuje o tom, zda byla z Microdrive přečtena všechna data.

G R A F I K A

Nové prostředky:

ALTER, CONTROL CODES, CSIZE, DRAW TO, FILL, GET, OVER 2, PLOT, POKE, ROLL, SCROLL, WINDOW

Funkce:

SINE, COSE, FILLED, MEMORY$, SCRN$

Zvláštní proměnné:

xos, yos, xrg, yrg

E D I T A C E P R O G R A M U

Nové prostředky:

ALTER, AUTO, DEF KEY, DELETE, LIST ... TO ..., LIST DATA, LIST VAL, LIST VAL$, LIST DEF KEY, LIST PROC, LIST REF, REF, RENUM

Funkce:

MEM()

S Y N T A X E P Ř Í K A Z Ů

Nepovinné parametry nebo části příkazů jsou uzavřeny do hranatých závorek []. Znak '!' zastupuje nebo.

<CS>	= Caps Shift	<CS>+<SPACE>	= <BREAK>
<SS>	= Symbol Shift	#výraz	= numerický
<CS>+<SS>	= <Extended>	$výraz	= řetězcový
<CS>+<9>	= <Graphics>		

ALTER [popis-atributů] TO popis-atributů <GRAPHICS>,<A>
ALTER reference TO reference
==========

1) Umožňuje manipulaci s atributy obrazovky. Jednodušší forma změní atributy celé obrazovky bez vymazání textu. Složitější forma změní jen ty atributy, které odpovídají podmínkám prvního popisu atributů (může jich být několik).

Položky, které lze v popisu atributů použít jsou INK, PAPER, FLASH a BRIGHT.

Změna atributů bez vymazání obrazovky:

```
100 PRINT AT 10,10;"TEST"
    PAUSE 50
    ALTER TO PAPER 4
```

Možné jsou i kombinace:

```
ALTER TO PAPER 2, INK 7, FLASH 1
```

Nechceme-li změnit vše, ale jen něco, napíšeme:

```
ALTER INK 7 TO INK 0
```

Všechno, co bylo napsáno bílým inkoustem, bude napsáno černým. Vyzkoušejte:

```
ALTER INK 3, BRIGHT 1, PAPER 7 TO INK 5, FLASH 1
```

Následující program vám ukáže na obrazovce šachovnici.

```
100 LET A=2,B=4
110 FOR L=1 TO 5
120   FOR N=1 TO 16
        PRINT INK A; PAPER B;"XXXX"; PAPER A; INK B;"0000";
      NEXT N
130   LET TEMP=A,A=B,B=TEMP
    NEXT L
140 LET T=30
150 ALTER INK A TO INK B
    PAUSE T
160 ALTER PAPER A TO INK A
    PAUSE T
170 ALTER INK A TO PAPER B
    PAUSE T
180 ALTER INK B TO PAPER A
    PAUSE T
190 LET T=T-T/10-1
    GOTO 150
```

Poznámka: Položkou seznamu popisu atributů nemůže být atribut INVERSE.

2) Výskyt první reference je v celém programu nahrazen druhou referencí.

a) ALTER a$ TO b$	Změní v programu a$ na b$.
b) ALTER sum TO s	Změní všechny výskyty jména proměnné sum na s.
c) ALTER 1 TO 23	Změní 1 na 23, včetně 5-bajtové reprezentace.
d) ALTER "stary" TO "novy"	Změní řetězce i uvnitř uvozovek.
e) ALTER (s$) TO "řetězec"	Změní přiřazení v programu pro všechny proměnné jejichž aktuální obsah je roven obsahu s$. Skutečný obsah proměnných bude nezměněn.

AUTO [číslo řádku][,krok] <GRAPHICS>,<6>
==========

Jsou automaticky generována čísla řádků při vkládání programu. Pokud nejsou zadány hodnoty je použito 10. Pokud je krok>číslo řádku, je příkaz ignorován. Příklady:

```
AUTO       od řádku 10 s krokem 10
AUTO 100   od řádku 100 s krokem 10
AUTO 100,5 od řádku 100 s krokem 5
```

Automatické číslování řádek je ukončeno dosažením hodnoty 9995, nebo stiskem <BREAK> na dobu asi 1s, nebo vymazáním čísla řádky a např. LIST.

BREAK <BREAK>
=====

BREAK v BB03 funguje i pro rutiny ve strojovém kódu. Pro zastavení programu je třeba stisknout <BREAK> na dobu asi 1s. Poznámka: BB03 pracuje v režimu IM2 (Interrupt mode 2).

CLEAR bajty <CLEAR>
==========

Pro bajty<767 se RAMTOP posune o zadané číslo dolů. Je možné zadat i záporné číslo. Tento posun nevlivňuje obrazovku, proměnné a zásobník GO SUB, DO, PROC ...

Definice funkčních kláves a WINDOW se rovněž posunou.

CLOCK číslo|řetězec <GRAPHICS>,<C>
==========

Umožňuje uživateli přístup k 24-hodinovým hodinám řízeným přerušením, které mohou být na požádání zobrazeny v pravém horním rohu obrazovky. Na určitý čas lze nastavit alarm po jehož dosažení může podle vaší volby zaznít zvukový signál nebo dojít ke skoku do podprogramu.

! UPOZORNENI !

Při provádění BEEP, přístupů na periferie a strojových programů, při nichž je zakázáno přerušení hodiny stojí.

a) CLOCK n - nastavení režimu

n	Alarm GO SUB	Alarm BEEP	Zobrazení
0	-	-	-
1	-	-	ano
2	-	ano	-
3	-	ano	ano
4	ano	-	-
5	ano	-	ano
6	ano	ano	-
7	ano	ano	ano

Po načtení BB03 jsou hodiny v režimu 0.

b) CLOCK "HH:MM:SS" - Nastavení času
Dvojtečky lze vynechat, je akceptováno pouze prvních šest číslic a vše ostatní kromě 'a' a 'A' je ignorováno. Pokud např. zadáte CLOCK "xyz10" bude to interpretováno jako CLOCK "10:00:00".

c) CLOCK "aHH:MM:SS" - Nastavení alarmu
Při dosažení nastaveného času vydá počítač v režimech 2,3,6 a 7 krátkou sérii 'bipů'. V režimech 4-7 skočí do podprogramu, který začíná na řádku 8-9999, nebo je to první řádek za CLOCK. CLOCK "a 06:20" nastaví alarm na 6 hodin 20 minut.

d) CLOCK číslo řádku nebo
CLOCK: příkaz: příkaz: příkaz: ... : RETURN
Při dosažení času se dokončí prováděný programový řádek a přejde se do podprogramu.
Režim hodin lze ovlivňovat jednak na adrese 56866, kde můžeme stanovit kolik padesátin sekundy je rovno jedné sekundě - lze tedy změnit rychlost hodin. Na adrese 56870 je 54 má-li minuta 60 sekund a 58, zatoužíte-li, aby minuta měla 100 sekund.

Následující příklad ukazuje, jak lze tohoto příkazu použít- bude každou minutu číst data z portu 127.

```
8999 STOP
9000 PRINT "Běží alarmová rutina"
9010 LET cislo=PEEK 27000
     POKE cislo+27000,IN 127
9020 LET cislo=cislo+1
     IF cislo>100 THEN
       LET cislo=0
```

```
9030 POKE 27000,cislo
     LET z$=TIME$()
9040 LET hod=VAL z$(1 TO 2),min=VAL z$(4 TO 5)
9050 LET min=min+1
     IF min=60 THEN
       LET hod=hod+1, min=0
9060 CLOCK "a"+STR$ hod+STR$ min
     RETURN
```

Po vložení tohoto programu zadejte přímý povel:

```
CLEAR 26999: POKE 27000,0: CLOCK 9000: CLOCK 5
```

Tím se vytvořilo místo pro ukládání čísel a první uložená hodnota byla nula. Na řádku 9000 začíná podprogram ošetření alarmu. Teď můžete nastavit čas na nějakou hodnotu a pomocí CLOCK "a####" nastavit čas, ve kterém má být zahájeno čtení dat z portu 127 a zápis těchto hodnot do paměti od adresy 27001 pro další použití nebo zpracování. Podprogram může uložit jen 100 čísel, ale berte jej jen jako příklad. Podprogram můžete použít i k jiným účelům - k pravidelnému provádění určité činnosti.
Poznámka: Mnohá přídavná zařízení zatěžují datové linky Spectra tak, že program při zapnutých hodinách, nebo jiné interrupt rutině vypadává. Pak se musíte rozhodnout - buď CLOCK a nebo periférie.

CLS [číslo okna] <CLS>

Vymaže se aktuální okno (WINDOW). Pokud je zadáno číslo, vymaže se okno tohoto čísla, pokud bylo definováno. Není li okno definováno, přijde chybová zpráva "J Invalid I/O device".
CLS 0 vymaže vždy celou obrazovku bez ohledu na právě aktuální okno.

CONTROL CODES

Jde o zvláštní řídící znaky, které se na obrazovce přímo nezobrazují, ale uskutečňují na ní různé akce.

ŘÍZENÍ KURZORU:

CHR$ 2	KURZOR VLEVO	Platí pro aktivní okno
CHR$ 3	" VPRAVO	"
CHR$ 4	" DOLU	"
CHR$ 5	" NAHORU	"
CHR$ 8	" VLEVO	Nezávisle na aktivním okně
CHR$ 9	" VPRAVO	"
CHR$ 10	" DOLU	"
CHR$ 11	" NAHORU	"
CHR$ 12	Výmaz znaku	"
CHR$ 15	Speciální ENTER (CR/LF)	"

ŘÍDÍCÍ ZNAKY PRO OBRAZOVÉ BLOKY:

CHR$ 0	Následuje 8 bajtů grafické informace
CHR$ 1	Následuje 1 bajt atribut a 8 bajtů grafické informace (viz příkaz GET)

COPY pro řetězcová a numerická pole <COPY>
==========

Příkaz je natolik podobný příkazu JOIN, že je popsán tam.

CSIZE šířka [,výška] <GRAPHICS>,<CS>+<8>
==========

Příkaz řídí velikost písma pro příkazy PRINT, PLOT a LIST. Hodnoty jsou uváděny v pixlech. Normální hodnota je 8,8.

Hodnota 256,176 zobrazí na obrazovce jediný znak. Pokud zadáte jen šířku, je velikost znaků změněna v obou směrech stejně. Při malé rozteči je nutné přejít do režimu OVER 1, aby se znaky nepřepisovaly. Pak lze dosáhnout až hustotu 85 znaků na řádek, ale moc čitelné už to není. Ještě dobře čitelná je velikost CSIZE 4,8. Velikost znaků odpovídá Taswordu.

Tisková rutina BB03 je velmi flexibilní, může tisknout na libovolnou pixlovou pozici. Příkazem CSIZE 0 můžete přejít do rychlejší rutiny v ROM, ale ta neumí zpracovávat řídicí kódy CHR$ 0 a CHR$ 1.

Všechny obvyklé možnosti řízení pozice tisku ('AT','TAB', ',' atd.) jsou automaticky korigovány na platnou velikost CSIZE.

Znaky UDG jsou pro nastavenou šířku menší než 6 pixlů zobrazeny jen jako pravé poloviny.

Obrazové bloky přebírané příkazem GET jsou přebírány po blocích 8x8 pixlů. Nastavená velikost CSIZE je tedy významná. Je třeba dodržovat celé násobky (4,8,16,24, atd.), aby souhlasily hrany.

CSIZE lze použít lokálně jako položku příkazu PRINT např.:

```
PRINT CSIZE 16;"velké"
```

vytiskne text znaky o dvojnásobné výšce a šířce a další výstupy jsou opět podle trvale nastavené velikosti.

DEFAULT proměnná=hodnota [,proměnná=hodnota] ...
DEFAULT = SAVE/LOAD zařízení <GRAPHICS>,<CS>+<2>
==========

1) Účinek je podobný příkazu LET. Používá se v případech, kdy dané proměnné nebyla přiřazena hodnota jiným způsobem. Další informace viz procedury.

2) Nastavuje zařízení implicitní pro příkazy SAVE a LOAD.

```
DEFAULT = m   Microdrive 1
DEFAULT = m2  Microdrive 2
DEFAULT = n5  Sít
DEFAULT = B   RS 232, kanál B
DEFAULT = t   Páska (standardně)
```

DEF KEY znak; řetězec [+";"] <GRAPHICS>,<CS>+<1>
DEF KEY znak: příkaz: příkaz: ... [:]
==========

Definice řetězce, který bude generovat klávesa 'znak'. Můžete si určit,zda má být příkaz nebo sekvence příkazů vykonána hned, nebo vložena do editačního řádku a provedena po stisku <ENTER>. Pokud je posledním znakem řetězce nebo sekvence příkazů ':', je příkaz zobrazen v editačním řádku a čeká na <ENTER>.

Aby byly nadefinované klávesy akceptovány,je nutné přejít do hvězdičkového režimu současným stiskem kláves <SS>+<SPACE>.

Funkční klávesy můžete kdykoliv předefinovat. Pokud vložíte prázdný řetězec, nebo je seznam příkazů prázdný, je tato klávesa zrušena.

DEF KEY ERASE zruší všechny definice funkčních kláves.

Definice kláves jsou uloženy nad RAMTOPem a jsou tedy chráněny před NEW. Ukládací rutina BB03 z řádku 1 ukládá i definice kláves. LIST DEF KEY všechny definice vypíše.

Příklady:

```
DEF KEY "a":PRINT DPEEK (23730):
<ENTER>
```

Po <SS>+<SPACE> se objeví ve vstupním řádku '*' a po následujícím stisku <a> se objeví opět ve vstupním řádku PRINT DPEEK(23730) a po <ENTER> se na obrazovce vytiskne aktuální hodnota RAMTOP.

```
DEF KEY "l": INPUT "Název souboru:";q$: LOAD q$
```

Po vstupu do hvězdičkového režimu a stisku klávesy <l> jste požádán o název souboru a pak je zahájeno čtení tohoto souboru z default LOAD zařízení.

DEF PROC jméno procedury [parametr][,REF parametr] ...
DEF PROC jméno procedury DATA <GRAPHICS>,<1>
==========

Podrobnosti viz kapitola o procedurách. Příkazem začíná definice procedury. Příkaz musí být prvním příkazem na řádku. (Předcházející mezery popř. kódy barev jsou povoleny.) Jméno procedury může být stejné jako jméno proměnné a nedojde ke kolizi. Může obsahovat libovolné znaky kromě klíčových slov a znaků UDG a musí začínat písmenem.

Za jménem následuje seznam parametrů, který může být prázdný nebo klíčové slovo DATA. V seznamu mohou být i jména polí (b() apod.), ale před nimi musí být uvedeno klíčové slovo REF.

DELETE [číslo řádku1] TO [číslo řádku2] <GRAPHICS>,<7>
DELETE jméno pole ([#výraz] TO [#výraz])

1) Vymaže úsek programu od řádku1 po řádek2. Není-li číslo prvního řádku uvedeno,dosadí se číslo prvního programového řádku většího než 0. Není-li specifikováno číslo řádku2, je uvažován poslední řádek programu.

DELETE TO vymaže celý program kromě řádku 0, hodnoty proměnných jsou zachovány.

Pokud chcete z nějakého důvodu vymazat řádek 0, obsahující definice funkcí BB03, napište DELETE 0 TO 0.

Zadané řádky musí existovat, jinak je hlášena chyba. Příkaz může být použit i v programu. Nedoporučuje se jeho použití v podprogramech a procedurách. Není vhodný ani uvnitř smyček, protože by došlo k posunu programových řádek vzhledem k absolutním adresám v RAM a např. v případě smyčky FOR dostanete po příkazu NEXT chybové hlášení '1 NEXT without FOR', popř. 'Statement lost'.

Příklady:

```
DELETE TO 200          - vymaže všechny řádky větší než
                         0 až 200 včetně
DELETE 100 TO          - vymaže řádku 100 a všechny větší
DELETE 100 TO 100      - vymaže pouze řádku 100
DELETE 20 TO 80        - vymaže uvedený interval řádek
                         programu včetně uvedených
```

Příkaz je výhodný např. pro vymazání bloku dat po jejich přečtení - uvolní se tím paměť.

Má-li příkaz DELETE vymazat také sám sebe, musí být posledním příkazem mazaného bloku.

2) Příkaz vymaže část numerického nebo řetězcového pole, popř. celé. Retězec pak nemá délku 0, ale je zrušen úplně.

Příklad:

```
10 DIM a(10,10)
20 FOR i=1 TO 10
     FOR j=1 TO 10
       LET a(i,j)=i
       PRINT USING "###";a(i,j)
     NEXT j
     PRINT
   NEXT i
30 PRINT
   DELETE a(1 TO 5)
   FOR i=1 TO LENGTH(1,"a()")
     FOR j=1 TO 10
       PRINT USING "###";a(i,j);
     NEXT j
     PRINT
   NEXT i
```

DO <GRAPHICS>,<D>
DO WHILE podmínka <GRAPHICS>,<J>
DO UNTIL podmínka <GRAPHICS>,<K>
==========

Viz také LOOP, EXIT IF. DO a LOOP umožňují používat cykly bez potřeby skoků.

a) DO ... LOOP - nekonečná smyčka
b) DO WHILE podmínka ... LOOP
- smyčka se provádí jen do té doby, dokud je podmínka pravdivá. V opačném případě program pokračuje za LOOP.
c) DO UNTIL podmínka ... LOOP
- smyčka se provede jen v případě, že podmínka není splněna. V opačném případě se pokračuje za LOOP.

Adresa každého DO je uložena do zásobníku, proto se nesmí cyklus opouštět pomocí GOTO. Je možné použít buď EXIT IF nebo POP s následným GOTO. Lze vytvářet cykly, které jsou řízeny až třemi podmínkami. Např.

```
DO WHILE a<100
  ...
EXIT IF x=0
  ...
LOOP UNTIL y$="N"
```

DPOKE adresa, číslo <GRAPHICS>,<P>
==========

Viz také funkci DPEEK. DPOKE je dvojitý POKE a ekvivalent v Sinclair BASICu vypadá následovně:

```
POKE adresa, číslo-INT(číslo/256)*256
POKE adresa+1, INT(číslo/256)
```

Dolní bajt se uloží na adresu a horní na adresu+1. Je možné zpracovávat čísla v intervalu 0-65535. Pomocí tohoto příkazu lze snadno měnit hodnoty dvoubajtových systémových proměnných.

DRAW TO x,y [,úhel] <DRAW>,<SS>+<F>
==========

Na rozdíl od původního relativního DRAW, nakreslí čáru z poslední pozice PLOT do bodu x,y.

EDIT [číslo řádku] <0> nebo <GRAPHICS>,<CS>+<5>
EDIT řetězcová proměnná nebo <0>,<ENTER>
EDIT: numerická proměnná
==========

1) Příkaz neodpovídá staré funkci klávesy <EDIT>, i když původní funkce je stále přístupná. Je možné uvést číslo řádku, který má být editován. Není-li číslo uvedeno, je editován řádek s programovým kurzorem.

2) a 3) Příkaz umožňuje editovat hodnoty proměnných. Aktuální hodnota se přesune do editační zóny a po případné změně, je opět uložena zpět. Syntaxe EDIT je podobná příkazu INPUT, alespoň pokud se týče řídících znaků a případného textu.

Příklad:

```
10 LET a=10, a$="klokan"
   PRINT a,a$
   EDIT "Oprav číslo: ";a
   EDIT "Oprav text: ";a$
   PRINT a,a$
```

ELSE příkazy <GRAPHICS>,<E>
==========

Příkaz je součástí struktury IF-THEN-ELSE. Pokud je podmínka za IF nepravdivá, neprovedou se příkazy za THEN, ale pokračuje se za nejbližším ELSE. Před ELSE musí být ':'. Pokud je ELSE použito bez IF funguje stejně jako REM. Rozhodovací struktury lze vnořovat do sebe.

Příklady:

```
100 IF a=10 THEN
      PRINT a
      IF b=0 THEN
        PRINT "Nulou nelze dělit"
      ELSE
        PRINT a/b
110 IF c<10 THEN
      LET c=10
      IF r/2>c THEN
        BEEP 1,1
      ELSE
        BEEP 1,-10
    ELSE
      PRINT c
120 IF pocet <=5 THEN
      PRINT "Nedostatek dat"
    ELSE
      FOR i=1 TO LENGTH(1,"a()"(
        PRINT a(i)
      NEXT i
      INPUT "Pokračovat? ";q$
      IF SHIFT$(1,q$)="A" THEN
        GOTO 20
      ELSE
        STOP
```

END PROC <GRAPHICS>,<3>
========

Viz také kapitola a procedurách. Příkaz ukončuje definici procedury. Je-li při běhu programu nalezeno DEF PROC a procedura nebyla volána, je celá procedura přeskočena a pokračuje se až za END PROC. Pokud přijde END PROC a nebylo DEF PROC, dostanete chybovou zprávu "W Missing DEF PROC".

EXIT IF <GRAPHICS>,<I>
=======

Viz také DO ... LOOP. Příkaz je součástí struktury DO ... LOOP. Je-li podmínka splněna pokračuje se za odpovídajícím LOOP. Příkaz vybírá adresu ze zásobníku DO.

Příklad:
```
100 DO
      PRINT "Rádek 100 "
      PAUSE 20
110 EXIT IF INKEY$="s"
120   PRINT "Rádek 120"
      PAUSE 20
    LOOP
130 PRINT "Mimo smyčku"
```

Smyčka je po stisku klávesy "s" opuštěna. Pokud použijete EXIT IF mimo smyčku, dostanete chybovou zprávu "S Missing LOOP".

FILL x,y <GRAPHICS>,<F>
FILL INK barva;x,y
FILL PAPER barva;x,y
==========

Viz také funkci FILLED()

1) a 2) Vyplní uzavřenou plochu bodů barvy PAPER barvou INK. Pokud má tato oblast již barvu INK, nic se nestane.
3) Vyplní souvislou plochu barvou PAPER.

Je třeba respektovat omezení grafiky Spectra. 8x8 pixlů může mít jen dvě barvy (PAPER a INK).

Příklad:
```
10 DEF PROC kruznice
20   CIRCLE 128,88,50
30 END PROC
40 kruznice
   FILL 128,88
   PAUSE 150
50 CLS
   kruznice
   FILL INK 5;128,88
   PAUSE 150
60 CLS
   kruznice
   FILL 128,88
   FILL PAPER 7;128,88
```

Počet pixlů použitých pro vyplnění plochy lze zjistit funkcí FILLED().

GET #proměnná!$proměnná <GRAPHICS>,<G>
GET $proměnná,x,y [,šířka,výška][,typ]
==========

1) GET čte znak z klávesnice. Ceká na stisk klávesnice a nečeká na <ENTER>. Pokud je uvedena řetězcová proměnná, je načten znak. Do numerické proměnné jsou vložena čísla 0-9 a pak A=10, B=11, atd.

Příkaz má využití u programů s menu řízením, zvláště ve spojení s příkazem ON.

2) Z obrazovky je do řetězcové proměnné přečtena pravoúhelníková oblast se souřadnicemi levého horního rohu x,y. Šířka a výška je udána v print pozicích, x a y jsou standardní souřadnice PLOT. Takto uloženou část obrazovky lze zobrazit na jiném místě pomocí příkazů PRINT nebo PLOT.

Tento řetězec se nedá vytisknout při CSIZE 0, ale rozměry lze měnit použitím příkazu CSIZE.

Základní jednotka řetězce se skládá z 9 znaků. První je řídící kód CHR$ 0 a udává, že dalších 8 bajtů je kódováno jako grafika. V případě větších bloků jsou automaticky vkládány kódy řízení kurzoru.

Při OVER 0 se obrazovka vzorem přepisuje, při OVER 1 a OVER 2 vznikají jiné efekty.

Pokud není uveden typ, pak se implicitně předpokládá typ 0, t.j. pouze pixly. Obrázek pak má aktuální barvy INK a PAPER. Je-li deklarován typ 1 jsou při GET uloženy i atributy a jsou tedy při následujícím tisku zachovány.

Mějte na paměti omezení grafiky Spectra.

Příklady:

```
10 PRINT AT 20,0;"++";AT 21,0;"++"
20 GET a$;0,15,2,2
30 PRINT CSIZE 8;AT 0,0;a$
40 PRINT CSIZE 16;AT 1,1;a$
50 PRINT CSIZE 24,16;AT 1,1;a$
60 PLOT CSIZE 40; 128,88;a$
```

nebo:

```
10 CLS
   FOR i=1 TO 100
     LET x=RNDM(255),y=RNDM(175)
     DRAW TO x,y
   NEXT i
   PRINT CSIZE 16;AT 5,3;"OBRAZOVKA"
20 GET b$,0,175,32,22
   CSIZE 8
30 DO
     CLS
     PLOT 0,175;b$
     PAUSE 150
   LOOP UNTIL INKEY$<>""
```

Druhý z těchto programů uloží do b$ celou obrazovku. Mechanismus není příliš hbitý. Pokud 20. řádek změníte na:

```
20 LET b$=MEMORY$()(16384 TO 22527)
```

a třetí příkaz 30. řádku na

```
POKE 16384,b$
```

funguje program stejně, ale mnohem rychleji.

JOIN [číslo řádku] <GRAPHICS>,<CS>+<6>
JOIN řetězec ! řetězcové pole ! numerické pole

Viz také SPLIT.

1) Příkaz spojí daný řádek s řádkem následujícím. Pokud chybí číslo řádku, vezme se řádek s programovým kurzorem. Touto operací se ušetří 4 bajty, zvýší se rychlost programu. Nezapomínejte však na logiku programu. Spojíte-li dva řádky:

```
100 IF a<1 THEN
      LET b=SIN a
110 LET c=123
```

pomocí JOIN 100, dostanete

```
100 IF a<1 THEN
      LET b=SIN a
      LET c=123
```

a to samozřejmě není totéž.

2) JOIN spojuje řetězce a pole, COPY je kopíruje. Syntaxe je ovšem stejná a tak oba příkazy probereme společně na tomto místě.

a) JOIN řetězec1 [[#výraz] TO [#výraz]] TO řetězec2 [#pozice]
   COPY řetězec1 [[#výraz] TO [#výraz]] TO řetězec2 [#pozice]

Přidá řetězec1 k řetězci2. Příklad:

```
10 LET a$="12345",b$="ABCDEFG"
20 JOIN a$ TO b$
30 PRINT b$
   REM vypíše ABCDEFG12345
40 PRINT a$
   REM a$ už neexistuje proto -> chybové hlášení
```

Pokud změníte v řádku 20 JOIN na COPY, zůstane řetězec a$ zachován.

b) JOIN pole1 [[#výraz] TO [#výraz]] TO pole2 [#pozice]
   COPY pole1 [[#výraz] TO [#výraz]] TO pole2 [#pozice]

Použitelné např. v případě, že se pole naplnilo a je třeba, aby bylo dimenzováno na větší rozměr. Příklad: máme pole a$(100,30), nestačí nám a potřebujeme přidat dalších 20 řetězců. Postup: DIM b$(20,30): JOIN b$ TO a$. Funkce LENGTH(1,"a$()") nám vrátí velikost prvního rozměru=120.

Vkládaný řetězec přizpůsobí svoji délku podle toho, kam je vkládán. Pokud je např. šířka 30 znaků málo, potom utvoříme pomocné pole s potřebnou délkou a přidáme k němu úzké pole:

```
DIM b$(1,40): JOIN a$ TO b$
```

Nyní znovu vytvoříme pole s původním názvem, ale potřebnou šířkou a přidáme k němu pomocné pole:

```
DIM a$(1,40): JOIN b$ TO a$
```

Nakonec ještě vymažeme nadbytečné řetězce:

```
DELETE a$(1 TO 2)
```

Upozornění: Zpracovávaná pole mohou mít maximálně dva rozměry.

Příklad:

```
10 DEF PROC vypis REF x$
    LOCAL i
    FOR i=1 TO LENGTH(1,"x$()")
      PRINT x$(i)
    NEXT i
   END PROC
20 DIM a$(10,10)
   DIM b$(10,10)
30 FOR i=0 TO 9
     LET a$(i+1)=STRING$(10,CHR$(i+65))
     LET b$(i+1)=STRING$(10,STR$ i)
   NEXT i
40 PRINT "a$()"
   vypis a$()
   PRINT "b$()"
   vypis b$()
   PAUSE 150
50 JOIN b$(3 TO 5) TO a$(5)
60 PRINT "a$()"
   vypis a$()
```

<GRAPHICS>,<CS>+<4>

KEYIN řetězec

Příkazem KEYIN a$ je do programu zaveden obsah a$ jako programový řádek. Tímto způsobem se může program sám modifikovat. Příklad: Potřebujete zapisovat velké množství položek příkazu DATA. Práci vám ulehčí následující program.

```
10 INPUT "DATA od řádku:";start,"krok:";krok
20 DO
30   LET a$=STR$ start+" DATA ": REM DATA=klíčové slovo
40   INPUT LINE b$
50 EXIT IF SHIFT$(1,b$)="KONEC"
60   LET a$=a$+b$
70   KEYIN a$
80   LET start=start+krok
90 LOOP UNTIL start>9999
```

Je nutné, aby data byla ukládána z a smyčkou DO, protože jinak by došlo ke ztrátě adresy skoku ve smyčce. Retězec dat vkládaný v řádku 40 musí samozřejmě splňovat syntaxi parametrů příkazů DATA.

Příkaz lze použít i k umístění testu mimo smyčku. Například máte proceduru, která má počítat jednoduchý a dvojný integrál. Ideový návrh řešení tohoto problému by s pomocí KEYIN mohl být následující:

```
330 IF integral=1 THEN
      KEYIN "350 'Výpočet jednoduchého integrálu'"
    ELSE
      KEYIN "350 'Výpočet dvojného integrálu'"
340 FOR i=start TO konec STEP krok
(350 REM Sem jsou řádkem 330 vloženy potřebné vzorce)
360 NEXT i
```

Pokud by nešlo použít KEYIN, musel by být test uskutečňován uvnitř smyčky - tedy při každém průběhu (to zabírá čas), nebo napsat dvě různé procedury, což zabírá paměť.

Příkaz INPUT "Heslo:";a$: KEYIN a$ umístěný na začátku programu by mohl představovat určitý způsob ochrany programu, protože nestačí znát jen heslo, ale je třeba znát i číslo řádku.

V režimech KEYWORDS 3 nebo 4 se klíčová slova zapsaná znaky převádějí na jednobajtová klíčová slova.

V každém případě buďte při použití tohoto příkazu opatrnost sama.

KEYWORDS číslo <GRAPHICS>,<8>

Číslo 0 a 1 řídí výstup pro PRINT a LIST. 2 a 4 řídí vkládání programu. Významy:

0 - Zobrazují se grafické znaky.
1 - Zobrazují se nová klíčová slova BB03.
2 - Klíčová slova je nutné vkládat jednou klávesou.
3 - Po vstupu je řádek otestován a všechna vypsaná slova jsou převedena na jednobajtová klíčová slova (tokens) Tento režim akceptuje oba druhy vstupu.
4 - Neexistuje kurzor 'K'. Všechny příkazy je třeba vypisovat po znacích.

LET proměnná=výraz [,proměnná=výraz] ... <LET>

Do jednoho příkazu LET je možno vložit víc přiřazení. Program se tím zrychlí.

LIST [číslo řádku] TO [číslo řádku] <LIST>
LLIST [číslo řádku] TO [číslo řádku] <EXTEND>,<V>

Vypíše zadaný úsek programu. Přitom je možné některou z hodnot vynechat. Použití je podobné jako při DELETE.

LIST DATA -> Všechny proměnné <LIST>,<EXTEND>,<0>
LIST VAL -> Číselné proměnné <LIST>,<EXTEND>,<J>
LIST VAL$ -> Retězcové proměnné <LIST>,<EXTEND>,<CS>+<J>
==========

Vypíše se přehled aktuálních proměnných oblasti VARS a jejich obsahy. Proměnné vypíše v pořadí:

1 - numerická pole
2 - řídící proměnné smyček FOR
3 - proměnné s jednoznakovým jménem
4 - proměnné s víceznakovým jménem

Dále se vypíší:

5 - řetězcová pole
6 - řetězcové proměnné

U řídících proměnných je uveden i řádek jejich posledního použití. Pro pole se vypíší jen rozměry, nikoliv obsah. Pro řetězce je vypsáno jen prvních 15 znaků.

LIST DEF KEY <LIST>,<GRAPHICS>,<CS>+<1>
==========

Příkaz vypíše definice všech definovaných funkčních kláves.

LIST FORMAT číslo <LIST>,<EXTEND>,<CS>+<0>
==========

Příkaz nastavuje režim, ve kterém bude prováděn výpis programu. Po načtení je nastaven režim 0. Význam čísel:

0 - Klasický výpis Spectra, s tím rozdílem, že řádky delší než 32 znaků pokračují na dalším řádku až od pátého sloupce. Příklad:

```
10 PRINT "Smyčka začíná.": FOR
    i=1 TO 10: PRINT i, SIN i:
   NEXT i
20 PRINT "Smyčka ukončena, kon
   ec programu."
```

1 - Pro vnořené struktury je prováděn odskok o jeden sloupec vpravo. Není zobrazena ':' oddělující příkazy, je tiskovou rutinou interpretována jako nový řádek (CR) a proto je každý příkaz na samostatném řádku obrazovky. Příklad:

```
10 PRINT "Smyčka začíná."
   FOR i=1 TO 10
    PRINT i,SIN i
   NEXT i
20 PRINT "Smyčka ukončena, kon
   ec programu"
```

2 - Platí totéž, co pro 1, ale odskok je prováděn o dva sloupce vpravo.

3 - Jako 0, ale nejsou zobrazena čísla řádků.

4 - Jako 1, ale bez čísel řádků.

5 - Jako 2, opět bez čísel řádků.

BB03 obsahuje tak bohaté prostředky, podporující strukturované programování, že čísla řádků skutečně ani nepotřebujete a můžete psát zcela pascalupodobné programy.

Strukturu lze zdůraznit i u příkazů IF...THEN...ELSE. Výpis bude přehlednější, když napíšete dvojtečku za každým THEN a ELSE

LIST PROC jméno <LIST>,<GRAPHICS>,<2>

Vypíše text požadované procedury. Po provedení výpisu je na adrese 23625 uloženo číslo prvního řádku procedury a na adrese 57358 je uloženo číslo posledního řádku. To lze využít k řízenému DELETE nebo např.:

```
 10 DEF PROC BINDEC B$, REF DEC
 20   LOCAL X,I
 30   LET X=0
 40   FOR I=1 TO LEN B$
        IF B$(I)="1" THEN
          LET X=X+2^(LEN B$-I)
 50   NEXT I
      LET DEC=X
    END PROC
100 LIST PROC BINDEC
110 LET START=DPEEK(23625),KONEC=DPEEK(57358)
120 PRINT START,KONEC
130 SAVE START TO KONEC;"PROCBINDEC"
```

LIST REF údaj <LIST>,<GRAPHICS>,<CS>+<7>

Příkaz vypíše čísla řádků, na kterých se reference údaj nachází. Může to být jméno proměnné, číslo, sekvence znaků.

LOCAL proměnná [,proměnná] <GRAPHICS>,<CS>+<3>

Vytvoří se speciální proměnné, které jsou známy jen v dané proceduře. Formální parametry jsou pro proceduru automaticky lokální. V proceduře se může vyskytovat několik příkazů LOCAL. Má-li být lokální pole, musí být deklarováno sekvencí příkazů:

```
LOCAL p(): DIM p(,,)
```

LOOP <GRAPHICS>,<L>
LOOP WHILE podmínka <GRAPHICS>,<L>,<GRAPHICS>,<J>
LOOP UNTIL podmínka <GRAPHICS>,<L>,<GRAPHICS>,<K>

Příkaz ukončuje strukturu DO - LOOP. Podobně jako u DO je možné použít příkazy pro podmíněné ukončení cyklu.

MERGE [mechanismus:] "jméno" <EXTENDED>,<CS>+<T>
==========

Lze uvést zařízení (viz DEFAULT). Funguje MERGE z drajvu.

MOVE <EXTENDED>,<CS>+<6>
====

Na drajv lze přenášet programy i soubory CODE.

GO TO ON výraz: řádek1, řádek2, ... <GO TO>,<GRAPHICS>,<O>
GO SUB ON výraz: řádek1, řádek2, ... <GO SUB>,<GRAPHICS>,<O>
ON výraz: příkaz: příkaz: ... příkaz <GRAPHICS>,<O>
==========

1) Skáče se na řádek, jehož číslo je na pozici odpovídající hodnotě výrazu, jehož celočíselný výsledek musí být v rozmezí <1..254>.
2) Skáče se do podprogramu na daném řádku, za stejných podmínek jako v bodě 1.
3) Provede se příkaz, jehož pořadové číslo je rovno hodnotě výrazu.

Pokud je hodnota výrazu mimo počet řádků nebo příkazů, pokračuje program následujícím příkazem nebo řádkem.
Příklad:

```
10 DEF PROC hodnocení chyby
20   LET chyby=chyby+1
30   ON chyby
       PRINT "Výborně"
       PRINT "Chvalitebně"
       PRINT "Dobře"
       PRINT "Dostatečně"
       PRINT "Nedostatečně"
40   IF chyby>=6 THEN
       PRINT "Kopyto"
50 END PROC
```

ON ERROR číslo řádku <GRAPHICS>,<N>
ON ERROR : příkaz: příkaz ...
==========

1) V případě chyby (kromě 0 OK a 9 STOP) je proveden skok do podprogramu na daném řádku. Příklad:

```
  10 INPUT "n=";n
  20 ON ERROR 1000
  30 PRINT "100/";n;"=";100/n
  40 ON ERROR 0
  50 GO TO 10
1000 IF lino=20 AND error=6 THEN
       IF n=0 THEN
         PRINT"Neumím dělit nulou."
1010 RETURN
```

2) Provádějí se příkazy na řádku. Pokud není chyba, jsou přeskočeny.

Vlastní zpracování je během podprogramu ošetřujícího chyby vypnuto a obnovuje se při RETURN. CONTINUE vrací program na řádek, který způsobil chybu, takže chyba se vypíše a program se zastaví. Před CONTINUE je ovšem nutné vybrat návratovou adresu ze zásobníku GOSUB pomocí POP.

ON ERROR 0 zpracování chyb vypíná.

Dojde-li k chybě, vytváří BB03 tři nové proměnné:

lino = číslo řádku s chybou
stat = pořadové číslo příkazu v řádku 'lino'
error= číslo chyby (viz příloha C)

Proměnné stejného jména se mohou používat i ve vašem programu, ale je-li aktivní ON ERROR nebo TRACE, mohou se přepsat.

Rutiny pro ošetření chyb je třeba používat s rozumem, protože by se snadno mohly rozrůst tak, že by byly rozsáhlejší než vlastní program. Příklad:

```
100 ON ERROR 5000
110 FOR n=1 TO 10
      INPUT "x=";x,y=";y
120   PLOT x,y
    NEXT n
4999 STOP
5000 IF error=11 AND lino=120 THEN
      RETURN
    ELSE
      POP
      CONTINUE
```

Příkaz STOP brání nechtěnému použití POP. Testujeme i číslo řádku, protože chyba "Integer out of range" je častou chybou. My však chceme zachytit vznik této chyby pouze na řádku 120. Pokud je hodnota x nebo y mimo povolený rozsah, pak RETURN na řádku 5000 skočí za příkaz, který chybu způsobil, tedy na NEXT n.

Objeví-li se v programu jiná chyba, nebo dojde k chybě "Integer out of range" na jiném řádku než 120, vrací POP řízení do hlavního programu a CONTINUE nechá příkaz způsobivší chybu zopakovat, aniž by bylo zapnuto sledování chyb. Program se tentokrát zastaví a je vydána odpovídající chybová zpráva.

Chybu "BREAK into program" je třeba ošetřit jinak. Řekněme, že chcete vyvolat určitou akci tím, že na krátkou dobu stisknete <BREAK>. Ošetření chyb se vypne, jakmile dojde ke skoku do chybové rutiny. Pokud stále ještě držíte <BREAK>, program se zastaví a to nechceme. Řešení je následující: první příkaz chybové rutiny bude téměř neslyšitelný BEEP, který nelze pomocí BREAK stopnout a tím vám umožní zvednout prst(y) z klávesnice:

```
  100 ON ERROR 5000
  110 PRINT "ještě jednou, a"
      PAUSE 10
      GOTO 110
 4999 STOP
 5000 IF error=21 THEN
        BEEP 1,55
        BORDER RND(7)
        RETURN
      ELSE
        POP
        CONTINUE
```

Stisknete-li <BREAK> na dobu kratší než 1s, změní se náhodně barva okraje obrazovky, je-li stisk <BREAK> delší než 1s program se zastaví.

OVER číslo <EXTEND>,<CS>+<N>
==========

Příkaz akceptuje jako parametr čísla 0, 1 a 2. První dva mají týž význam jako v původním Sinclair BASICu. Parametr 2 má za následek, že nový tisk se s původním ORuje. Znamená to, že znaky se nepřepisují ani nedochází ke XOR, jako v případě OVER 1. Příklad:

```
  10 DEF PROC znaky volba
       LOCAL i
  20   CLS
       OVER volba
       PRINT AT 21,0;"OVER ";volba
       CSIZE 80
       FOR i=1 TO 30
         PRINT AT 0,0;CHR$(RND(50)+33);
       NEXT i
       CSIZE 0
       OVER 0
  30 END PROC
 100 znaky 0
     PAUSE 0
     znaky 1
    PAUSE 0
     znaky 2
```

PLOT x,y;řetězec <PLOT>
================

Kromě obvyklého zobrazení bodu je možné na místě x,y zobrazit i řetězec. Může jít o normální řetězec nebo o řetězec vytvořený pomocí GET. x a y určují levý horní roh pozice, na kterou bude řetězec vytištěn.

Kromě INK a PAPER lze jako parametr použít i CSIZE ke změně velikosti tištěného řetězce. Výhodou příkazu je, že má stejný souřadnicový systém pro grafiku i text.

```
PRINT CSIZE 32; INK 2; 100,88;"AHOJ!"
```

nebo

```
10 FOR a=0 TO 2
     OVER a
20   FOR i=12 TO 100 STEP 4
       PLOT i,i;"AHOJ"
     NEXT i
30   OVER 0
     BEEP .5,a*10
     PAUSE 50
   NEXT a
```

Pomocí PLOT můžeme zadat mnohem jemnější pohyby, než je možné pomocí PRINT AT.

```
100 FOR x=16 TO 224
      PLOT x,x/2;"<>"
    NEXT x
```

Použijete-li STEP 2, 3 nebo více,dosáhnete větší rychlosti. Dovednost PLOT tisknout řetězce je velmi výhodná při popisu tabulek a diagramů. Popisy můžete, vzhledem ke společnému souřadnicovému systému, přesně umistovat.

```
100 LET a$="Na-2+SO-4+má na cm+3-kolik"+CHR$ 13+"molekul ?"
110 LET x=0,y=160
120 FOR c=1 TO LEN a$
130   IF a$(c)="-" THEN
        LET y=y-3
      NEXT c
140   IF a$(c)="+" THEN
        LET y=y+3
      NEXT c
150   IF a$(c)=CHR$13 THEN
        LET x=999
        GOTO 170
160   PLOT x,y;a$(c)
      LET x=x+8
170   IF x>=248 THEN
        LET x=0,y=y-12
180 NEXT c
```

V tomto příkladě jsou znaky '+' a '-' použity k řízení pozice PLOT - nemohou tedy být tisknuty. Na řádku 160 můžete změnit vzdálenost mezi znaky.

POKE adresa, řetězec <POKE>

Kromě čísla v intervalu 0..255 lze vkládat i řetězce. Ekvivalent PEEK v BB03 zastupuje v tomto případě funkce MEMORY$()([#výraz] TO [#výraz]). Příklad:

```
10 CLS
   PRINT CSIZE 64;"TEST"
20 POKE 18432,MEMORY$()(16384 TO 18431)
30 LET a$=MEMORY$()(16384 TO 18431)
40 LET a=LEN a$
   DIM b$(a)
   CLOCK "0"
   CLOCK 1
   PRINT AT 19,0;"Prosim o strpeni 35 s."
   LET b=a+1
   FOR i=a TO 1 STEP -1
     LET b$(b-i)=a$(i)
   NEXT i
50 POKE 18432,b$
   CLOCK 0
```

POP [#proměnná] <GRAPHICS>,<Q>
==========

Příkaz vybere návratovou adresu ze zásobníku (GO SUB, DO, PROC). Pokud je udána proměnná, uloží se do něj číslo řádku. Můžete vyskočit z podprogramů, cyklů a procedur, aniž by došlo ke zmatkům. Příklad:

```
100 GOSUB 500
110 STOP
500 POP odkud
510 PRINT "Podprogram je volán z řádku ";odkud
520 GOTO odkud+1
```

Pokud by byl na řádku 520 RETURN, dostali bychom chybovou zprávu '7 RETURN without GOSUB'. Pokud provedete POP a zásobník návratových adres je prázdný, je chybová zpráva 'V No POP data'.

[PROC] jméno [parametr][,parametr] ... <GRAPHICS>,<2>
==========

Příkaz volá proceduru. Je to obdoba GOSUB, ale nemusíte znát číslo řádku. Klíčové slovo PROC je možné vynechat. Podrobnější popis viz stat o procedurách. Aby překladač proceduru našel, musí být DEF PROC první příkaz na řádku. Tělo procedury může být definováno na libovolném počtu řádků a musí být zakončeno klíčovým slovem END PROC.

Ideální strukturovaný program obsahuje mnoho procedur, z nichž každá plní specifický úkol a odděleně od ostatních částí programu. Např.:

```
100 UKAZ HRACI POLE
110 HRA
120 UKAZ SKORE
    STOP
```

Pokud procedura nebyla definovana dostanete zprávu 'W Missing DEF PROC', tatáž zpráva se obvykle zobrazí i při použití END PROC bez příslušného DEF PROC (vyjímkou je případ, kdy je v zásobníku nějaká návratová adresa GOSUB nebo DO). Pokud zapomenete END PROC dostanete 'X No END PROC'.

READ LINE $proměnná [,$proměnná] ...
<EXTEND>,<A>,<EXTEND>,<CS>+<3>

Příkaz umožňuje přečíst do proměnných hodnoty, aniž by bylo nutné, aby řetězce v příkazu DATA byly v uvozovkách. Příklad:

```
10 FOR i=1 TO 3
     READ LINE a$
     PRINT a$
   NEXT i
20 DATA klokan,potkan,daman
```

REF reference <GRAPHICS>,<CS>+<7>
REF proměnná

1) Příkazem se prohledá celý program na výskyt dané reference. Reference je proměnná, číslo, sekvence znaků. Pokud je nalezen souhlas, je nalezený řádek přesunut do editační oblasti a editační kurzor je umístěn za nalezenou referencí. Pokud řádek nechcete opravit, je nutné stisknout <ENTER>. Pokud stisknete <ENTER> ještě jednou - hledání pokračuje. Když v době hledání vložíte nějaký příkaz (třeba LIST), je hledání přerušeno. Příklady:

```
REF a$     - hledá výskyt a$
REF sum    - hledá se proměnná _sum_ (_ = mezera)
REF "SUM"  - hledá se SUM
REF 1      - hledá se číslo 1, i jeho 5-ti bajtový tvar
REF "1"    - hledá se 1
REF 12*4   - hledá se uvedený součin včetně 5-ti bajtových
             tvarů
REF (a$)   - je hledán obsah a$
REF (x)    - hledá se obsah x včetně 5-ti bajtového tvaru
```

Při hledání nehraje velikost písmen roli.

2) V seznamu formálních parametrů určí proměnné, jejichž hodnota je předávána adresou. Znamená to, že jakákoliv změna obsahu proměnné uvnitř procedury bude známa i po jejím ukončení. Pole je možné předávat jen přes REF.

RENUM [*][[řádek1] TO [řádek2]][LINE start][STEP krok]
<GRAPHICS>,<4>

Celý program a nebo část programu je možné přečíslovat, přesunout úseky, nebo zkopírovat na jiné místo programu.

a) RENUM — Přečísluje celý program tak, že začíná na řádku 10, s krokem 10.
b) RENUM od TO do — Přečíslování úseku.
c) * — Původní blok se nesmaže, je jen překopírován na jiné místo.

Příklady:

```
RENUM LINE 100 STEP 20          Přečísluje program, takže je na
                                řádcích 100,120,140 ...
RENUM 100 LINE 300              Rádek 100 na 300
RENUM 1540 TO LINE 2000         Vše od 1540 se přesune na 2000 a
                                dále
RENUM 10 TO 100 LINE 1000 STEP 5
                                Rádky v intervalu 10 až 100 včetně
                                se přesunou na řádek 1000 s krokem
                                5.
```

Příkaz přečísluje všechny reference v GO TO, GO SUB, RESTORE, RUN, ON, ON ERROR, TRACE, LIST, LINE a DELETE. Případná čísla řádků v CLOCK je nutné změnit ručně. Pokud se stane, že číslo řádku v GO TO, GO SUB atd. je výraz, např:

```
100 GO TO a
```

je vypsána zpráva 'Failed at ..:..' (100:1)

Poznámka: RENUM využívá jako pracovní paměť oblast obrazovky. Umožňuje to přečíslovat i dlouhé programy. Pro nižší verze než 3.0 se doporučuje přečíslovávat jen je-li vypnuto zobrazení hodin.

ROLL kód směru [,pixly][;x,y;šířka,výška] <GRAPHICS>,<R>

Viz také SCROLL. Příkaz přesouvá celý nebo část obrazu o daný počet pixlů v zadaném směru. To co na jedné straně z obrazovky odejde, se objeví na druhé straně (na rozdíl od SCROLL).

Obraz se přesouvá rychleji, pokud zadáte větší počet pixlů. Nejrychlejší přesun je při přesunu ve vodorovném směru o 4 nebo 8 pixlů.

Kód směru	Směr	Pohybuje se
1	vlevo	atributy
2	dolů	atributy
3	nahoru	atributy
4	vpravo	atributy
5	vlevo	pixly
6	dolů	pixly
7	nahoru	pixly
8	vpravo	pixly
9	vlevo	obojí
10	dolů	obojí
11	nahoru	obojí
12	vpravo	obojí

Vzhledem k tomu, že se atributy mohou přesouvat jen po osmi pixlech, ignoruje se počet zadaných pixlů.

V případě přesunů části obrazů se zadává šířka v PRINT pozicích (1-32), výška v pixlech (1-176).

Pokud má k přesunu dojít pouze v aktuálním okně, stačí jen jednoduchá forma příkazu.

ROLL lze efektivně nasadit např. k nepřerušovanému pohybu hracích figur nebo pozadí.
Příklady:

```
100 LIST
110 ROLL 5;0.175;32,88
    ROLL 6;0,175;16,176
120 ROLL 8;0,87;32,88
    ROLL 7;128,175;16,176
130 GO TO 110
```

nebo

```
200 FOR n=1 TO 7
      LIST
    NEXT n
210 FOR l=1 TO 175
      ROLL 8;0,175;32,1
    NEXT l
```

SAVE [řádek1] TO [řádek2;][mechanismus;]"jméno" <SAVE>
SAVE DATA [mechanismus;]"jméno"
==========

Uložit je možné nejen celý program, ale i část programu, případně jen proměnné (DATA). Při čtení pomocí LOAD je třeba mít na paměti, že dojde k vymazání nejen celého programu i řádku 0.

Tímto způsobem můžete uložit na pásek různé procedury a vytvořit si tak knihovnu užitečných procedur. Je vhodné provést před uložením RENUM na vysoká čísla řádků a po načtení provést RENUM na požadované místo a tím si uvolnit místo pro načtení další procedury.

SCROLL [kód směru][,pixly][;x,y;šířka,výška] <GRAPHICS>,<S>
==========

Viz také ROLL. Při volání bez parametru se posouvá celý obraz o řádek nahoru. Pokud použijete kód 5-8, přesune se celé aktuální okno o 1 pixl. Informace vysunutá z okna se ztrácí a na druhém konci vstupují prázdné řádky nebo sloupce.

SORT #pole|$pole|řetězec <GRAPHICS>,<M>
SORT INVERSE #pole|$pole|řetězec
<GRAPHICS>,<M>,<EXTEND>,<CS>+<M>
==========

Příkaz uspořádá pole nebo řetězce ve vzestupném, nebo sestupném pořadí podle čísla nebo podle abecedy.

Pro pole je možné použít jeden nebo dva rozměry. Numerické hodnoty jsou pro SORT uspořádány od největšího prvku k nejmenšímu.

Podívejte se na následující program:

```
10 DEF PROC vypis
     LOCAL i,q$
     LET q$="####"
     FOR i=1 TO 20
       PRINT USING q$;a(i),USING q$;b(i)
     NEXT i
   END PROC
20 DIM a(20)
   DIM b(20)
   FOR i=1 TO 20
     LET a(i)=RNDM(1000),b(i)=a(i)
   NEXT i
   SORT a()
   vypis
   PAUSE 0
   CLS
   SORT INVERSE a()
   vypis
```

Další příklady: máme pole a(5,5) s následujícím obsahem:

Původní obsah:					Obsah po SORT a()(5):				
13	18	5	60	16	84	46	99	27	99
84	46	99	27	99	47	76	85	81	55
97	49	49	86	23	97	49	49	86	23
14	38	40	74	14	13	18	5	60	16
47	76	85	81	55	14	38	40	74	14

Řádky pole jsou setříděny sestupně podle hodnot obsažených v pátém sloupci pole.

Původní obsah:					Obsah po SORT a(2 TO 4)(3):				
5	64	40	47	23	5	64	40	47	23
12	5	25	71	43	76	32	94	89	78
76	32	94	89	78	3	44	83	76	87
3	44	83	76	87	12	5	25	71	43
77	61	75	97	8	77	61	75	87	8

Jsou sestupně setříděny řádky pole 2 až 4 podle hodnot ve sloupci 3.

Třídění řetězcových polí je podobné. Máme pole a$(5,5):

Původní obsah	1) SORT a$()()	2) SORT b$()(5)
m o u k a	d l a s k	z e b r a
s l u c h	m o u k a	m o u k a
d l a s k	s l o v o	s l u c h
z e b r a	s l u c h	d l a s k
s l o v o	z e b r a	s l o v o

3) SORT a$()(2 TO 4)

```
z e b r a ¦
d l a s k ¦
s l o v o ¦
s l u c h ¦
m o u k a ¦
  -----
```

4) SORT a$(2 TO 4)(3 TO 4)

```
m o u k a
d l a s k ¦
z e b r a ¦
s l u c h ¦
s l o v o
    ---
```

Příkazy označené 1-4 byly vždy aplikovány na původní pole a$(5,5). V prvním případě je abecedně seřazeno celé pole, ve druhém je opět seřazeno celé pole, ale podle hodnoty znaků v pátém sloupci. Celé pole je tříděno i ve třetím případě, ale tentokrát podle sloupců 2 až 4. Nakonec jsou setříděny pouze řádky 2 až 4 podle hodnoty znaků ve sloupcích 3 až 4. Tento příkaz je velmi silný a to nejen proto, že třídění je prováděno rutinou strojového kódu (a je tudíž velmi rychlé).

Nepřehlédněte, že při třídění numerických polí není povolen jako druhý parametr rozsah sloupců, ale jen číslo jediného sloupce.

SPLIT <SS>+<W>
=====

Pracuje s řádky v dialogovém řádku (editačním). Přesunete kurzor za oddělovač příkazů ':', stisknete '<>' a to co je nalevo od '<>' se po <ENTER> uloží do programu a část, která byla napravo v editačním řádku zůstane a to s tímtéž číslem řádku. Příklad: editujeme řádek:

```
10 PRINT "Nazdar": GO TO 20: <> PRINT "Příklad"
```

po stisku <ENTER> je do programu odeslán řádek:

```
10 PRINT "Nazdar": GO TO 20
```

a v editačním řádku zůstane:

```
10 (kurzor) PRINT "Příklad"
```

číslo řádku změníte, jak potřebujete.

TRACE číslo řádku <GRAPHICS>,<T>
TRACE příkaz: příkaz: příkaz: ... : RETURN
==========

Příkaz se používá při ladění programu. Umožňuje například výpis řádků během běhu programu. Příklad:

```
TRACE: LIST lino TO lino: PAUSE 0: RETURN
```

1) Před každým řádkem se odskočí na podprogram začínající na řádku deklarovaném v příkazu TRACE.
2) Příkazy stojící za TRACE se před každým řádkem provedou jako podprogram.

V době provádění trasovacího programu je režim trasování vypnut. TRACE 0, RUN a CLEAR vypínají tento režim úplně.

Podprogramem při trasovacím režimu může být rutina k nejrůznějším účelům. Příklad: výpis čísla řádku a příkazu a proměnné a$ na určitém místě.

```
9000 LET sloupec=PEEK 23688, radek=PEEK 23689
9010 PRINT AT 0,0;INVERSE 1;lino;":";stat,"A$= ";a$
9020 POKE 23688,sloupec
     POKE 23689,radek
     RETURN
```

Při zobrazování proměnných při trasovacím režimu je třeba dbát na to, aby proměnné byly systému známy.

UNTIL podmínka <GRAPHICS>,<K>
==========

Viz DO ... LOOP.

USING <GRAPHICS>,<U>
=====

Používá se při formátování tisku. Používá se ve tvaru:

PRINT USING formátovací řetězec; číselný výraz

Na tomto místě probereme také funkci USING$. Příkaz USING a funkce USING$ umožňují výstup čísel ve formátovaném tvaru. USING lze použít pouze ve spojení s příkazem PRINT. USING$ má širší použití. Výsledkem USING je výstup na obrazovku. USING$ vrací řetězec. Ve formátovacím řetězci se používají následující znaky:

```
 # - mezera nebo číslice před desetinnou tečkou
 0 - nula nebo číslice před desetinnou tečkou
písmeno - zobrazí se
     /Za desetinnou tečkou jsou v obou případech číslice
     zaokrouhlené podle matematických pravidel./
 . - desetinná tečka
```

Například:

```
100 FOR i=1 TO 50
      LET x=RND*100
      PRINT x,USING "###.##";x
    NEXT i
```

Podívejte se např. jak bude různými formátovacími řetězci vytištěno číslo 12.3456:

```
"##.#"      12.3
"###.#"      12.3
"####.##"     12.35
"000.00"    012.35
"00"        12
"DM00.00"   DM12.35
"0.00"      % ..3
```

Znak '%' v posledním příkladu signalizuje tzv. přetečení formátu. Pomocí USING nelze zpracovávat čísla v exponenciálním tvaru. Zaregistrujte, že příkazy PRINT USING a$;číslo a PRINT USING$(a$,číslo) mají stejný efekt.

VERIFY [[řádek1] TO [řádek2];][mechanizmus;]"jméno"
<EXTEND>,<CS>+<R>

==========

Popsáno u SAVE.

WINDOW číslo[,x,y,šířka,výška]
<GRAPHICS>,<5>

==========

Viz také CLS, CSIZE. Příkaz vytváří pro zobrazení výstupu pravoúhlá okna. Příkazy PRINT a LIST je možno směrovat jen do nich, bez vlivu na zbytek obrazovky. Každé okno má svou vlastní pozici PRINT, OVER, BRIGHT, FLASH, CSIZE, INK a PAPER. Okna mohou mít čísla v intervalu 1..127. Okna vytvořená tímto způsobem mají uložena informace o svých parametrech nad RAMTOPem a informace jsou proto chráněny před NEW.

Zvláštním případem je okno 0, které je aktivní po natažení BB03. Prvotně jsou jeho rozměry stanoveny na celou obrazovku.

Příklad: WINDOW 1,0,175,128,176

Okno 1 bude mít levý horní roh v pozici 0,175, je široké 128 pixlů a vysoké 176 pixlů (levá polovina obrazovky). Atributy jsou stejné jako v okně 0.

Okno se aktivuje příkazem WINDOW 1. Při vypnutí aktivního okna, je aktuální CSIZE a pozice PRINT uložena nad RAMTOP.

Příkaz WINDOW ERASE vymaže všechny definice oken.

Příklad:

```
10 WINDOW 1,0,175,128,176      ! definice 1. okna
   WINDOW 1
   PAPER 6
   CSIZE 16,8
   CLS
   WINDOW 0
20 WINDOW 2,128,175,128,88     ! definice 2. okna
   WINDOW 2
   CSIZE 4,8
   BRIGHT 1
   CLS
   WINDOW 0
30 WINDOW 3,128,87,128,88      ! definice 3. okna
   WINDOW 3
   CSIZE 8
   INK 6
   PAPER 2
   CLS
   WINDOW 0
40 FOR i=1 TO 1000
     LET a$=CHR$(RND(92)+32)
     FOR j=1 TO 3
       WINDOW j
       PRINT a$;
       INPUT;          ..........  ! tento netypický INPUT
     NEXT j                        ! zabraňuje dotazu SCROLL.
   NEXT i                          ! Funguje i v Sinclair BASIC.
50 WINDOW ERASE                    ! Místo POKE 23692,255.
   CSIZE 0
```

XOS, YOS, XRG, YRG
==========

Viz také přílohu D. Nejde o klíčová slova, ale zvláštní proměnné. Umožňují měnit souřadnicový systém obrazovky pro příkazy PLOT, DRAW, DRAW TO a CIRCLE. Tyto proměnné příkazy RUN a CLEAR nevymažou, ale jsou inicializovány na standardní počáteční hodnoty. Po

```
LET xrg=2,yrg=2,xos=1,yos=1
```

mohou být parametry grafických příkazů x,y v intervalu -1..1. PLOT 0,0 udělá tečku uprostřed obrazovky.
Příklad:

```
 10 GOSUB 100
 20 LET xrg=128
    GOSUB 100
 30 LET yrg=88
    GOSUB 100
 40 LET xrg=256
    GOSUB 100
    STOP
100 CLS
    PLOT 0,0
    DRAW 50,0
    DRAW 0,50
    DRAW -50,0
    DRAW 0,-50
    PAUSE 100
    RETURN
```

Podprogram nejprve nakreslí čtverec, potom ležící obdélník, dále stojící obdélník a nakonec opět čtverec. Příklad:

```
100 LET xrg=2*PI
    REM 360 stupňů
110 LET yrg=2.2
    REM sinus je od -1 do 1
120 LET yos=1.1
    REM nula na ose y je uprostřed
130 FOR i=0 TO 2*PI STEP 2*PI/256
      PLOT i,SIN i
    NEXT i
```

POZOR! Při použití DRAW, DRAW TO a CIRCLE je správně přepočítán začátek a konec, resp střed, ale poloměr a oblouk je nezměněn. Tzn. máte-li xrg=yrg=1000, pak nelze, stejně jako při standardních parametrech, nakreslit kružnici s větším průměrem než 87.

Nedostaneme tedy při deformované souřadnicové soustavě a při zadání příkazu CIRCLE elipsu, jak bychom očekávali. Je to proto, že BB03 pro CIRCLE využívá původní rutinu v ROM a upravuje pouze souřadnice středu.

Pokud snad zatoužíte po deformaci kružnic, musíte si pro kružnici nadefinovat podobnou proceduru, jako v následujícím příkladu:

```
10 DEF PROC kruznice x,y,r
20   LOCAL i,krok
30   PLOT x+r,y
40   FOR i=2/r TO 2*PI+2/r STEP 2/r
       DRAW TO x+r*COSE(i),y+r*SINE(i)
     NEXT i
50 END PROC
60 FOR j=300 TO 150 STEP -30
     LET xrg=j,xos=j/2
     FOR i=10 TO 70 STEP 40
       kruznice 0,88,i
     NEXT i
   NEXT j
```

Další příklad:

```
10 FOR j=300 TO 150 STEP -30
     LET xrg=j,xos=j/2
     FOR i=10 TO 70 STEP 40
       PLOT -50,88
       DRAW 100,0,PI
       DRAW -100,0,PI
     NEXT i
   NEXT j
```

F U N K C E

V BB03 máte k dispozici 26 nových funkcí. Všechny jsou definovány v řádku 0. Jejich skutečné definice jsou z důvodů rychlosti ve strojovém kódu BB03. Při listingu jsou zobrazovány jako klíčové slovo. Po napsání FN s$ se v editačním řádku objeví STRING$ a kurzor jej přeskakuje, jako by šlo o klíčové slovo.

Pokud převádíte pod BB03 nějaký program, který používá definované funkce, které kolidují s funkcemi BB03, musíte je přejmenovat na některou funkci, kterou BB03 nepoužívá.

Jména funkcí je možné vložit vypsáním nebo FN písmeno a $ nebo (.

Při načítání programů, které nebyly napsány v BB03 je nutné použít MERGE. Pro programy napsané v BB03 je možné použít LOAD, protože SAVE uložil i řádek 0.

AND (číslo,číslo) — FN A(číslo,číslo)

Funkce realizuje bitový AND (logický součin) mezi dvěma čísly. Čísla mohou být v intervalu 0 - 65535.

BIN$ (číslo) — FN B$(číslo)

Funkce převede číslo 0 - 255 do osmiznakového řetězce, který je dvojkovou reprezentací čísla a číslo 256 - 65535 do šestnáctiznakového.

Pokud máte touhu místo 1 a 0 zobrazovat jiné znaky, můžete vložit na adresu 62865 znak pro 1 a na adresu 62869 znak pro 0. Příklad:

```
10 POKE 62865,"I"
   POKE 62869,"O"
20 INPUT "1.číslo: ";c1,"2.číslo: ";c2
30 PRINT TAB 10;BIN$(c1);TAB 22;USING"###";c1'"AND";TAB 10;
   BIN$(c2);TAB(22);USING"###";c2'STRING$(32,"-")'"=";
   TAB 10;BIN$(AND(c1,c2));TAB 22;USING"###";AND(c1,c2)
40 PRINT
   GO TO 20
```

Program funguje správně, splňují-li vstupy následující podmínky:

0<=c1<=255 AND 0<=c2<=255 AND c1+c2<=255

CHAR$ (číslo) FN C$(číslo)

Viz také NUMBER(řetězec). Funkce převádí celá čísla v rozsahu 0..65535 na dvouznakové řetězce. Výsledek je nasnadě. Ušetříte paměť. Obrat LET c$=CHAR$(cislo) lze ve standardním BASICu Spectra napsat následovně:

```
LET A=INT(cislo/256)
LET B=cislo-A*256
LET c$=CHR$ A + CHR$ B
```

Pokud byste chtěli takto získané řetězce tisknout, dostanete velmi často zprávu "K Invalid colour", protože často obsahují řídící tiskové kódy. Zpět na číslo tento řetězec převedete funkcí NUMBER. Uložená čísla zabírají místo obvyklých pěti bajtů pouze dva bajty.

V mnoha případech není na překážku, že čísla nejsou celá. Můžete například číslo 87.643 transformovat na 8764.3. CHAR$ převede číslice před desetinnou tečkou na řetězec o délce 2 bajty, pak při zpětném převodu pomocí NUMBER dostanete číslo 87.64, což je pro velké množství aplikací přesnost zcela dostačující. Příklad:

```
10 DIM a(1000)
   DIM a$(1000,2)
   DIM b$(1000,2)
   PRINT "Pracuji."
   FOR i=1 TO 1000
     LET a(i)=RNDM(1000),a$(i)=CHAR$(a(i)),b$(i)=a$(i)
   NEXT i
20 PRINT "Třídění $ pole:";
   CLOCK "0"
   SORT b$ ()()
   PRINT TIMES$()
   PRINT "Třídění # pole"
   CLOCK "0"
```

pokračování na další straně

```
      SORT INVERSE a()
      PRINT TIME$()
      BEEP 1,20
   30 PAUSE 100
      CLS
      LET f$="########"
      FOR i=1 TO 1000
        PRINT USING f$;a(i);USING f$:NUMBER(a$(i));USING f$;
        NUMBER(b$(i))
      NEXT i
```

COSE (číslo) FN C (číslo)

Funkce počítá kosinus. Proč, když Sinclair BASIC už COS má? Je cca 6% rychlejší, alespoň to manuál tvrdí, ale současně je méně přesná. Pro grafické aplikace a hry je to ovšem přesnost dostačující. (Jiný autor uvádí zrychlení 16x. Vyzkoušejte.)

DEC (řetězec) FN D (řetězec)

Funkce vrací desítkový ekvivalent hexadecimálního řetězce, který je jejím argumentem. Délka řetězce 1 - 4 znaky, velikost znaků nerozhoduje. Příklady:

```
DEC ("FF") = 255
DEC ("10") = 16
DEC ("4000") = 16384
DEC ("e") = 14
```

Je-li argument neplatný, dostanete 'A Invalid argument'.

DPEEK (adresa) FN P (adresa)

Jedná se o 'dvojitou' funkci PEEK pro danou a následující adresu. Manuál tvrdí, že pomalá BASICová verze vypadá následovně:

```
PEEK adresa + 256 * PEEK (adresa+1)
```

Funkční předpis souhlasí, ale BASICová verze je téměř 2x rychlejší než DPEEK.

EOF (číslo kanálu) FN E (číslo kanálu)

Funkce vrací při čtení dat z drajvu 0 a v případě posledního údaje dá 1. EOF = End Of File = konec souboru.

FILLED () FN F ()

Funkce vrací počet pixlů použitých k vyplnění uzavřeného obrazce příkazem FILL. (Týká se vždy posledního FILL). Příklad:

```
10 FOR i=20 TO 80 STEP 20
     CIRCLE 128,88,i
     FILL 128,88
     PRINT AT 0,0;"FILLED ()=";FILLED ()
     PAUSE 150
     CLS
   NEXT i
```

HEX$ (číslo) FN H$ (číslo)

Viz také DEC. Numerický výraz, jehož hodnota musí být v intervalu -65536,,65535, je převeden na šestnáctkový řetězec. Hodnoty mimo tento rozsah mají za následek chybu 'B Integer out of range'. Příklady:

```
HEX$ (32) = "20"
HEX$ (255) = "FF"
HEX$ (-1024)= "FB00"
```

V případě záporného argumentu je výsledek dvojkovým doplňkem.

Výsledný řetězec je závislosti na argumentu dvou- nebo čtyřznakový.

INARRAY ($pole (start [,[#výraz] TO [#výraz]]),řetězec) FN U (.)

Viz také INSTRING. Funkce prohledává dané řetězcové pole a hledá v něm řetězec, který je posledním parametrem funkce. Vrátí číslo řádku, když řetězec nalezne, nebo 0, není-li řetězec nalezen. V podstatě jde o INSTRING aplikovaný na celé pole. V případě, že jsou některé znaky řetězce nepodstatné, lze na jejich místo vložit '#'.

Funkce je použitelná na maximálně dvourozměrná pole. Příklad:

```
10 DIM a$(22,32)
   FOR i=1 TO 22
     FOR j=1 TO 32
       LET a$(i,j)=CHR$(RNDM(25)+65)
     NEXT j
   NEXT i
20 DEF PROC pis
     CLS
     FOR i=1 TO 22
       PRINT a$(i)
     NEXT i
   END PROC
30 pis
40 LET a$(8,10 TO 15)="KLOKAN"
50 PAUSE 0
   pis
60 PAUSE 0
   CLS
   LET radek pole=INARRAY(a$(1),"KLOKAN")
   LET sloupec pole=INSTRING(1,a$(radek pole),"KLOKAN")
   PRINT radek pole, sloupec pole
70 pis
   PRINT OVER 1; BRIGHT 1; AT radek pole-1, sloupec pole-1;
   "      ";
   PRINT #1; AT 0,0;"řádek=8, sloupec=10"
   PAUSE 0
```

INSTRING (start, řetězec1, řetězec2) FN I (...)

Funkce prohledává řetězec1 od pozice start na výskyt řetězce2. Je-li řetězec nalezen, vrátí funkce pozici prvního znaku, na kterém se řetězce shodují. V opačném případě vrátí nulu. Znaky v řetězci2, které nejsou podstatné, je možné nahradit znakem '#'.

Příklad: INSTRING (1,a$,"M##ER") může v řetězci a$ najít pozici podřetězců: MAYER, MILER, MEIER ap. Znak '#' můžete hledat jen v případě je-li prvním znakem řetězce2.

Retězec1 může mít libovolnou délku, řetězec2 může být max. 255 znaků dlouhý, jinak zpráva 'Invalid argument'. Nulový výsledek dostanete také v případě, že řetězec2 je delší než řetězec1 nebo hodnota start je větší než délka řetězce1 a také v případě, že je jeden z řetězců prázdný.

Možnost určení začátku prohledávání je výhodná při hledání několikanásobného výskytu. Příklad:

```
100 DIM a$(1000)
110 FOR n=1 TO RNDM(10)+3
120   LET pos=RNDM(995)
130   LET a$(pos TO pos+3)="TEST"
140 NEXT n
150 PRINT "TESTy jsou ukryty""Stiskněte klávesu"
160 PAUSE 0
170 LET loc=1
180 LET loc=INSTRING(loc,a$,"TEST")
190 IF loc<>0 THEN
      PRINT "Nalezeno na pozici ";loc
      LET loc=loc+1
      GO TO 180
200 PRINT ".. a víc jich není."
```

INSTRING lze s úspěchem používat ve výukových programech při testování správnosti odpovědí. Pokud je např. očekávána odpověď "NAPOLEON" a testovaný mučeň odpověděl "NAPOLEON BONAPARTE", pak je to obvykle nesprávná odpověď. Pomocí INSTRING můžete nalézt krátkou odpověď v dlouhé a váš počítač se tváří poněkud inteligentněji.

Další využití této výkonné funkce: řetězce různé délky lze umístit do společného řetězce a tak uspořit spoustu místa. Zvolíte si některé kódy ASCII jako vaše vlastní řídicí znaky, které budou označovat začátek a konec vloženého podřetězce, délku, prostě tak, jak budete potřebovat.

ITEM () FN T ()

Viz kapitolu o procedurách. Funkce informuje o typu další položky, která má být čtena příkazem READ. Je určena především pro použití v procedurách, ale lze ji použít i pro normální příkazy DATA a READ. Vrací hodnotu:

0 - všechny položky byly přečteny
1 - následující položka bude řetězec
2 - následující položka bude numerická

LENGTH (n,"jméno pole") FN L (n,a$)

Funkce vrací informace o velikosti pole, což je v BB03 důležitá informace, protože velikost polí se dá pružně měnit. Funkce může vrátit i adresu, na níž daný řetězec leží.

Pro n = 1 dostanete velikost 1. rozměru
n = 2 dostanete velikost 2. rozměru
n = 0 vrátí adresu prvního prvku pole nebo řetězce

Příklad:

```
10 DIM a$(9,32)
   LET a$(1)="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdef"
20 PRINT "První rozměr a$()=";LENGTH(1,"a$()")
30 PRINT "Druhý rozměr a$()=";LENGTH(2,"a$()")
40 PRINT "Adresa prvního prvku=";LENGTH(0,"a$()")
50 PRINT a$(1)
60 POKE LENGTH(0,"a$()"),"````"   (=znaky libry)
   PRINT a$(1)
```

MEM () FN M()

Tato bezargumentová funkce vrací velikost volné paměti v bajtech. Závorky musí být a to prázdné. Příklad:

```
10 PRINT MEM()
   DIM a$(20000)
   PRINT MEM()
```

Bez BB03, v Sinclair BASICu, byste mohli použít obrat:

```
PRINT 65535 - USR 7962
```

MEMORY$ ()([#výraz] TO [#výraz]) FN M$()()

Viz také POKE řetězcem. Funkce má k dispozici celou paměť Spectra jako řetězcovou proměnnou. Z technických důvodů není přístupná adresa 0 a poslední tři paměťové buňky, takže platí LEN (MEMORY$())=65532.

Obrazovku je možné odložit příkazem: LET a$=MEMORY$()(16384 TO 22527).

Můžete hledat v celé paměti libovolný řetězec:

```
10 REM asdfg
20 PRINT INSTRING(1,MEMORY$(),"asdfg")
```

MOD (číslo,číslo) FN U (číslo,číslo)

Výsledek funkce je zbytek po celočíselném dělení čísla1 číslem2 (dělení modulo). Např.:

```
MOD (5,2)=1
MOD (3,4)=3
MOD (-7,3)=2
```

NUMBER (řetězec) FN N (řetězec)

Viz také CHAR$(číslo). Dvouznakový řetězec je převeden na číslo od 0 do 65535. Pokud nemá argument dva znaky, je hlášena chyba 'Invalid argument'. Činnost této funkce ilustruje BASICový ekvivalent:

```
LET cislo=256 * CODE c$(1) + CODE c$(2)
```

OR (číslo1,číslo2) FN O (číslo1,číslo2)

Uskutečňuje bitový OR mezi oběma argumenty.

RNDM (číslo) FN R (číslo)

Je-li argument 0 je výsledkem číslo v intervalu 0..1. Pokud je číslo větší než 0, je výsledkem celé číslo v intervalu 0..číslo.

SCRN$ (řádek,sloupec) FN K$ (řádek,sloupec)

Pracuje podobně jako SCREEN$, ale umí rozeznat i UDG znaky.

SHIFT$ (číslo,řetězec) FN Z$ (číslo,řetězec)

Funkce používaná ke konverzím řetězců. V závislosti na prvním numerickém parametru jsou prováděny příslušné konverze řetězce:

1 - Všechny znaky abecedy na velké
2 - Všechny znaky abecedy na malé
3 - Inverze malých a velkých písmen
4 - Všechny řídicí znaky kromě CHR$ 13 jsou změněny na '.'
5 - Jako 4, navíc změní CHR$ 128-255 na CHR$ 0-127
6 - Jako 5, ale transformuje i CHR$ 13 na '.'
7 - Změní tokens na vypsaný tvar
8 - Opak 7, ignoruje velikost znaků
9 - ?
10 - ?
11 - Jako 8, ale řetězce, které mají tvar klíčového slova, musí být psány velkými písmeny.

SINE (číslo) FN S (číslo)

Rychlý sinus. Viz COSE.

STRING$ (číslo,řetězec) FN S$ (číslo,řetězec)

Funkce funguje následovně:

```
STRING$(16,"-") vrátí "----------------"
STRING$(4,"AB")   "   "ABABABAB"
STRING$(3,"A"+CHR$ 10) vrátí  A
                              A
(samozřejmě při tisku)        A
```

TIME$() FN T$ ()

Vrací systémový čas ve tvaru "HH:MM:SS".

USING$ (formát,číslo) FN U$ (,,,)

Číslo je převedeno na řetězec, který má tvar v souhlase s formátovacím řetězcem. Podrobnosti viz USING.

XOR (číslo1,číslo2) FN X (číslo1,číslo2)

Funkce relizuje bitový exkluzivní součin (EXCLUSIVE OR).

---*** následují přílohy ***---

Natištěno pro ZO Svazarm Karolinka

Březen 1990

(celkem 62 stran i s přílohami)

P Ř Í L O H A A

V režimu KEYWORDS 1 jsou klávesám v grafickém režimu přiřazeny následující významy:

Kód	Klávesa	Klíčové slovo
128	8	KEYWORDS
129	1	DEF PROC
130	2	PROC
131	3	END PROC
132	4	RENUM
133	5	WINDOW
134	6	AUTO
135	7	DELETE
136	CS+7	REF
137	CS+6	JOIN
138	CS+5	EDIT
139	CS+4	KEYIN
140	CS+3	LOCAL
141	CS+2	DEFAULT
142	CS+1	DEF KEY
143	CS+8	CSIZE
144	A	ALTER
145	B	BLANK
146	C	CLOCK
147	D	DO
148	E	ELSE
149	F	FILL
150	G	GET
151	H	BLANK
152	I	EXIT IF
153	J	WHILE
154	K	UNTIL
155	L	LOOP
156	M	SORT
157	N	ON ERROR
158	O	ON
159	P	DPOKE
160	Q	POP
161	R	ROLL
162	S	SCROLL
163	T	TRACE
164	U	USING

Kódy označené BLANK nejsou obsazeny klíčovým slovem. Znaky pro velikost CSIZE 4,8 jsou obsaženy v RAM na adresách 51291 až 51626 (CHR$ 32 až CHR$ 127). Vždy dva znaky zabírají 7 bajtů. Sudé znaky obsazují 4 levé bity každého bajtu, liché 4 pravé bity. Pro každý znak se vytvoří 1 řada prázdných pixlů.

P Ř Í L O H A B

Nová chybová hlášení a rozšířený význam starých chybových hlášení:

Kód	Význam	Situace
G	No room for line Nové přečíslování řádků by vedlo k neočíslovaným řádkům, nebo k řádkům s čísly většími než 9999.	RENUM
S	Missing LOOP Po EXIT IF, nebo podmíněném DO (s WHILE nebo UNTIL) nebyl nalezen příslušný LOOP.	DO, EXIT IF
T	LOOP without DO LOOP bez příslušného DO.	LOOP
U	No such line Nebyl nalezen řádek uvedený v DELETE.	DELETE
V	No POP data Pokus o přečtení návratové adresy v prázdném zásobníku. Nebyly aktivovány cykly, podprogramy ani procedury.	POP
W	Missing DEF PROC Byla volána procedura, aniž byla definována, popř. byl nalezen END PROC bez odpovídajícího DEF PROC.	PROC, END PROC
X	No END PROC Program se pokusil přeskočit definici procedury, ale nenalezl její konec.	DEF PROC

P Ř Í L O H A C

Následující seznam obsahuje hodnoty, které jsou při vzniku chyby přiřazovány proměnné error vytvářené příkazem ON ERROR. Nejsou zachyceny chyby 0 a 9.

Hodnota error	Kód	Hlášení
0	0	OK - není zpracováno
1	1	NEXT without FOR NEXT bez FOR
2	2	Variable not found Proměnná nebyla nalezena
3	3	Subscript wrong Spatný index
4	4	Out of memory Přeplněná paměť
5	5	Out of screen Mimo obrazovku
6	6	Number too big Příliš velké číslo
7	7	RETURN without GOSUB RETURN bez GOSUB
8	8	End of file Konec souboru
9	9	STOP statement - není zpracováno Příkaz STOP
10	A	Invalid argument Neplatný argument
11	B	Integer out of range Celé číslo mimo povolený rozsah
12	C	Nonsence in BASIC Nesmysl v BASICu
13	D	BREAK-CONT repeats Přerušení, CONTINUE bude opakovat
14	E	Out of DATA Příkaz READ nenašel data
15	F	Invalid file name Spatné jméno souboru
16	G	No room for line Pro řádek není místo
17	H	STOP in INPUT V INPUT byl vložen STOP
18	I	FOR without NEXT FOR nemá NEXT
19	J	Invalid I/O device Neplatné I/O zařízení
20	K	Invalid colour Neplatná barva
21	L	BREAK into program Přerušení programu, CONTINUE nebude opakovat, ale pokračovat.
22	M	RAMTOP no good Spatné nastavení vršku paměti

23	N	Statement lost Skok na ztracený příkaz
24	O	Invalid stream Nesprávný příkaz pro Interface 1
25	P	FN without DEF Volání nedefinované funkce
26	Q	Parameter error Chybný parametr
27	R	Tape loading error Chyba čtení pásku

CHYBY B.B.O.S

28	S	Missing LOOP Chybí LOOP
29	T	LOOP without DO LOOP bez DO
30	U	No such line Tento řádek neexistuje
31	V	No POP data Zásobník je prázdný
32	W	Missing DEF PROC Nedefinovaná procedura
33	X	No END PROC Procedura nemá konec

CHYBY INTERFACE 1

43	b	Program finished Program skončil
44	c	Nonsence in BASIC Nesmysl v příkazu pro Interface 1
45	d	Invalid stream number Neplatné číslo kanálu
46	e	Invalid device expression Neplatný výraz pro zařízení
47	f	Invalid name Neplatné jméno
48	g	Invalid drive number Neplatné číslo drajvu
49	h	Invalid station number Neplatné číslo stanice v síti
50	i	Missing name Chybí jméno
51	j	Missing station number Chybí číslo stanice
52	k	Missing drive number Chybí číslo drajvu
53	l	Missing baud rate Chybí rychlost přenosu
54	m	Header mismatch error Chyba v hlavičce

55	n	Stream already open Proud je již otevřen
56	o	Writing to a "read" file Pokus o zápis do souboru otevřeného pro čtení
57	p	Reading a "write" file Pokus o čtení ze souboru otevřeného pro zápis
58	q	Drive "write" protect Zápis na daný mechanismus není povolen
59	r	Microdrive full Kazeta drajvu je plná
60	s	Microdrive not present Drajv není připojen
61	t	File not found Soubor nebyl nalezen
62	u	Hook code error Chyba při volání ROM v Interface 1
63	v	CODE error Chyba kódu
64	w	MERGE error Chyba při MERGE
65	x	Verification failed Verifikace se nezdařila
66	y	Wrong file type Spatný typ souboru

P Ř Í L O H A D

Zvláštní proměnné a nové systémové proměnné:

a) grafické: nelze je vymazat pomocí CLEAR a RUN, těmito příkazy jsou pouze nastaveny na standardní hodnoty. Jejich hodnotu lze nastavovat pomocí LET a mají vliv na souřadnicový systém příkazů DRAW, DRAW TO, PLOT, CIRCLE, GET a FILL.

Jméno	st.hodnota	význam
xos	0	PLOT x+xos,y
xrg	256	0<=x<=xrg
yos	0	PLOT x,y+yos
yrg	176	0<=y<=yrg

b) TRACE a ON ERROR proměnné jsou vytvořeny za zvláštních podmínek. Můžete použít vlastní proměnné se stejnými jmény, ale pokud dojde k nějaké události, budou jejich obsahy přepsány hodnotami, které dodá systém.

Jméno	význam
error	Kód poslední chyby (viz příloha C)
lino	TRACE - číslo následujícího řádku ON ERROR - číslo řádku s chybou
stat	TRACE - pořadové číslo následujícího příkazu ON ERROR - pořadové číslo příkazu s chybou

c) Další systémové proměnné:

23692 (2)		Číslo prvního řádku procedury po LIST PROC
57358 (2)		Číslo posledního řádku procedury po LIST PROC
57362	OVER2T	Stav dočasného OVER 2
57363	OVER2P	Permanentní OVER 2 stav (1=ano, 0=ne)
57364		X-souřadnice výstupu na obrazovku (ne v případě CSIZE 0)
57365		Y-souřadnice (jako předchozí)
57366	XRIGHT	Max X-souřadnice (limit okna)
57367	XLEFT	Min X
57368	YTOP	Max Y
57369	YBOT	Min Y
57376	CURWIN	Aktuální okno
57381	PRETTY	1 pokud je požadován strukturovaný listing, jinak 0
57382	LNS	0 v případě, že se nemají vypisovat čísla řádků
57383	KEYWD	Vstupní režim 2, 3, 4
57384		Počet řádků v horní části obrazovky, závisí na CSIZE a WINDOW

57391		Počet znaků na řádku
57395	DEFAULT	Číslo drajvu, číslo stanice v síti
57395	DEFAULT	Písmeno zařízení (T,M,N,B)
61316		Dolní bajt, pozice pointru pro CLOCK, jako adresa obrazovky
61318		Horní bajt.
56866		Počet padesátin sekundy, které posunou hodiny o 1 sekundu
56870		54 pro 60 s za minutu 56 pro 100 s za minutu
23729		Pro INTERFACE 1 s výrobním číslem < 87316, počet znaků v řádku pro kanál t
57500		Pro INTERFACE 1 s výrobním číslem > 87316 počet znaků v řádku pro kanál t

P Ř Í L O H A E

Dostupnost příkazů v jednotlivých verzích:

BETABASIC 3.0	BB 1.0	BB 1.8	SB 1.10T
ALTER	o	o	o
AUTO	+	+	+
CLEAR	-	-	-
CLOCK	+	+	+
CLS (pro okna)	-	-	-
COPY (pro pole)	-	-	-
CSIZE	-	-	-
DEFAULT	-	-	-
DEF KEY	-	+	+
DEF PROC	o	o	o
DELETE	o	o	o
DO	+	+	+
DPOKE	+	+	+
DRAW TO	-	-	-
EDIT	o	o	o
ELSE	+	+	+
END PROC	+	+	+
EXIT IF	+	+	+
FILL	-	+	+
GET	o	o	o
JOIN	-	o	o
KEY IN	-	+	+
KEYWORDS	-	o	o
LIST DATA	-	-	-
LIST VAL	-	-	-
LIST VAL$	-	-	-
LIST DEF KEY	-	-	-
LIST FORMAT	-	-	-
LIST PROC	-	-	-
LIST REF	-	-	-
LOCAL	-	-	-
LOOP	+	+	+
ON	o	o	o
ON ERROR	o	o	o
OVER 2	-	-	-
POP	+	+	+
PROC	+	+	+
READ LINE	-	-	-
REF	-	-	-
RENUM	+	+	+
ROLL	+	+	+
SCROLL	+	+	+
SORT	+	+	+
TRACE	+	+	+
UNTIL	+	+	+
USING	+	+	+
WINDOW	-	-	-
WHILE	+	+	+

\+ = příkaz je dostupný
o = příkaz má proti BB03 omezené možnosti
\- = příkaz není dostupný

OBSAZENÍ DEFINOVANÝCH FUNKCÍ

FN	BB 3.0	BB 1.0	BB 1.8	SB 1.10T
A	AND	-	+	+
B	-	-	-	-
C	COSE	-	+	+
D	DEC	+	+	+
E	EOF	-	-	-
F	FILLED	-	+	+
G	-	-	-	-
H	-	-	-	-
I	INSTRING	+	+	+
J	-	-	-	-
K	-	-	-	-
L	LENGTH	-	-	-
M	MEM	+	+	+
N	NUMBER	+	+	+
O	OR	-	+	+
P	DPEEK	+	+	+
Q	-	-	-	-
R	RNDM	-	+	+
S	SINE	-	+	+
T	ITEM	-	-	-
U	INARRAY	-	-	-
V	MOD	-	+	+
W	-	-	-	-
X	XOR	-	+	+
Y	-	-	-	-
Z	-	-	-	-
A$	-	-	-	-
B$	BIN$	-	+	+
C$	CHAR$	+	+	+
D$	-	-	-	-
E$	-	-	-	-
F$	-	-	-	-
G$	-	-	-	-
H$	HEX$	+	+	+
I$	-	-	-	-
J$	-	-	-	-
K$	SCRN$	-	+	+
L$	-	-	-	-
M$	MEMORY$	-	+	+
N$	-	-	-	-
O$	-	-	-	-
P$	-	-	-	-
Q$	-	-	-	-
R$	-	-	-	-
S$	STRING$	+	+	+
T$	TIMES$	+	+	+
U$	USING$	+	+	+
V$	-	-	-	-
W$	-	-	-	-
X$	-	-	-	-
Y$	-	-	-	-
Z$	SHIFT$	-	-	-

P Ř Í L O H A F

ROZDÍLY BETABASIC 1.0

Na pásce je nahrán následovně:

```
1. BetaBas1.0  P      1       535
2. headerless                 18
3. headerless               5455
```

ALTER

Dostupná je pouze varianta měnící atributy.

DEF PROC

Procedury nemohou mít parametry.

GET

Dostupná je pouze varianta čtoucí klávesnici.

KEYWORDS

Lze nastavit pouze: 0 - zapnuté UDG
1 - zapnutá nová klíčová slova

ON

Použitelná syntaxe pouze:
GO TO!GO SUB ON výraz; č.řádku, č.řádku, ...

ON ERROR

Povoleno pouze:
ON ERROR č.řádku

PROC

Při volání procedury je nutné používat klíčové slovo PROC.

TRACE

Povolený tvar: TRACE č.řádku.

BB 1.0 má navíc chybovou zprávu 'Y Too hard'. Chybová situace může nastat při RENUM, narazí-li algoritmus na počítaný skok (např. GO TO 100*a).

Příklady programů chodících v BB 1.0

```
 10 LIST: LET w=8+8+8
 20 PLOT 90,70
 30 INK 2: GOSUB 200
 40 FOR N=1 TO W
 50   ROLL 5; 30,150;8,80
 60   ROLL 7; 50,170;2,120
 70 NEXT N
 80 FOR N=1 TO W
 90   ROLL 6;50,170;2,120
100   ROLL 8;38,150;8,80
110 NEXT N
120 LET W=W/8: IF W>=1 THEN GO TO 40
130 PLOT 90,70
140 INK 0: GO SUB 200
150 STOP
200 DRAW OVER 1; 30,20,4977: RETURN
```

Následující program ukazuje rozumné použití chybové rutiny.

```
 10 ON ERROR 130
 20 BORDER 0: PAPER 0: INK 6: CLS
 30 DO
 40  LET X=30
 50  LET Y=50
 60  INPUT "Z - (STOP při Z=999) ";Z
 70  CLS
 80  PRINT AT 21,0;Z
 90  PLOT 110,80
100  DRAW X,Y,Z
110 LOOP UNTIL Z=999
120 CLS: LIST: STOP
130 IF error=11 AND line=80 THEN RETURN: ELSE POP: CONTINUE
```

Zkuste pro Z zadat např.: 451, 777, 1e8 atd.

P Ř Í L O H A G

R O Z D Í L Y B E T A B A S I C 1.8

Pro ALTER, DEF PROC, DELETE, GET, KEYWORDS ON, ON ERROR, PROC, TRACE platí tatáž omezení jako jsou uvedena u V 1.0.

POKE adresa, řetězec

Kromě např. uschování obrazovky a jejího vracení pomocí POKE, má tento příkaz i další zajímavé použití. Můžete si např. rozdělit paměť na několik oblastí a uschovat si v každé samostatný BASICový program. Každý si uložíte i se stavem systémových proměnných. Programy pak můžete mezi sebou vzájemně přepínat a celý proces bude poměrně rychlý, že budete mít dojem, že programy pracují paralelně. Napište CLEAR 33900 a pak pomocí RUN spustte následující program:

```
  100 POKE 34000, MEMORY$() (23552 TO 33800)
  110 REM Zde začíná dlouhý program
  120 LIST: FILL 0,0
 9999 STOP
```

Nyní je program uložen nad oblastí RAMTOP. Pokud nevěříte, dejte NEW a LIST - žádný program nebude v paměti. Vložte POKE 23552,MEMORY$()(34000 TO 44248). Můžete se přesvědčit, že se program vrátil zpět.

Poznámky k funkcím

AND

Pomocí AND (číslo, maska), můžete dané číslo překrýt, (maskovat) maskou a tím eliminovat nežádoucí bity. Masku můžete pro přehlednost zadat pomocí BIN.

```
PRINT AND (BIN 00000111, ATTR (řádek, sloupec))
```

Uvedený příkaz vrátí barvu INK pro definovaný znakový čtverec obrazovky. Místo BIN 00000111 lze napsat 7.

Následující příklad zobrazí "AND", vždy když stisknete klávesu <F>. Přitom není podstatné, je-li zároveň stisknuta ještě jiná klávesa.

```
100 IF AND (BIN 00001000, IN 65022)=0 THEN PRINT "AND";
110 GOTO 100
```

SCRN$

Funguje jako SCREEN$, ale rozeznává i znaku UDG 'A' - 'U'. Nerozeznává však blokovou grafiku z číselných kláves.

P Ř Í L O H A H

R O Z D Í L Y V 1.10 T a 1.12

Verze 1.10T vysílá po LPRINT a LLIST text na zdířku MIC, pravděpodobně ve formátu pro dálnopis.

- Obě verze umožňují vynechávat klíčové slovo LET.

IF ... THEN

Je možné použít tvar: IF e=10 PRINT e
nebo IF e=10 THEN b=e-1
nebo IF e=10 LET b=e-1

ON

Jsou použitelné formáty:

GOTO ON výraz;řádek, řádek, ...

a

ON výraz GOTO řádek, řádek

Zrychlené nahrávání

Do těchto verzí byla zahrnuta rutina Speedyload fy Ness Micro Systems. Rychlost práce s mgf je asi dvojnásobná. Použití: příkazy pro práci s magnetofonem mají stejnou syntaxi, jen před jménem se uvede '#'. Např.: SAVE #"jméno".

KEYWORDS #výraz1, #výraz2

Oba výrazy mohou nabývat pouze hodnot 1 a 0.
výraz1 = 1 - zobrazení nových klíčových slov
= 0 - zobrazení UDG resp. ASCII
výraz2 = 1 - režim vstupu celých klíčových slov
= 0 - režim vstupu klíčových slov po písmenech

PROC, DELETE, JOIN, EDIT, COPY, ALTER, GET, ON, ON ERROR, TRACE

Jejich schopnosti jsou proti BB 3.0 omezeny.

CALL /adresa parametry <GRAPHICS>,<5>

Volání rutiny strojového kódu podle adresy. Po zadání adresy už není kontrolována syntaxe, takže lze uvést libovolnou kombinaci parametrů. Na tomto řádku už nemohou být další příkazy BASICu. Strojový kód, volaný příkazem CALL musí být schopen převzít parametry a zajistit, aby po návratu do BASICu ukazoval čítač CHADD na první bajt za parametry.

CALL /adresa DATA výraz, výraz, výraz, výraz, výraz (max 5)

Volání procedury ve strojovém kódu s předem definovanými obsahy registrů procesoru. Hodnoty výrazů se postupně ukládají do registrových párů BC, DE, HL, AF a IX. Při použití této syntaxe může být na tomtéž řádku další příkaz. Např.:

CALL /949 DATA 0,1000,100

vyšle 100 tónových impulzů s časovací konstantou 1000. Programy volané pomocí CALL běží na stejné úrovni jako všechny příkazy jazyka, t.j. na úrovni 2. podprogramu. Instrukce RET vrací program přímo do smyčky interpreteru (test BREAK).

LOCK řetězec <GRAPHICS>,<CS>+<7>

Používá se pro "zamčení" systému proti nežádoucímu zásahu. Jedná se zejména o zásahy do programu, vypouštění a editace řádek, nahrání programu nebo jeho přepsání jiným, smazání programu apod. Znakový výraz v příkazu LOCK tvoří heslo, po jehož vložení systém přestane reagovat na pokusy o jeho ovlivnění. Vložené programy lze libovolně spouštět a počítač používat k běžným výpočtům. Nelze však používat systémové příkazy (POKE, PEEK, USR, CALL, SAVE, NEW, LOAD, ...), jsou pro uživatele nepřístupné. Jsou-li však obsaženy v programu, jsou prováděny normálně - nejsou dostupné jen v přímém režimu. Zrušit tento stav je možné dalším použitím příkazu LOCK se s t e j n ý m heslem, jako při prvním použití. Heslo je libovolný řetězec znaků o maximální délce 8. Po provedení příkazu LOCK "" (prázdné heslo) již systém nelze odemknout. Musíte použít RESET nebo síťový vypínač.

Nicméně pokud na takový program narazíte, pak jej stačí natáhnout bez BB přes MERGE a příslušné LOCK vypustit a je po problémech s utajováním.

ON #výraz GOTO řádek,řádek,... <GRAPHICS>,<N>
ON #výraz GOSUB řádek,řádek,...

Na rozdíl od ostatních popisovaných verzí je možná i tato syntaxe. Systém ovšem kvůli kompatibilitě připouští i tvary:

GOTO ON #výraz; řádek,řádek,...
GOSUB ON #výraz; řádek,řádek,...

Pokud je hodnota výrazu mimo zadaný rozsah pokračuje program na následujícím řádku.

STANDARDNÍ PŘÍKAZY SE ZMENENOU SYNTAXÍ

Je povoleno:

BEEP 1,10;1,20;.5,15 apod.

DIM a(10),b(100),a$(10,10) apod.

LET v=1,x=2,s=3 nebo i v=1,x=2,s=3 apod.

TABULKA INSTRUKCÍ A JEJICH ZKRATEK

Neexistuje-li zkratka, je to vyznačeno znakem '-'. Hvězdička '*' znamená, že instrukce není možno zadávat po písmenech, ale jako klíčové slovo v příslušném módu kurzoru. Všechny instrukce, v jejichž zápisu je mezera, je nutno zadávat bez této mezery, jinak nebudou rozpoznány.

ABS	-	BIN	-	CLS	-	DO	-
ACS	-	BORDER	B.	CONTINUE	CON.	DPOKE	DP.
ALTER	AL.	BRIGHT	BR.	COPY	C.	DRAW	D.
AND	A.	CALL	-	COS	-	EDIT	ED.
ASN	-	CAT	-	DATA	DA.	ELSE	EL.
AT	-	CHR$	CH.	DEF FN	DE.	END PROC	EN.
ATN	-	CIRCLE	CI.	DEF KEY	DEFK.	ERASE	E.
ATTR	*	CLEAR	CL.	DEF PROC	DEFP.	EXIT IF	EXI.
AUTO	AU.	CLOCK	-	DELETE	DEL.	EXP	-
BEEP	BE.	CLOSE	CLO.	DIM	-	FILL	FI.
FLASH	FL.	INKEY	*	LIST	LI.	NEXT	N.
FN	-	INPUT	IN.	LLIST	LL.	NOT	-
FOR	F.	INT	*	LN	-	ON	-
FORMAT	*	INVERSE	INV.	LOAD	LO.	ON ERROR	ON.
GET	-	JOIN	J.	LOCK	-	OPEN	OP.
GOSUB	G.	KEYIN	K.	LOOP	-	OR	-
GOTO	-	KEYWORDS	KEYW.	LPRINT	LP.	OUT	O.
IF	-	LEN	-	MERGE	M.	OVER	OV.
IN	-	LET	L.	MOVE	MO.	PAPER	PAP.
INK	-	LINE	-	NEW	-	PAUSE	PA.
PEEK	PE.	REM	-	SGN	-	TO	-
PI	-	RENUM	REN.	SIN	-	TRACE	TR.
PLOT	P.	RESTORE	RES.	SORT	SO.	UNTIL	UN.
POINT	POI.	RETURN	R.	SQR	-	USING	USI.
POKE	PO.	RND	-	STEP	-	USR	U.
POP	-	ROLL	RO.	STOP	ST.	VAL	-
PRINT	PR.	RUN	-	STR$	-	VAL$	*
PROC	-	SAVE	S.	TAB	-	VERIFY	V.
RANDOMIZE	RA.	SCREEN$	SC.	TAN	-	WHILE	W.
READ	-	SCROLL	SCRO.	THEN	TH.		

NEDOSTATKY PROGRAMU

Není-li v paměti přítomen program, může použití příkazu JOIN vést ke zhroucení systému.

Při syntaxi DATA příkazu CALL není kontrolován počet následujících výrazů. Nesmí jich být víc než 5 (BC, DE, HL, AF a IX registry). Je-li jich více než 6, dojde k přepsání části programu a ten se velmi pravděpodobně zhroutí.

SYSTÉMOVÉ PŘIPOMÍNKY

Systém lze kdykoliv vypnout příkazem CALL /59904 (nebo RANDOMIZE USR 59904). Znovu nastartovat lze přes RANDOMIZE USR 58419. Všechny uživatelské funkce lze používat i ze standardního systému.

OBSAH :