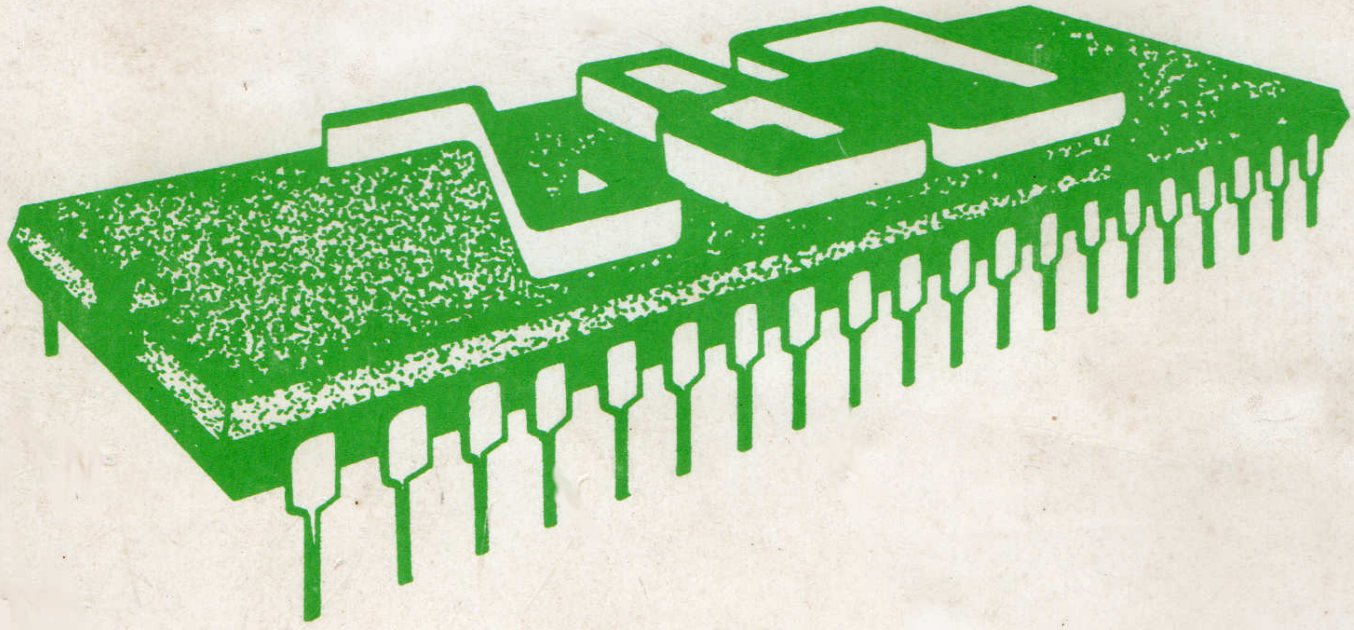
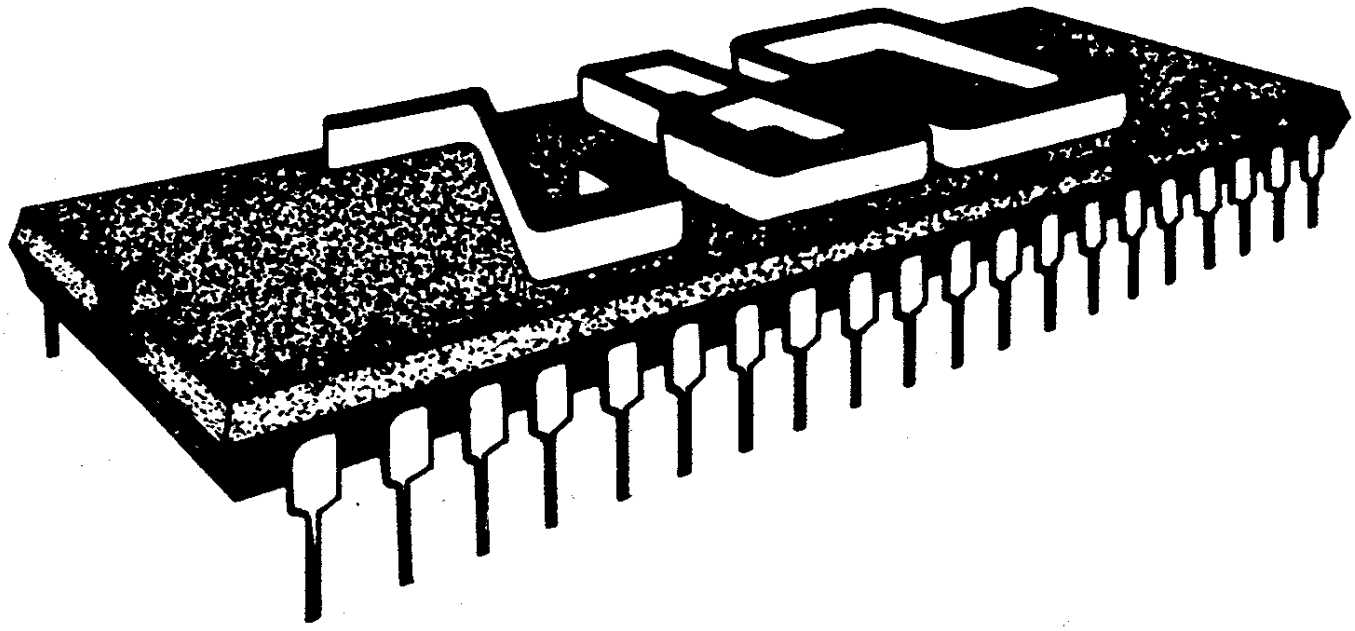
PODROBNÝ POHĽAD DO VÁŠHO POČÍTAČA ZX SPECTRUM



PODROBNÝ POHĽAD DO VÁŠHO POČÍTAČA ZX SPECTRUM



Táto publikácia je určená pre všetkých majiteľov niektorého z širokej rodiny počítačov vychádzajúcich zo štandartu SINCLAIR ZX SPECTRUM:

ZX Spectrum /+/128K/+2/+2A/+3, Delta, Didaktik GAMA, Didaktik M.

PODROBNÝ POHĽAD DO VAŠEHO POČÍTAČA ZX SPECTRUM

O B S A H

Vložte a spustite tento program. Čo robí?

```
10 DIM i$ (704)
20 PRINT AT RND*20,RND*31;CHR$ (RND*223+32)
30 PRINT AT 0,0; OVER 1; INVERSE 1;i$
40 GO TO 20
```

Za pár sekund sa niečo objaví na obrazovke a potom sa kompletne všetko invertuje. Kto potrebuje strojový kód? Bolo to prevedené tak, že sa vytlačila celá obrazovka medzier pomocou OVER a celá obrazovka sa zmenila pomocou INVERSE. Čo bolo čierne, je biele a čo bolo biele, je čierne. Normálne by ste očakávali, že OVER použije svoju vlastnosť XOR k vymazaniu nejakej časti, ale v retazci medzier nie je pre XOR nič na práci a tak prebehne inverzia obrazovky veľmi rýchlo. Pracuje to dobre v čiernej a bielej, avšak je ľahké pridať miestne riadiace príkazy farieb /PAPER, INK, FLASH a BRIGHT/, každý s parametrom 8 k zaisteniu celkovej súhry farieb. To všetko zaisťuje, že sú udržované rovnaké atribúty, pričom pôsobí INVERSE 1.

```
10 DIM i$ (704)
15 FOR i=1 TO 50
20 PRINT AT RND*20, RND*31; INK RND*7; PAPER RND*7;
   FLASH RND; BRIGHT RND; CHR$ (RND*223+32)
25 NEXT i
30 PRINT AT 0, 0; INVERSE 1; OVER 1; PAPER 8; INK 8;
   BRIGHT 8; FLASH 8; i$
```

Rovnaká myšlienka môže byť použitá pri obrátení celého textu a grafiky na obrazovke do jednotlivých farieb pri vynechaní príkazu INVERSE 1 (alebo napísaní INVERSE 0) a špecifikácii farby pomocou INK, ako ponechanie INK 8. Napr. tento program píše náhodné znaky na obrazovku v náhodných farbách INK a PAPER a potom ich všetky zmení na čierne, pretože drží atribúty BRIGHT, FLASH a PAPER stále rovnaké:

```
10 DIM i$ (704)
15 FOR i=1 TO 50
20 PRINT AT RND*20, RND*31; INK RND*7; PAPER RND*7;
   FLASH RND; BRIGHT RND; CHR$ (RND*223+32)
25 NEXT i
30 PRINT AT 0, 0; OVER 1; PAPER 8; INK 0; BRIGHT 8; FLASH 8;i$
```

Možno ste si všimli, že na riadku 20 vzniknú občas náhodne rovnaké farby INK a PAPER. To je problém. Problém? Nie, stačí len špecifikovať INK 9! A môžete čítať všetko.

```
10 DIM i$ (704)
15 FOR i=1 TO 50
20 PRINT AT RND*20, RND*31; INK RND*7; PAPER RND*7;
   FLASH RND; BRIGHT RND;CHR$ (RND*223+32)
25 NEXT i
30 PRINT AT 0,0; OVER 1; PAPER 9; INK 8; BRIGHT 8; FLASH 8;i$
```

To isté môžeme urobiť s PAPER. Pomocou špecifikácie farby PAPER a

ponechaním ostatných atribútov tak, ako sú, možno meniť farbu pozadia a pritom netreba na obrazovke nič meniť, alebo používať CLS. Treba poznamenať, že niečo napísané v tejto farbe môže zmiznúť, veď predsa napr. zelený text na zelenom papieri nie je zrovna ľahké prečítať. Tento príklad píše náhodné znaky náhodnou farebnou kombináciou a potom zmení papier na žltú:

```
10 DIM i$ (704)
15 FOR i=1 TO 50
20 PRINT AT RND*20, RND*31; INK RND*7; PAPER RND*7;
   FLASH RND; BRIGHT RND;CHR$ (RND*223+32)
25 NEXT i
30 PRINT AT 0,0; OVER 1; PAPER 6; INK 8; BRIGHT 8; FLASH 8;i$
```

S vyššie uvedeným programom môžete získať zaujímavé efekty s ktorou koľvek oblasťou, ktorá má u BRIGHT atribút 1. Ak potrebujete zvýrazniť používateľské výpisy pomocou BRIGHT 1 alebo FLASH 1 a potom zvýraznenie opäť zrušiť, môžete to urobiť nasledujúcou cestou:

Vypnutie jasu zjasnených miest:

```
10 DIM i$ (704)
15 FOR i=1 TO 50
20 PRINT AT RND*20, RND*31; INK RND*7; PAPER RND*7;
   FLASH RND; BRIGHT RND;CHR$ (RND*223+32)
25 NEXT i
30 PRINT AT 0,0; OVER 1; PAPER 8; INK 8; BRIGHT 0; FLASH 8;i$
```

Vypnutie blikania blikajúcich miest:

```
10 DIM i$ (704)
15 FOR i=1 TO 50
20 PRINT AT RND*20, RND*31; INK RND*7; PAPER RND*7;
   FLASH RND; BRIGHT RND;CHR$ (RND*223+32)
25 NEXT i
30 PRINT AT 0,0; OVER 1; PAPER 8; INK 8; BRIGHT 8; FLASH 0;i$
```

Všimnite si, že vo všetkých uvedených príkladoch boli "triky na obrazovke" uskutočnené na jednom riadku programu.

Zapamätajte si ! Základný princíp je reťazec 704 medzier písaných pomocou OVER cez celú obrazovku vo farbe 8. Táto technika otvára zaujímavé možnosti. Ak potrebujete nakresliť zložité tvary, čo ide normálne pomaly, nakreslite ich najskôr v rovnakej farbe INK ako PAPER, aby bol proces kreslenia neviditeľný, potom použite techniku uvedenú hore ku zmene farby Vášho obrazu, čím sa stane okamžite viditeľným. Skúste nasledujúci program, ktorý kreslí štyri sústredné fialové kružnice na žltom pozadí. Kreslenie trvá asi 4 sekundy:

```
 5 INK 3: PAPER 6: CLS
10 DIM i$ (704)
15 FOR i=10 TO 70 STEP 20
20 CIRCLE 120, 90, i
25 NEXT i
```


Prejdite nasledujúci program, ktorý spočiatku kreslí žlté kruhy na žltom pozadí a po ich nakreslení zmení farbu kruhov na fialovú. Uvidíte na niekoľko sekúnd čistú bielu obrazovku, ale ako sa začnú kružnice vyfarbovať, ide to takmer okamžite. Prakticky ste schopný oneskorenie zamaskovať, takže kreslenie prebieha akoby v zlomku sekundy.

```
 5 INK 6: PAPER 6: CLS
10 DIM i$ (704)
15 FOR i=10 TO 70 STEP 20
20 CIRCLE 120, 90, i
25 NEXT i
30 PRINT AT 0,0; INK 3; OVER 1;i$
```

Toto sú len dielčie výsledky jedného nápadu, ale využitie prepisovania reťazcom medzier k riadeniu atribútov celého súboru je rýchly a výkonný programovací nástroj s širšou škálou aplikácií.

Hovorili sme o využití reťazca plného medzier pre ovplyvnenie farieb celej obrazovky. Môžete však použiť len toľko medzier, koľko potrebujete. Napr. pre zmenu zelenej príšery na bielu, keď ju zasiahnete Vašou špeciálnou zbraňou:

```
PRINT AT y, x; OVER 1; INK 7;"*"
```

alebo vykonaním cyklu s farbením cez celé slovo:

```
10 PRINT AT 5,5;"HELP"
20 FOR f = 0 TO 1: FOR b = 0 TO 1: FOR p=0 TO 7:
   FOR i=0 TO 7
30 PRINT AT 5,5; OVER 1; FLASH f; BRIGHT b; PAPER p;
   INK i;"    ": REM V úvodzovkách 4 medzery
40 NEXT i: NEXT p: NEXT b: NEXT f
```

Ú N I K Z P R Í K A Z U I N P U T

Ak potrebujete zastaviť program v priebehu INPUT, funkcia BREAK je neúčinná a pridá len ku vstupným dátam medzeru.

V prípade numerického INPUTu, ako napr. INPUT A, je treba stlačiť STOP a potom ENTER. Program sa zastaví s hláškou H STOP in INPUT.

V prípade reťazca je tomu trocha inak. STOP môže byť kľudne použitý, ale kurzor musí byť prvým znakom na riadku. Tým je mienené, že sa musí dostať mimo úvodzovky, a to pomocou DELETE alebo CURSOR LEFT (CAPS SHIFT 5). Potom napísať STOP a ENTER a program sa zastaví s hláškou H STOP in INPUT.

Pri použití prostriedkov INPUT LINE je záležitosť problematickejšia. STOP je akceptovaný ako plne platný INPUT znak a nemôže zastaviť program. K úniku z INPUT LINE je nutné použiť CURSOR DOWN (CAPS SHIFT 8). Netreba stlačiť ENTER. Program sa zastaví s hláškou H STOP in INPUT, i keď STOP nebol vôbec použitý!

C A P S L O C K V P R O G R A M E

Bit 3 systémovej premennej 23658 (FLAGS2) indikuje stav CAPS LOCK, ak je zapnutý, bit 3 je 1, ak je vypnutý, bit 3 je 0. To je možné využiť

často pre detekciu, či bola stlačená správna klávesa v odpovedi na nejaku otázku, bez ohľadu na to, či je to malé alebo veľké písmeno (napr. YES alebo NO). Pre zapnutie CAPS LOCK je treba POKE 23658,8 a pre vypnutie POKE 23658,0, pričom je treba brať do úvahy, že to ovplyvni aj iné indikátory systémových premenných.

Skúste zapnúť ZX PRINTER a zadajte:

```
POKE 23658,2
```

Nezničíte tlačiareň ani papier alebo nič podobné, ale zároveň s tlačiarňou vôbec nepohnete žiadnym príkazom, až po novom zapnutí počítača. Z toho vidíte, že pri POKE 23658,x je treba zachovávať istú opatrnosť.

Tu je príklad rutiny pre ošetrenie ÁNO/NIE:

```
1000 PRINT "Chcete novú hru (A/N)?"
1010 POKE 23658,8
1020 IF INKEY $="A" THEN RUN
1030 IF INKEY $=N" THEN STOP
1040 GO TO 1020
```

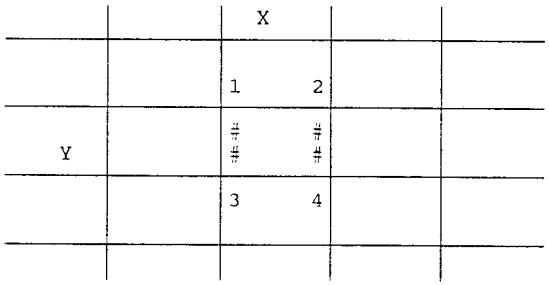
P A U S E A S L U Č K A F O R - T O - N E X T

S použitím PAUSE na Spectre nie sú bežné ťažkosti, ale pri nutnosti nastaviť nemenné oneskorenie môžu problémy nastať. PAUSE môže byť totiž skrátená stlačením akejkoľvek klávesy, takže doba ňou nastavená nie je potom dodržaná. Túto ťažkosť odstráni slučka FOR/NEXT, použitá ako oneskorovacia slučka. K dosiahnutiu oneskorenia asi 1 sekundy použite:

```
FOR a=1 TO 220: NEXT a
```

P O R O V N Á V A N I E S Ú R A D N Í C P R I N T A P L O T

Predpokladajme,že ste zadali PRINT AT y,x; súradnica y určuje umiestnenie podľa osi y, súradnica x určuje umiestnenie podľa osi x. Y je z intervalu <0,21> 0 je hore na obrazovke a 21 dole. X je z intervalu <0,31> 0 bude vľavo a 31 vpravo na obrazovke. Inými slovami povedané, štandartné PRINT súradnice.



Súradnice pre PLOT zodpovedajúce rohom 1 až 4 hore na obrázku (so značkou #) budú vo formáte PLOT x,y:

```
(1) x*8, (21-y)*8+7
(2) x*8+7, (21-y)*8+7
(3) x*8, (21-y)*8
(4) x*8+7, (21-y)*8
```

Z tohto budete schopný zistiť pozície všetkých pixelov (bodov jemnej grafiky) vo vnútri PRINT pozície, ak potrebujete PLOT alebo DRAW cez znamu PRINT pozíciu.

V Y T V O R E N I E K A T A L Ó G U P R O G R A M O V U L O Ž E N Ý C H N A P Á S K E

Ak máte plnú pásku programov a nemáte poňatia o ich menách alebo typoch (BASIC program, data alebo bytes), potom nepochybne pomôže výpis všetkých mien a typov na obrazovke, bez toho aby sa stratilo to, čo je práve v pamäti počítača.

Typy záznamov získaných touto cestou sú:

Character Array: A$

vytvorené príkazom SAVE "A" A$ ()

Number Array: A

vytvorené príkazom SAVE "A" A()

Program: meno vytvorené pomocou SAVE "meno"
Bytes:code vytvorené pomocou SAVE "code" CODE adresa,dĺžka

K získaniu výsledku použite VERIFY a meno programu, ktorý sa celkom určite na páske nenachádza. Napr. VERIFY "ZZZZZZZ". Najskôr sa ale uistite, že na obrazovke nie je žiadne meno programu alebo podobné, čo by Vás mohlo zmýliť (pred VERIFY urobte CLS).

Z O T R E T I E Č A S T I O B R A Z O V K Y

INPUT môže byť využitý aj ináč, než pre aktuálny vstup premenných. Môže byť použitý pre vymazanie spodnej časti obrazovky, pokiaľ INPUT AT nenarazí na PRINT pozície (potom začne obrazovka "rolovať nahor"). Použite INPUT AT ako v nasledujúcom programe, pričom si pamätajte, že INPUT AT súradnice začínajú na obrazovke dole:

```
10 INPUT "Koľko ?", k
20 FOR a = 0 TO 21: PRINT a: NEXT a
30 PRINT AT 0,0;
40 INPUT AT k,0;
```

Keď INPUT AT dosiahne existujúce PRINT pozície, začne obrazovka

rolovať. Môžete tlačové pozície uschovať, odstrániť tlač z cesty, vymazať spodok obrazovky a potom vrátiť tlač späť:

```
10 FOR a=0 TO 21: PRINT AT a,0;a;: NEXT a
20 LET x=33-PEEK 23688
30 LET y=24-PEEK 23689
40 PRINT AT 0,0;
50 INPUT "Koľko ?", jm
60 INPUT AT jm+1, 0;
70 PRINT AT y,x;
```

Riadok 10 čosi vytlačí zhora nadol. Riadky 20 a 30 uchovávajú súradnice kurzora po PRINT pozícii. Riadok 40 presúva kurzor k hornej časti obrazovky z cesty výmazu. Riadok 50 sa pýta, koľko z 22 riadkov chcete zdola vymazať. Ak dáte napr. 1, bude vymazaný len riadok 21. Programový riadok 60 prevádza výmaz a riadok 70 vracia späť tlačové pozície, premiestni kurzor na svoju pôvodnú pozíciu pred zotieraním .

O B R A Z O V K A A J E J P O H Y B

Niekedy je "rolovanie" obrazovky potrebné ešte skôr, než tak začne prevádzať počítač sám. V BASICu je nutné vytvoriť reťazcovú premennú (string), ktorá je schopná uchovať všetkých 22*32 znakov obrazovky. Rolovanie sa prevádza rotáciou prvkov premennej. Rolovanie nahor a dole ide skutočne rýchlo.

```
 1 REM Rolovanie nahor
10 DIM a$ (704)
20 INPUT a$
30 PRINT AT 0,0;a$
40 LET a$=a$ (33 TO)+"32 medzier"
50 GO TO 30
```

```
 1 REM Rolovanie dole
10 DIM a$ (704)
20 INPUT a$
30 PRINT AT 0,0; a$
40 LET a$="32 medier"+ a$ (TO 672)
50 GO TO 30
```

Rolovanie do strán ide pomalšie, ale napriek tomu môže byť potrebné:

```
 1 REM Rolovanie vľavo
10 DIM a$ (704)
20 INPUT a$
30 PRINT AT 0,0;a$
40 FOR f=1 TO 673 STEP 32
50 LET a$ (f TO f+31)=a$ (f+1 TO f+31)+" "
60 NEXT f
70 GO TO 30
```

```
 1 REM Rolovanie vpravo
10 DIM a$ (704)
20 INPUT a$
30 PRINT AT 0,0; a$
40 FOR f=1 TO 673 STEP 32
50 LET a$ (f TO f+31)=" " + a$ (f TO f+30)
60 NEXT f
70 GO TO 30
```


S A D A Z N A K O V

V tejto časti sa budeme zaoberať všetkými spôsobmi využitia sady znakov uložených v ROM, potom túto sadu prevedieme z ROM do RAM (obrazne povedané) a vytvoríme vlastnú sadu (tvary znakov).

Ale najskôr pre predvedenie dôležitej systémovej premennej skúste toto:

```
POKE 23606,8 a ENTER
```

Teraz skúste napísať nejaký program - ak to dokážete ! Teraz vyresetujte počítač (pomocou napájania) a spustite tento krátky program:

```
10 POKE 23606,8
20 PRINT INKEY$
30 IF INKEY$=" " THEN POKE 23606,0: STOP
40 GO TO 20
```

Spustite program a skúste niečo napísať. Čo sa stane? Písmeno A vyjde von ako B, B ako C atď. Môžete to skúsiť u Vášho miestneho predavača počítačov a potom mu povedať, že predáva pokazený tovar. V skutočnosti, akonáhle stlačíte SPACE, tak program všetko vráti do normálneho stavu. Ešte zábavnejšie je, ak zmeníte riadok 10 na:

```
10 POKE 23606,4
```

Tak napol popletiete i sami seba!

Vysvetlenie:

23606 a 23607 sú adresy systémovej premennej, ktorá počítaču hovorí, kde je uložená zostava dát, tvoriacich znaky, ktoré sa majú objaviť na obrazovke. Táto dvojbajtová systémová premenná obsahuje číslo, a to štandartne 15360, ktoré je po zapnutí počítača o 256 menšie ako je počiatočná adresa súboru znakov v ROM, Takže 15360+256=15616. To je adresa, kde začína v pamäti ROM sada znakov. Urobme si PEEK, aby sme videli čo je tam.

Spustite nasledujúci program:

```
10 FOR a=15616 TO 15639
20 PRINT PEEK a
30 NEXT a
```

```
0  0  16 0  0
0  0  16 0  0
0  0  16 36 0 výpis
```

```
0  0  0  36 0
0  16 16 0
```

To nie je zrovna veľmi informatívne, že? Číslo 15639 nebolo dôležité, zvolili sme ho, aby sme získali krátky výpis.

Ak sa pozrieme sa zoznam znakov v manuáli, nevidíme žiadnu suvislosť s naším výpisom. No dobre, budeme si lámať našu nechápavú hlavu ďalej.

Skúsme to dvojkovo (binárne), snáď nám počítač ponúkne niečo k využitiu. Zadajte a spustte tento program, pričom nezabudnite na dva apostrofy v riadku 70 !

```
10 FOR a=15616 TO 15639
20 LET p=PEEK a
30 FOR b=7 TO 0 STEP -1
40 PRINT AT 21,b;p-2*INT (p/2)
50 LET p=INT (p/2)
60 NEXT b
70 PRINT AT 21,12; a ''
80 NEXT a
```

```
00000000  15616     00010000  15628
00000000  15617     00000000  15629
00000000  15618     00010000  15630
00000000  15619     00000000  15631
00000000  15620     00000000  15632
00000000  15621     00100100  15633
00000000  15622     00100100  15634
00000000  15623     00000000  15635
00000000  15624     00000000  15636
00010000  15625     00000000  15637
00010000  15626     00000000  15638
00010000  15627     00000000  15639
```

Toto nevyzerá o nič lepšie, i keď skúsme vypísať ľavý stĺpec bez núl:

```
      15616     1    15628
      15617          15629
      15618     1    15630
      15619          15631
      15620          15632
      15621     1 1  15633
      15622     1 1  15634
      15623          15635
      15624          15636
1     15625          15637
1     15626          15638
1     15627          15639
```

To, čo vidíte, je medzera, výkričník a úvodzovky (ak máte poriadnu predstavivosť). V našom ďalšom triku prezrieme celý znakový generátor, aby sme videli, čo sa skrýva v jeho temných hlbinách. Pre dosiahnutie lepšieho výpisu na obrazovke použijeme # miesto 1.

```
10 FOR a=15616 TO 16383
20 LET p=PEEK a
```



```
30 FOR b=7 TO 0 STEP -1
40 PRINT AT 21,b;"#" AND (p-2*INT (p/2)=1)
50 LET p=INT (p/2)
60 NEXT b
70 PRINT AT 21,12;a ''
80 NEXT a
```

Ak máte tlačiareň a chcete výpis, spustite nasledujúci program, ktorý Vám poskytne kompletný výpis všetkých znakov z generátora v ROM.

Upozornenie: Budete potrebovať dosť papiera. Ak sa priskoro minie, alebo chcete tlač prerušiť, stlačte BREAK.

```
10 FOR a=15616 TO 16383
20 LET p=PEEK a
30 DIM a$ (8)
40 FOR b=8 TO 1 STEP -1
50 IF p-INT (p/2)*2 THEN LET a$(b)="#"
60 LET p=INT (p/2)
70 NEXT b
80 LPRINT a$;TAB 10;a;TAB 18;PEEK a
90 NEXT a
```

Teraz by Vás malo napadnúť, že generátor znakov uchováva jednotlivé znaky bit po bite tak, ako majú vyzerať na obrazovke. Ak porovnáte obrazovku alebo vytlačenie s Appendixom A v manuáli, vidíte, že znaky sú v oboch prípadoch v rovnakom poradí (samozrejme v rozsahu 32-127). Tie ostatné sú buď riadiace znaky (kde nie je čo zobraziť), grafické znaky (ktoré sú uložené inde), "bloková" grafika (ktorá sa "vypočítava", miesto aby bola uložená ako vzorka) alebo kombinácia znakov (kľúčové slová), ktoré sú v ROMe uložené inde, v inej tabuľke, ktorá určuje ich kombináciu.

To nie je náhoda, že znaky sú usporiadané v ich numerickom poradí. Je to pre uľahčenie hľadania, keď ich počítač potrebuje. Pre porozumenie celej veci sa pozrime, ako je znak organizovaný z hľadiska obrazovky, napr. číslo 8:

```
........
..XXXX..
.X....X.
..XXXX..
.X....X.
.X....X.
..XXXX..
........
```

Akýkoľvek znak je tvorený maticou 8x8 pixelov (obrazové bunky) na obrazovke. Našťastie je 8 bitov v jednom bajte (to je ale náhoda, čo?). Teda, ak jeden bajt, zložený z 8 bitov reprezentuje jeden riadok naprieč znakom, potom uložíme vzorku celého znaku v 8 bajtoch.

Presne tak pracuje generátor znakov: má pre každý znak uložených 8 bajtov, v ktorých je vzorka znakov zložená z núl a jedničiek. Jedničky označujú miesta, ktoré na obrazovke budú mať farbu INK, nuly potom miesta s farbou pozadia PAPER.

Takto je napr. uložená v ROM osmička (ľavý stĺpec bitové vzorky,

stredná je adresa v ROM a pravý stlpec je desiatková hodnota ľavého stlpca):

```
00000000  15808      0
00111100  15809     60
01000010  15810     66
00111100  15811     60
01000010  15812     66
01000010  15813     66
00111100  15814     60
00000000  15815      0
```

Z toho vyplýva pre nás užitočná možnosť. Môžeme ktorýkoľvek znak zväčšiť. Ak použijeme miesto každej 1 plný štvorček (#), môžeme znaky zväčšiť osemkrát. Napr.:

```
................
..XXXX....XXXX..
.X...XX..X....X.
.X..X.X.....XX..
.X.X..X.......X.
.XX...X..X....X.
..XXXX....XXXX..
................
```

Skúsme to! Program je trochu odlišný od toho, ktorý sme už použili, preto ho študujte pozorne.

```
 10 LET naprieč=0
 20 LET dole=0
 30 INPUT a$
 40 LET c=CODE a$
 50 FOR k=0 TO 7
 60 LET p=PEEK (15360+c*8+k)
 70 FOR f=0 TO 7
 80 PRINT AT dole+k, naprieč+7-f;"#" AND (p-2*INT(P/2)=1)
 90 LET p=INT (p/2)
100 NEXT f
110 NEXT k
120 IF naprieč+8>31 THEN LET dole=dole+8
130 LET naprieč=naprieč-8 AND naprieč+8<=31
140 GO TO 30
```

Dve premenné naprieč a dole určujú miesto, kde bude zväčšovaný znak umiestnený na obrazovke. a$ je zadaný zväčšovaný znak. Musí to byť niektorý zo zobraziteľných znakov, ich CODE je medzi 32 a 127 (tj. SPACE až (c) - copyright symbol). Premenná c je ich CODE, t.j. poradie v ASCII tabuľke. V riadku 60 použité číslo 15360 označuje vrch tabuľky znakov. Spomínate si, že toto číslo je o 256 menšie ako počiatočná adresa tabulky?

Prečo?

Dobre, prvý znak v tabulke je SPACE, čo je CHR$ (32). Spomínate, že

každý znak je uložený v 8 bajtoch, takže sa všetko násobí 8? A 8x32 je 256 a 15360+256 je 15616, čo je počiatočná adresa tabuľky v ROM. Pri každej slučke sa prenos dát delí dvoma a nachádza sa tak zvyšok po delení 2, ktorý slúži k tomu, aby určil, či miesto na obrazovke bude tmavé alebo svetlé. Tým sú nastavené hodnoty premenných dole a naprieč. Môžete k programu pridať riadok, ktorý zaisťuje, že znak, ktorý tento program nemôže obslúžiť, bude ignorovaný.

```
35 IF CODE a$<32 OR CODE a$>127 THEN GOTO 30
```

Produkované znaky sú veľké a vojde sa ich na obrazovku len niekoľko. Nasleduje program, ktorý využíva PLOT a DRAW k vytvoreniu znakov rôznych veľkostí.

```
  1 REM Znaky
 10 INPUT "Koľkonásobne rozšíriť (1=normál)?";wider
 20 INPUT "Koľkonásobne zvýšiť (1=normál)?";taller
 30 LET across=wider*8-1: LET down=176
 40 INPUT a$: IF a$ < " " OR a$ > "(c)" THEN GO TO 40
 50 FOR a=0 TO 7
 60 LET peek=PEEK (15360+CODE(a$)*8+a)
 70 FOR b=0 TO 7
 80 IF peek-2*INT (peek/2) THEN FOR t=1 TO taller: PLOT
    across-b *wider, down-a*taller-t: DRAW 1-wider,0: NEXT t
100 LET peek=INT (peek/2): NEXT b
110 NEXT a
120 LET across=across+wider*8
130 IF across>255 AND down-taller*8>taller*8-1 THEN
    LET down=down-taller*8: LET across=wider*8-1
140 IF across>255 AND down-taller*8<taller*8 THEN PRINT
    AT 21,31': FOR a=1 to taller: PRINT: NEXT a: LET across=wider*8-1
150 GO TO 40
```

Program je komplikovaný, študujte teda nasledujúce informácie pozorne. Pri spúšťaní programu budete najskôr opýtaní koľkokrát má byť znak na obrazovke širší než normálne.

Ak chcete napr. 3x, potom zadajte 3 a ENTER. Ak chcete normálnu šírku, zadajte 1. To isté platí, až budete opýtaní, koľkokrát má byť znak dlhší (vyšší). Výpis začne vľavo hore a pokračuje naprieč a dole, až dosiahne spodnú hranicu obrazovky. Potom urobí Scroll, aby si uvoľnil potrebný počet riadkov, pokiaľ nie je so znakom hotový. Program beží v čiernej a bielej (alebo v trvalej farbe INK a PAPER, ktorú si zadáte) alebo môžete dodatočne použiť príkazy pre farby ako v iných Vašich programoch.

Názvy premenných taller (dlhší) a wider (širší) sú použité v plnej dĺžke, aby bol ich význam zreteľnejší. Ti isté platí i pre premenné peek (je malým písmom pre odlíšenie od kľúčového slova PEEK) a across (naprieč) a down (dole). Tieto slúžia ako súradnice pre príkaz PLOT, použitý ďalej.

176 je o 1 väčšie ako je limitná hodnota 175 pre PLOT. Nebojte sa, chyba však nevznikne. Across začína s nejakou hodnotou vo vnútri obrazovky, pretože PLOT a DRAW prebieha z prava do ľava. Ich hodnoty sú totiž vypočítavané binárnou cestou.

Riadok 40 sleduje, ktorý znak alebo symbol si prajete upraviť. Tento program využíva INPUT, i keď INKEY$ by Vám ušetrilo stláčanie ENTER. Niektoré znaky sú však pre INKEY$ len ťažko dostupné.

Riadok 50 štartuje slučku, ktorá vyhľadá v generátore znakov 8 bajtov príslušných pre Vami zvolený znak. Tieto bajty sú nájdené v riadku 60.

Slučka začínajúca na riadku 70 určuje vzorku znaku a dozerá, či sú správne miesta tmavé alebo svetlé. To všetko sa robí pri DRAW riadku na obrazovke, ktorá je wider krát dlhšia ako pixel. Tento DRAW sa prevádza taller krát, pre dosiahnutie požadovanej výšky.

Riadok 100 delí hodnotu peek dvoma, pretože je potrebné testovať nasledujúci bit.

Riadok 120 nastavuje novú hodnotu across (ak je vpravo mimo obrazovku) a novú hodnotu down, ak nie je dole dosť miesta pre znak (pomocou PRINT urobí potrebný priestor - scroll). Across je nastavené do východzej hodnoty pre nový riadok znaku. Po tomto všetkom je opäť prezretá klávesnica, ak je treba zväčšovať ďalší znak a celý proces sa opakuje.

Program pracuje najlepšie s celočíselnými hodnotami across a down. Tieto však nemusia byť celočíselné, ako je vidieť zo vzorca:

$$\text{Across} = \frac{\text{počet znakov normálne na riadku}}{\text{počet znakov požadovaných na riadku}}$$

napr. pre 40 znakov na riadku bude across 32/40=0,8.

Použitím rovnakého princípu možno dosiahnuť i zrkadlového obrazu. Toto je docielené kreslením v opačnom smere.

```
 1 REM Zrkadlo
10 INPUT "Koľkonásobne rozšíriť (1=normál)?";wider
20 INPUT "Koľkonásobne zvýšiť (1=normál)?";taller
30 LET across=0: LET down=176
40 INPUT a$: IF a$ < " " OR a$ > "(c)" THEN GO TO 40
50 FOR a=0 TO 7
60 LET peek=PEEK (15360+CODE a$*8+a)
70 FOR b=0 TO 7
80 IF peek-2*INT (peek/2) THEN FOR t=1 TO taller: PLOT across+
   b*wider, down-a*taller-t: DRAW 1-wider,0: NEXT t
100 LET peek=INT (peek/2): NEXT b
110 NEXT a
120 LET across=across+wider*8
130 IF across>256-wider*8 AND down-taller*8>taller*8-1
   THEN LET down=down-taller*8: LET across=0
140 IF across>256-wider*8 AND down-taller*8<taller*8
   THEN PRINT AT 21,31': FOR a=1 TO taller: PRINT:
   NEXT a: LET across=0
150 GO TO 40
```

Ďalšou úlohou je včleniť tento program ako podprogram do Vášho vlastného programu:

```
 1 REM Subroutine
 4 LET wider=2
 8 LET taller=4
12 LET across=35
16 LET down=110
20 LET a$="Demonštrácia"
24 GO SUB 40
```



```
30 STOP
40 FOR d=1 TO LEN a$
50 FOR a=0 TO 7
60 LET peek=PEEK (15360+CODE a$(d)*8+a)
70 FOR b=0 TO 7
80 IF peek-2*INT (peek/2) THEN FOR t=1 TO taller: PLOT
   across-b*wider, down-a*taller-t: DRAW 1-Wider,0:
   NEXT t
90 LET peek=INT (peek/2): NEXT b
100 NEXT a
110 LET across=across+wider*8
120 IF across>255 AND down-taller*8 >taller*8-1 THEN
   LET down=down-taller*8: LET across=wider*8-1
130 IF across>255 AND down-taller*8<taller*8 THEN
   PRINT AT 21,31': FOR a=1 TO taller: PRINT: NEXT a:
   LET across=wider*8-1
140 NEXT d
150 RETURN
```

Pred volaním podprogramu, ktorý je v riadkoch 40 až 160, musíte určiť štyri premenné (wider, taller, across a down) tak, ako ich potrebujete vo Vašom programe. Across a down pritom určujú PLOT súradnice pravého horného rohu prvého znaku. A$ je reťazec, obsahujúci zväčšované znaky. Zväčšenie prebieha v slučke d. Musíte však zaistiť, že A$ neobsahuje nedovolené znaky alebo pridať k programu nasledujúci riadok:

```
45 IF a$ (d)<" " OR a$ (d) >"(c)" THEN GO TO 150
```

Tento podprogram môže tiež zväčšovať používateľom definovanú grafiku, keď pridáme riadok, ktorý určuje, odkiaľ sa odvodzuje hodnota premennej peek.

```
 1 REM Subroutine
 4 LET wider=2
 8 LET taller=4
12 LET across=35
16 LET down=110
20 LET a$="Demonštrácia"
24 GO SUB 40
30 STOP
40 FOR d=1 TO LEN a$
50 FOR a=0 TO 7
60 IF CODE a$(d)>31 AND CODE a$ (d)<128 THEN LET peek=PEEK
   (15360+CODE a$(d)*8+a)
61 IF CODE a$ (d) > 143 AND CODE a$(d) < 165 THEN LET peek=
   PEEK "a"+(CODE a$ (d)-144)*8+a
70 FOR b=0 TO 7
80 IF peek-2*INT (peek/2) THEN FOR t=1 TO taller: PLOT
   across-b*wider, down-a*taller-t: DRAW 1-wider,0: NEXT t
90 LET peek=INT (peek/2): NEXT b
100 NEXT a
110 LET across=across+wider*8
120 IF across>255 AND down-taller*8>taller*8-1 THEN
   LET down=down-taller*8: LET across=wider*8-1
130 IF across>255 AND down-taller*8<taller*8 THEN
```

```
    PRINT AT 21, 31': FOR a=1 TO taller: PRINT: NEXT a:
    LET across=wider*8-1
140 NEXT d
150 RETURN
```

Našou ďalšou úlohou je vytvoriť v RAM novú sadu znakov, ktorú môžeme používať bez problémov v programoch, výpisoch a pod., rovnako ako sadu uloženú trvale v ROM.

Manuál k počítaču tvrdí, že je to možné, ale veľmi málo sa zmieňuje o tom, ako. Nová sada znakov bude v pamäti uložená nad RAMTOP pred UDG.

Nasledujúci popis nás povedie krok za krokom, ako a čo je treba urobiť. Všetko závisí na skutočnosti, že systémové premenné 23606 a 23607 ukazujú na začiatok sady znakov. Kde je to treba, sú udané adresy pre 16K RAM a pre 48K RAM, pričom je priložený i návod, ako adresy vypočítať pre rôzny rozsah pamäte.

Nová sada znakov bude tzv. vpravo sklonená italika. Predefinované boli len číslice 0123456789 a písmená, pričom je ďalej uvedené, ako možno predefinovať i ostatné znaky.

V tomto momente zostávajú symboly ako $, # ! % nezmenené. Môžete to však urobiť dodatočne.

Tento program napodobuje obyčajný písací stroj. Má i schopnosť vymazať posledný zadaný znak (stlačiť DELETE). Po stlačení ENTER začína nový riadok textu.

```
 5 BORDER 0
10 PRINT INKEY$;
20 IF INKEY$<>"" THEN GO TO 20
30 IF INKEY$="" THEN GO TO 30
34 BEEP .01,25
37 IF INKEY$=CHR$ 12 THEN PRINT CHR$ 8;" ";
   CHR$ 8;: GO TO 20
40 GO TO 10
```

Krok 1:

Prvým krokom je zníženie RAMTOP o 768 bytov, čím sa medzi časťou pamäte, vyhradenú pre program v BASICu a začiatkom UDG vytvorí priestor 768 bytov, ktoré sú potrebné pre novú sadu znakov.

16K Spectrum: Nový RAMTOP musí byť 31831, na rozdiel od normálnej hodnoty 32599. K zníženiu RAMTOP zadajte priamo príkaz CLEAR 31831.

48K Spectrum: Nový RAMTOP bude 64599 - starý bol 65367. Zadajte priamy príkaz CLEAR 64599.

Obe tieto verzie príkazov sú v absolútnych číslach. Ak máte pripojený iný rozsah pamäte, potom musíte tieto čísla nahradiť výrazom, ktorý umožní vypočítať adresu odpovedajúcu okolnostiam. Priamy príkaz potom bude:

```
CLEAR (PEEK 23730+256*PEEK 23731-768)
```

Ak nebude na nasledujúcich stránkach uvedená príslušná adresa pre 48K Spectrum, možno ju ľahko zistiť z adresy pre 16K Spectrum, a to pripočítaním hodnoty 32768 k tejto adrese (udanej pre 16K). Toto platí vo väčšine prípadov, vyjmúc samozrejme systémové premenné. Nasledujúce

diagramy ukazujú nové rozloženie pamäte:

Pre 16 Kb Spectrum platia v nasledujúcej tabuľke hodnoty menšie o 32768.

48 Kb Spectrum

	RAMTOP 64599	RAMTOP + 1	USR "A" 65368
koniec pamäte pre Basic	62 dec. 3E hex.	nová znaková sada	užívateľom definovaná grafika

Krok 2:

Teraz musíme stavajúcu sadu znakov prekopírovať z ROM do uvoľneného priestoru v RAM, aby sme ich mohli zmeniť na znaky, aké chceme (v ROM to nejde, tam sú "napevno", možno ich len čítať).

16K Spectrum

```
10 FOR a=15616 TO 16383
20 POKE 16216+a, PEEK a
30 NEXT a
```

48K Spectrum

```
10 FOR a=15616 TO 16383
20 POKE 48984+a, PEEK a
30 NEXT a
```

Je dôležité previesť toto presne, pretože akákoľvek chyba by bola neskôr len veľmi ťažko opraviteľná a museli by sme začať znovu.

Krok 3:

Teraz začneme s predefinovávaním. Najskôr Vám sa dozvieme ako zmeniť čísla, veľké a malé písmena. Najskôr zmeníme čísla. Napíšte tento program:

16K

```
10 FOR a=1 TO 80
20 INPUT b
30 POKE 31959+a,b
40 NEXT a
```

48K

```
10 FOR a=1 TO 80
20 INPUT b
30 POKE 64727+a,b
40 NEXT a
```

Spravte RUN zapísaného programu a zapíšte do INPUTu nasledujúce dáta.

Zapisujte najskôr 1. rad zľava doprava, potom 2 rad atď. (ak budete zapisovať stĺpec, potom marite zbytočne svoju energiu).

Dáta - pre obidva programy rovnaké:

```
0    60   70   74  148  164  120   0
0    48   80   16   32   32  248   0
0    28   34    4   56   64  124   0
0    30    4   24    4   72   56   0
0     6   10   20   36  126    8   0
0    30   16   60    2   68   56   0
0    30   32   60   66   68   56   0
0    30    2    4    8   16   32   0
0    28   36   56   68   68   56   0
```

Poznámka - teraz ešte neuvidíte žiadny efekt Vašej práce, pretože príslušné systémové premenné zmeníme až neskôr.

Krok 4:

Teraz premeníme veľké písmena. Použijeme príslušný program a dáta, tak ako nasledujú:

16K

```
10 FOR a=1 TO 208
20 INPUT b
30 POKE 32095+a,b
40 NEXT a
```

48K

```
10 FOR a=1 TO 208
20 INPUT b
30 POKE 64863+a,b
40 NEXT a
```

Dáta

```
0    12   18   34   62   66   66   0
0    28   18   60   34   66  124   0
0    28   34   32   64   68   56   0
0    24   20   34   34   68  120   0
0    30   16   60   32   64  120   0
0    30   16   60   32   64   64   0
0    28   34   32   76   68   56   0
0    18   18   60   36   72   72   0
0    62    8   16   16   32  248   0
0     2    2    4   68   72   56   0
0    18   20   56   40   68   66   0
0    16   16   32   32   64  124   0
0    34   54   42   66   68   68   0
0    18   26   42   44   68   68   0
0    28   34   34   68   68   56   0
0    28   18   34   60   64   64   0
```



```
0    60   66   66  164  148  120   0
0    28   18   34   60   68   66   0
0    28   34   24    4   68   56   0
0    62    8    8   16   16   32   0
0    17   34   34   68   68   56   0
0    34   34   36   36   40   16   0
0    33   33   66   66   90   36   0
0    34   20   24   56   68  130   0
0    34   20    8   16   32   64   0
0    62    4    8   16   32  124   0
```

Krok 5:

Teraz malé písmená. Použijete nasledujúci príslušný program a dáta:

16K

```
10 FOR a=1 TO 208
20 INPUT b
30 POKE 32351+a,b
40 NEXT a
```

48K

```
10 FOR a=1 TO 208
20 INPUT b
30 POKE 65119+a,b
40 NEXT a
```

Dáta

```
0     0   12    2   60   68   56   0
0    16   16   60   34   66  124   0
0     0   28   32   32   64   56   0
0     2    2   28   36   68   56   0
0     0   28   34  124   64   56   0
0     6    8   12   16   16   32   0
0     0   14   18   34   60    4 120
0    16   16   62   34   68   68   0
0     4    0   24    8   16  120   0
0     2    0    4    4    8   72  48
0    16   20   56   48   72   68   0
0     8   16   16   32   32   24   0
0     0   54   73   73  146  146   0
0     0   60   34   34   68   68   0
0     0   28   34   36   68   56   0
0     0   28   18   34   60   64  64
0     0   30   18   36   60    8  30
0     0   14   16   16   32   32   0
0     0   30   32   24    4  120   0
0     4   30    8   16   18   12   0
0     0   34   34   68   68   56   0
0     0   34   34   36   40   16   0
0     0   65   65  146  146  108   0
0     0   34   20   24   40   68   0
```

```
0     0   18   36   60    8   16  96
0     0   60    8   16   32  120   0
```

Krok 6:

Využitie. Určite Vás teší, že sa blížime k záveru našej práce. Aby sme mohli novú sadu znakov využívať, musíme zmeniť honotu uloženú v systémovej premennej 23606/7 CHARS (viď Spectrum manuál, hlava 25)., ktorá predstavuje počiatočnú adresu sady znakov. Táto narmálna hodnota je o 256 menšia než je počiatočná adresa prvého bajtu generátorov znakov, čím je umožnené ľahké nájdenie adresy dát pre každý znak.

Ako?

Je to umožnené jednoduchou matematickou manipuláciou pomocou CODE, pričom je adresa jednotlivého znaku nájdená podľa hodnoty uloženej v 23606/7 a CODE*8 hľadaného znaku. Pri použití generátora znakov v ROM majú systémové premenné hodnotu 0 (23606) a 60 (23607), a to pri oboch verziách pamäte (16KB a 48KB):

PEEK 23606 je 0
0+256*60 je 15360
PEEK 23607 je 60

Aby sme tento ukazovateľ počiatočnej adresy nastavili na našu novú sadu znakov, urobíme:

16K POKE 23606,88: POKE 23607,123

48K POKE 23606,88: POKE 23607,251

Je lepšie vložiť oba príkazy ako jeden dlhý príkaz, pretože pri vkladaní po častiach by nebolo druhý príkaz na obrazovke vidieť.
Od tejto chvíle, pokiaľ ste neurobili chybu, sa všetko, čo zapíšete na obrazovke, objaví v nových znakoch (samozrejme, že to, čo na obrazovke už bolo skôr, zostáva nezmenené).
Ak chcete zmeniť znaky späť na pôvodné, generované v ROM, potom stačí urobiť:

POKE 23606,0: POKE 23607,60

Oba druhy "prvkov" (t.j. aj nové znaky aj staré) môžete použiť v riadkoch Vášho programu a môžete tak ľubovoľne kombinovať na obrazovke staré a nové znaky (pokiaľ však znaky neprepisujete - OVER alebo nemažete).
V nových znakoch môžete získať i výpis na tlačiarni. Nové znaky majú na tlačiarni ZX veľkú výhodu - pomáhajú ukryť rozdiely v kvalite tlače, ktoré sa objavujú pri normálnych znakoch.
Nový vzhľad však vedie i k problémom. SCREEN$ identifukuje znaky na obrazovke tak, že v charakter set generátora v ROM hľadá odpovedajúce znaky. Tie však v tomto prípade nenájde a ako výsledok svojej činnosti vráti prázdny reťazec. SCREEN$ musí totiž nazerať do rovnakého generátora, ako je ten, ktorý znaky generoval na obrazovku. To je ale jediný problém, ktorý vzniká, keď používate dve, alebo viac sád znakov.

Krok 7:

SAVE - uloženie na páske a spätný výber - LOAD.

Skôr ako budete robiť čokoľvek iného, uchovajte novú sadu znakov na pásku, aby ste ich mohli nahrať z kazety, kedykoľvek to bude treba.

SAVE: na 16KB priamy príkaz

SAVE "chars" CODE 31832,768

na 48KB priamy príkaz
SAVE "chars" CODE 64600,768

LOAD: na 16KB priamy príkaz

CLEAR 31831: LOAD "chars" CODE 31832,768

na 48KB priamy príkaz
CLEAR 64599: LOAD "chars" CODE 64600,768

Nezabudnite po SAVE urobiť VERIFY, pretože ak sa stalo niečo pri SAVE (napr. chyba na páske), potom Vás čaká spúšta práce od začiatku.

Znaky vyzerajú takto:

Stará (normálna ROM sada znakov)

```
  1 " # $ % ^ ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
  A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [ \ ] + _
  a b c d e f g h i j k l m n o
p q r s t u v w x y z { / } " (c)
```

Nový (alternatívny RAM charakter set)

To isté, ale v italike.

Okrem funkcie SCREEN$ pracujú obe verzie presne rovnako:

Kľúčové slová a všetko ostatné sa objavuje v týchto znakoch, ktoré sú v danej dobe v činnosti. My sme doposiaľ predefinovali len číslice a písmená. Ak chcete predefinovať i ostatné znaky, urobte nasledujúce. Urobte všetko do kroku 2 vrátane. Potom napíšte tento program (2 verzie pre 16KB a 48KB):

```
16KB  5 POKE 23606,88: POKE 23607,126
     10 INPUT "Ktorý znak si praješ zmeniť ?";c$
     20 IF c$=" " THEN STOP
     30 IF c$ <" " OR c$>"(c)" THEN GO TO 10
     40 LET c=CODE c$
     50 FOR a=0 TO 7
```

```
    60 INPUT ("Aká hodnota pre riadok ";a+1;" ?");value
    70 POKE 31576+c*8+a, value
    80 PRINT AT 0,0;c$
    90 NEXT a
   100 GO TO 10

48KB  5 POKE 23606,88: POKE 23607,251
    10 INPUT "Ktorý znak si praješ zmeniť ?";c$
    20 IF c$=" " THEN STOP
    30 IF c$ <" " OR c$>"(c)" THEN GO TO 10
    40 LET c=CODE c$
    50 FOR a=0 TO 7
    60 INPUT ("Aká hodnota pre riadok ";a+1;" ?");value
    70 POKE 64344+c*8+a, value
    80 PRINT AT 0,0;c$
    90 NEXT a
   100 GO TO 10
```

Akonáhle ste previedli druhý krok, spustite horný program. Riadok 10 sa nás pýta, aký znak chcete zmeniť. Napr bodku, potom stačí len napísať bodku ENTER.

Ak chcete program zastaviť, stlačte len ENTER. Program sa zastaví, pretože ste zadali prázdny reťazec. Riadok 30 obmedzuje znaky, ktoré je možné predefinovať na znaky od SPACE-medzera po (c) (CHR$ 32 až 127).

Premenná c je CODE, čiže poradie v ASCII tabuľke meneného znaku. Slúži k nájdeniu jeho adresy. Slučka na riadku 50 umožňuje zmeniť všetkých 8 bajtov znaku. Riadok 60 od Vás chce udanie hodnoty pre týchto 8 bajtov (nové hodnoty a po jednotlivých bajtoch).

To môže byť prevedené dvoma spôsobmi. Najskôr si ale musíte pripraviť, ako má znak vypadať, podobne ako pri UDG-napísať si na papier bitovú vzorku.

	Hodnota								
Riadok	128	64	32	16	8	4	2	1	
1	0	1	0	1	0	1	0	1	64+16+4+1=85
2	0	0	0	0	0	0	0	0	=0
3	1	0	0	0	0	1	0	0	128+4=132
4	0	0	0	0	0	0	0	1	=1
5	0	0	0	0	0	0	1	1	2+1=3
6	0	0	0	0	0	0	0	0	=0
7	0	1	1	1	0	0	0	0	64+32+16=112
8	1	1	1	1	1	1	1	1	128+64+32+16+8 +4+2+1=255

Tak potom môžete pre riadok 1 napísať buď BIN 01010101 alebo 85, pokiaľ ste si dali tu prácu a dekadickú hodnotu ste vypočítali. Riadok 20 vypočítava, kde sa bude robiť (adresu) POKE nového znaku. Riadok 80 tento nový znak vytlačí v ľavom hornom rohu. Riadok 100 vracia program na riadok 10, aby mohol byť zadaný ďalší znak, ak je treba. Riadok 5 Vám umožňuje mať výpis v novodefinovaných znakoch.. Môžete tiež do riadku 10 pridať INPUT LINE c$, pokiaľ potrebujete zmeniť aj úvodzovky.(").

B L O K O V Á G R A F I K A

sú to "štvorčekové" grafické symboly CHR$ 128 až 143, ktoré vypadajú takto: (a viď Manuál Spectrum, Appendix A).

```
..    .X    X.    XX     pod bodkou si predstavte biely štvorec
..    ..    ..    ..     a pod X čierny štvorec
```

Na nich a okolo nich nie je nič zaujímavého, snáď len to, že je dobré vedieť, aký je vzťah medzi CODE týchto znakov a tým, ktorý štovrček je vyfarbený. CODE sa dá vypočítať pomocou tohoto programu:

2	1
8	4

Hodnota vo vyfarbenom štvorci sa pričíta ku 128 a výsledok je CODE príslušného znaku, napr.

```
.X          To by bol CHR$ (128+1+8),
X.          čo je CHR$ 137
```

Iný pohľad nám umožní, ak berieme do úvahy jednotlivé bity CODE príslušného znaku. Bity 0 až 3 z CODE majú tento význam:

- ak je bit 0 nastavený na 1, je vyfarbený pravý horný štvorček,
- ak je bit 1 nastavený na 1, je vyfarbený ľavý horný štvorček,
- ak je bit 2 nastavený na 1, je vyfarbený pravý dolný štvorček,
- ak je bit 3 nastavený na 1, je vyfarbený ľavý dolný štvorček.

bit 1	bit o
bit 3	bit 2

K N I Ž N I C A P O D P R O G R A M O V

Táto sekcia ponúka kolekciu podprogramov, ktoré môžete individuálne skladovať na páske a potom pomocou MERGE nahrať do iných programov. Zaiste môžu byť použité i ako základ rutín, ušitých presne pre Váš program.

Všetky majú rozdielne čísla riadkov, takže ich môžete spojiť príkazom MERGE ľubovoľný počet, bez toho aby sa navzájom prepísali. Číslovanie riadkov začína od 9000. Niektoré podporogramy majú rovnaké názvy premenných, pokiaľ prevádzajú rovnaké funkcie. Popis podprogramov obsahuje i popis použitých premenných a čísla riadkov, takže sú ľahko upraviteľné.

1. Výmaz časti obrazovky

riadky: 9000 až 9045
premenné: lines, f

Podprogram vymazáva zadaný počet riadkov od spodu obrazovky. PRINT pozícia sa posunie do ľavého horného rohu vymazaného priestoru a nastaví PLOT pozíciu na 0,0 (teda vľavo dole), ako by to bolo po CLS).

Budete opýtaný na zadanie čísla 0 až 22, aby podprogram vedel, koľko riadkov má riadok odspodu vymazať. Môžete tento riadok tiež jednoducho vynechať a zadať premennú LINES ešte pred volaním celého podprogramu. Ak budete meniť správu v riadku 9005, potom musí byť dostatočne krátka aby nedošlo ku scroll niečoho ďalšieho na obrazovke.

Riadok 9025 vytlačí reťazec 32 medzier na každý vymazaný riadok pomocou OVER 0, ktorý má miestny účinok (OVER 1 má účinok globálny a nič by sa nevymazalo). V podprograme nie je špecifikovaná žiadna farba, ale môžete použiť PAPER 8, BRIGHT 8, FLASH 8, tým sa vymaže text a grafika, ale pozadie zostáva nezmenené.

Riadok 9035 posúva PRINT pozíciu vľavo nahor do vymazanej časti. Riadok 9040 posúva PLOT pozíciu na 0,0 (vľavo dole) pomocou POKE do systémových premenných, ktoré túto pozíciu uchovávajú.

```
9000 REM Zmaž časť obrazovky
9005 INPUT "Koľko riadkov ?";lines
9010 IF lines<0 OR lines>22 OR lines<>INT lines THEN
     GO TO 9005
9015 IF lines=0 THEN RETURN
9020 FOR f=21 TO 22-lines STEP-1
9025 PRINT AT f,0; OVER 0;""
9030 NEXT f
9035 PRINT AT f+1,0;
9040 POKE 23677,0: POKE 23678,0
9045 RETURN
```

2. Odpoveď Áno alebo Nie

riadky: 9050-9080
premenné r$

Tento program dovoľuje zodpovedať na otázky, ktoré vyžadujú odpoveď typu Ano alebo Nie.

Každé slovo začínajúce a alebo A sa zmení na A, každé začínajúce na n alebo N sa zmení na N (môže však byť prijatá samozrejme len jedna alternatíva). Buď Ano alebo Nie musí byť zadané, inak Vás program nepustí ďalej.

Po návrate z podprogramu bude r$ obsahovať A alebo N. Malé písmená budú zmenené na veľké.

Riadok 9055 fixuje dĺžku r$ na 1 znak. To spôsobí, že ani stlačenie samotného ENTER neumožní splniť otázku.

Ak je napísané slovo s viacerými znakmi, je prijatý iba prvý, ostatné sú ignorované. Tým je zabezpečené, že riadok 9075 rozhoduje iba o tom, či bolo napísané A alebo N.

```
9050 REM Áno, či nie?
```



```
9055 DIM r$(1)
9060 INPUT "Ano alebo Nie ?";r$
9065 IF r$="a" THEN LET r$="A"
9070 IF r$="n" THEN LET r$="N"
9075 IF r$ = "A" OR r$ = "N" THEN RETURN
9080 GO TO 9060
```

3. Zapnutie CAPS LOCK

riadky: 9085-9100
premenné: zzz

Tento program jednoducho zapína CAPS LOCK automaticky, a to pomocou šestbajtového programu v strojovom kóde, ktorý môže byť vložený z klávesnice a nepotrebuje ani ďalší zvláštny vkladací program.

Pri snímaní, ktorá klávesa bola stlačená, je niekedy nutné kontrolovať len malé alebo veľké písmená, i keď by sme mali mať prehľad o obidvoch ! Je totiž problém, že si nemôžeme byť nikdy istý, že používateľ programu zvolí správny typ písmen (teda malé, keď majú byť malé a naopak).

Všetko, čo tento podprogram robí, je, že mení na 1 bit 3 v systémovej premennej 20568, čím zapína CAPS LOCK (teda všetko ako veľké písmená).

Keď potom Váš vlastný program prehliada klávesnicu, aby zistil, ktorá klávesa bola stlačená, nepotrebujeme už kontrolovať obe možnosti (a to ani v prípadoch, keď CAPS LOCK nejde urobiť ručne, teda pri INKEY$;IN a pod.).

Strojový kód je obsiahnutý v príkaze REM na riadku 9095 (a nemôže to byť čokoľvek, čo by úbohý počítač len poplietlo, ale iba šesť uvedených znakov):

```
!-výkričník
j-malé j
\-znak 92-EXTENDED MODE a SHIFT D (nie je to lomítko)
THEN-kľúčové slovo THEN
OVER-kľúčové slov OVER
<>-symbol nie je rovno (SYMBOL SHIFT, W)
```

Strojový kód je volaný pomocou systémovej premennej NXTLIN, ktorá nám pômože nájsť, kde tento podprogram leží v pamäti a kde začína strojový kód. NXTLIN priebežne udáva adresu začiatku programového riadku, ktorý nasleduje za práve vykonávaným riadkom . Nasledovný riadok pozostáva z dvoch bajtov pre číslo riadku, dvoch bajtov pre ukazateľ dĺžky riadku a jedného bytu pre znak REM atď.

To je vysvetlenie, prečo sa v riadku 9090 pripočítava 5. Premenná zzz nie je dôležitá (jej názov), pretože iba uchováva číslo, ktoré sa vrátilo po prejdení rutiny v strojovom kóde. Môže to byť ktorákoľvek premenná, nesmie však byť inde použitá.

```
9085 REM Zapnutie CAPS LOCK
9090 LET zzz=USR (PEEK 23637+256*PEEK 23638+5)
9095 REM !j\ THEN OVER<>
9100 RETURN
```

4. Zrušenie CAPS LOCK

riadky: 9105-9120
premenná: zzz

Tento podprogram je doplnkom k predchádzajúcemu. Ruší CAPS LOCK, takže môžu
byť používané a detektované i malé písmená. Toto môžete jednoducho obísť pomocou INKEY$ a CAPS SHIFT, ale podprogram je trošku užitočnejší. Okrem strojového kódu a čísel riadkov je podprogram podobný predchádzajúcemu. Šesť znakov po REM na riadku 9115 je:

!-výkričník
j-malé j
\-znak 92-(EXTENDED MODE, SHIFT D-nie lomítko)
THEN-kľúčové slovo THEN
O-grafické O, tj. znak CHR $ 158
<>-nerovná sa, SYMBOL SHIFT, W

```
9105 REM Zrušenie CAPS LOCK
9110 LET zzz=USR (PPEK 23637+256*PEEK 23638+5)
9115 REM !j\ THEN O<>
9120 RETURN
```

5. Stlač ľubovoľnú klávesu, aby program pokračoval.

riadky: 9125-9140
premenné: žiadne

Často je nevyhnutné v programe čakať, pokiaľ nepríde nejaký signál od operátora, napr. že prečítal inštrukciu. Väčšina programátorov vkladá podprogram " Pres any key to continue" (stlač akúkoľvek klávesu,aby program pokračoval), ale keď stlačíme niektorú z kláves SHIFT, tak sa nič nedeje.

Tento podprogram robí to, čo bolo uvedené a naviac pomocou IN kontroluje i klávesy SHIFT. Program môže zhavarovať pri súčasnom stlačení dvoch alebo viacerých kláves na horných troch riadkoch, ale to je možné úspešne previesť len veľmi ťažko.

```
9125 REM Press any key
9130 PRINT "Stlač klávesu ..."
9135 IF INKEY$ ="" AND IN 65278=255 AND IN 32766=255 THEN
     GO TO 9135
9140 RETURN
```

6. Využitie SCREEN$ k detekcii používateľom definovanej grafiky (UDG na obrazovke)

riadky: 9145-9195
premenné: x, y, a, b, a$

Nedostatkom SCREEN$ je to, že dokáže detekovať na obrazovke len znaky, ktorých CODE je medzi 32 a 127. SCREEN$ pracuje tak, že si zo

systémových premenných vytiahne počiatočnu adresu sady znakov a tú potom prehliada, až nájde rovnaký znak ako je na obrazovke.

Aby SCREEN$ detekovala i užívateľom definovanu grafiku UDG, musíme počítaču oznámiť, že sada znakov v generatore ROM (ktoru vie prehliadať) začína tam, kde je užívateľom definovaná grafika. SCREEN$ potom bude prehliadať túto grafiku. Mala matematická manipulácia potom SCREEN$ prinúti, aby vydalo správne CODE.

Pri používaní tohto podprogramu je nutné zadať dve hodnoty premenných y a x. Tie slúžia pre SCREEN$ (y,x). y je súradnica osi Y obrazovky a x je súradnica osi X obrazovky. Udávajú miesto, ktore ma SCREEN$ testovať.

Pôvodné hodnoty systémových premenných sú uložené do a a b, takže po prebehnutí programu sa všetko vráti do pôvodného stavu. Je to preto, že môžeme používať niekoľko generátorov znakov (nielen ten jeden v ROM) a tento podprogram sa potom vráti k tomu, ktorý bol práve využívaný.

Ak používate niekoľko sád UDG, potom tento program bude prehliadať len tú sadu, ktorá je práve v prevádzke. Podporogram kontroluje najskôr normálne znaky a ak neuspel, potom UDG. Pokiaľ táto činnosť povedie k identifikácii znaku, bude CODE tohto znaku zmenené, aby sa zabránilo spleteniu UDG A napr. s medzerou. Je to prevedené v riadku 9180.

```
9145 REM SCREEN$
9150 LET a$=SCREEN$ (Y,X)
9155 IF a$<> "" THEN RETURN
9160 LET a=PEEK 23606: LET a=PEEK 23607
9165 POKE 23606,PEEK 23675
9170 POKE 23607, PEEK 23676-1
9175 LET a$=SCREEN$ (Y,X)
9180 IF a$<>"" THEN LET a$=CHR$ (CODE a$+112)
9185 POKE 23606,a
9190 POKE 23607,b
9195 RETURN
```

7. Hľadanie kópie nejakého reťazca v inom reťazci

riadky: 9200-9235
premenné: p, b$, c$

Tento podprogram umožňuje vyhľadať nejaký reťazec v inom reťazci. To môže byť užitočné, povedzme v hre, kde hľadáte a určujete určité zadané slová a pod. Pokiaľ sa c$ nájde vo vnútri b$, potom po skončení podprogramu obsahuje premenná p číslo prvku, na ktorom hľadaný reťazec začína. Napr. pri hľadaní reťazca TO v reťazci MOTOTECHNA bude p obsahovať 3, pretože TO začína na tretej pozícii. Ak nie je kópia nájdená, potom p obsahuje 0. b$ je hlavný reťazec, ktorý prehliadame, či neobsahuje reťazec c$. Dĺžka b$ môže byť cez celú pamäť. Je ale zrejmé, že prehliadanie dlhých reťazcov tiež dlho trvá.

```
9200 REM Hľadaj reťazec v reťazci
9205 LET p=0
9210 IF LEN c$=0 OR LEN b$=0 OR LEN c$>LEN b$ THEN RETURN
9215 FOR p=1 TO LEN b$-LEN c$+1
9220 IF b$ (p TO p+LEN c$-1)=c$ THEN RETURN
9225 NEXT p
9230 LET p=0
```

```
9235 RETURN
```

8. Prevod dekadických čísel na dvojkové reťazce

riadky: 9240-9280
premenné: dec, d$, j

Pri práci s pamäťovými miestami je občas potrebné testovať stav jednotlivých bitov alebo vzoriek núl a jedničiek pri dvojkovom čísle. Okrem toho môže byť tento podprogram využitý v niektorých matematických aplikáciach, kde je treba prevádzať dekadické čísla na dvojkové a naopak. Ak máte číslo j, potom tento podprogram ho prevedie na osem alebo šestnásťmiestny reťazec d$ tak, že si môžete predstaviť každú číslicu ako bajt. Tak čísla 0 až 255 budú prevedené na osemmiestny reťazec a čísla 256 až 65535 na šestnásťmiestny. Ak nechcete túto schopnosť "automatickej dĺžky", potom proste vymažete riadky 9270 a 9275. Pred volaním tohoto podprogramu definujte číslo j, ktoré si prajete previesť na dvojkové (napr. LET j=255) a potom zadajte GOSUB 9240. j sa zakopíruje do premennej dec, aby jeho hodnota mohla byť v priebehu programu menená tak, ako je treba a pritom sa j zachovalo v pôvodnom stave. Opakované delenie dvoma premieňa d$ na reťazec ktorý je dvojkovým ekvivalentom j. Ak chcete previesť d$ na desiatkové číslo, použite VAL ("BIN"+D$).

```
9240 REM Prevod z dekadickej do binárnej sústavy
9245 LET dec=j
9250 LET d$=""
9255 LET d$=STR$ (dec-INT (dec/2)*2)+d$
9260 LET dec=INT (dec/2)
9265 IF dec<>0 THEN GO TO 9255
9270 IF LEN d$<8 THEN LET d$="00000000"(TO 8-LEN d$)+d$
9275 IF LEN d$<16 AND LEN d$>8 THEN LET d$="00000000" (TO 16-LEN d$)+d$
9280 RETURN
```

9. Prevod dvojkového reťazca na dekadické číslo

riadky: 9285-9315
premenné: dec, sum, e, e$

Pred použitím tohto podprogramu je treba zadať reťazec E$ ako rad dvojkových čísel, kde 1 predstavuje dvojkovú 1 a 0 dvojkovú 0. Napr. LET e$="1101". GOSUB 9285 potom prevedie do premennej dec dekadický ekvivalent e$. Určite by to bolo možné urobiť aj pomocou BIN, ale niekedy je nutné uložiť informáciu ako reťazec, aby bolo možné testovať i jednotlivé bity.

Tento podprogram Vám umožňuje pracovať s dvojkovými i dekadickými číslami a prevádzať ich z jedného tvaru do druhého tak, ako práve potrebujete. Je to možné urobiť i pomocou BIN (previesť dvojkový reťazec na dekadické číslo, ale veľkou okľukou pomocou VAL);tak je to vidieť v druhej alternatíve, ale to je obmedzené funkciami VAL a BIN.

```
9285 REM Prevod z dvojkovej sústavy do dekadickej
9290 LET dec=0: LET sum=1
9295 FOR e=LEN e$ TO 1 STEP-1
9300 IF e$(e)="1" THEN LET dec=dec+sum
```



```
9305 IF e$(e)="0" OR e$(e)="1" THEN LET sum=sum*2
9310 NEXT e
9315 RETURN

9285 REM Prevod z dvojkovej sustavy do dekadickej
9290 LET dec=VAL ("BIN"+e$)
9295 RETURN
```

V oboch verziách programu môžete aj pridať do dvojkoveho reťazca medzery, aby bol čitateľnejší. Napr. reťazec "1111111111111111" (16 jednotiek) môžete zadať ako "1111 1111 1111 1111". Prevod to neovplyvni, pretože program vynechá proste všetko, čo nie je 1 alebo 0. Pri použití BIN môžete číslice odeľovať len medzerami, ale akýkoľvek iný oddeľovací znak spôsobí chybu C, Nonsence in Basic.

10. Zaokrúhľovanie na dve desatinné miesta

riadky: 9230-9345
premenné: f$, value, lf

Tento podprogram zaokrúhľuje hodnotu premennej value na dve desatinné miesta a pridáva nulu pred desatinnú bodku tam, kde je to treba. Je to veľmi potrebné pri práci s čiastkami peňazí. Hodnota value sa vracia ako reťazec f$ pripravený pre tlač alebo dekódovane pomocou VAL, ak sa má s ňou zachádzať ako s číslom. Napr. hodnota value 0.687 sa vráti v f$ ako 0.68. Do tlače potom stačí len pridať napr. označenie meny ($, Kčs apod.). Pred spustením programu je treba zadať hodnotu (buď LET value= alebo pomocou INPUT). Pretože Spectrum udržuje v pamäti dekadické čísla, môže nastať pri zaokrúhľovaní drobná nepresnosť. Tak napr. číslo 0.005 bude zaokrúhlené na 0.00 namiesto 0.01. Na to je jednoduchá rada: pripočitavať v riadku 9325 trošku viac, napr. +0.500001 namiesto 0.5.

```
9320 REM 2 desatinné miesta
9325 LET f$=STR$  (INT (value*100+.5)/100)
9330 IF f$(1)="." THEN LET f$="0"+f$
9335 LET lf=LEN f$-LEN STR$ INT VAL f$
9340 LET f$=f$+(".00" AND lf=0)+("0" AND lf=2)
9345 RETURN
```

11. Klávesnica ako pomocník pri grafických hrách

riadky: 9350-9380
premenné: x, y

Popísaný podprogram ukazuje, ako je možné využiť celé sekcie klávesnice pri ovládaní pohybu na obrazovke. Už nebude napr. potreba pracného hľadania kláves 5 a 8.

V prvom programe pohybuje ľavá polovica klávesnice vpred vľavo a pravá polovica opačne. Pri možnosti voľby z 20 kláves by ste už nemali mať ťažkosti s nájdením tej pravej. Jedine, načo sa musí dávať pozor, je súčasné stlačenie CAPS SHIFT a SPACE, čo spôsobí BREAK a zastavenie programu.

Ovládanie prebieha tak, že sa mení hodnota premennej x, ktorá tak

môže byť využitá pre PRINT alebo ako PLOT súradnica nejakého objektu na obrazovke. Je na Vás, aku určíte minimálnu a maximálnu hodnotu, ktorú môže x dosiahnuť.

```
9350 REM Pohyb vľavo, alebo vpravo
9355 LET x=x+(IN 61438+IN 57342+IN 49150+IN 32766<>1020)-
     (IN 63486: IN 64510+IN 65022+IN 65278<>1020)
9360 RETURN
```

Druhý program zabezpečuje pohyb v 8 smeroch, pričom sa klávesnica rozdelí na 4 časti-horná rada hore, ľavá polovica prostredných dvoch radov vľavo, ich pravá polovica vpravo a spodný rad dole. Stlačenie kombinácie kláves spôsobí pohyb diagonálne. X je súradnica PRINT pozície naprieč obrazovkou a y je súradnica PRINT pozícia dole obrazovkou. Samozrejme, že si opäť môžeme zadať hranice, medzi ktorými sa môžu obe súradnice pohybovať.

```
9365 REM Štvorsmerný pohyb
9370 LET x=x+(IN 49150+IN 57342<>510)-(IN 64510+IN 65022<>510)
9375 LET y=y+(IN 65278+IN 32766<>510)-(IN 65278+IN 61438<> 510)
9380 RETURN
```

12. Triedenie reťazcov

riadky: 9385-9415
premenné: s, t, s$, u$, used

Tento podprogram Vám umožní triediť reťazce v reťazcovom poli v abecednom poradí, pričom ale musíte vedieť, koľko reťazcov sa má triediť. Used je premenná, ktorá udáva práve počet triedených reťazcov. Tak napr. pokiaľ ste skôr deklaroval DIM s$ (20,20) a máte zapísaných 15 reťazcov, potom musíte do used vložiť 15, pretože pre triedenie ostávajúcich piatich reťazcov nie je záchytný bod. Ostatné premenné sú riadiace premenné dvoch slučiek a jedna pomocná reťazcová, slúžiaca pre prehodenie poradia dvoch susedných reťazcov.

```
9385 REM Triedenie reťazcov
9390 FOR s=1 TO used-1
9395 FOR t=s TO used
9400 IF s$(t)<s$(s) THEN LET u$=s$(s): LET s$(s)=s$(t): LET s$(t)=u$
9405 NEXT t
9401 NEXT s
9415 RETURN
```

13. Výpis reťazcových polí bez uzavierajúcich medzier

riadky: 9420-9440
premenné: u,w,s$

Podprogram umožňuje výpis reťazcov alebo reťazcových polí bez problému, čo so záverečnými medzerami, ktoré spôsobujú nerovnaké vzdialenosti dvoch slov. Pre využitie programu je treba nahradiť reťazcové pole menom toho poľa, ktoré chcete vypísať. W ukazuje, ktorý reťazec z

celého poľa sa má vypísať. Tak, ak máte napr. DIM s$ (20,20) a chcete vypísať s$(8), potom w musí byť 8 (LET w=8), este pred tým, než budete podprogram volať.

```
9420 REM Výpis bez medzier poľa
9425 FOR u=LEN s$(w) TO 1 STEP-1
9430 IF s$(w,u)<>" " THEN PRINT s$ (w, TO u): RETURN
9435 NEXT u
9440 RETURN
```

14. Výroba zvuku

riadky: 9445-9465
premenná: z

Program vyrába zvláštny zvuk s klesajúcou výškou, ktorý je vhodný pri rôznych príležitostiach, napr. zostrelení votrelca.

```
9445 REM Vytvor zvuk
9450 FOR z=0 TO 20 STEP 2
9445 BEEP .01,69-(z* 2.5)
9460 NEXT z
9465 RETURN
```

15. Prevod dekadických čísel na hexadecimálne (šesnástkové)

riadky: 9470-9505
premenné: dec, number, h$, n1

Občas máte dekadické číslo, ktoré potrebujete previesť do hexadecimálneho tvaru pre programovanie v strojovom kóde apod.

Tento podprogram prevádza hodnotu v dec na hexadecimálny reťazec v h$ tak, že dekadická 16 bude hexadecimálna 10. Pokiaľ bude dec kladné, neexistuje obmedzenie pre jeho rozsah (vyjmúc obmedzenia dané aritmetikou Spectra). Vzhľadom na to, že dve hexadecimálne číslice normálne odpovedajú jednému bytu pamäte, predáva tento program na začiatok hex číslo 0 (nula), aby posledný hex reťazec h$ mohol byť prípadne zavedený zavádzacím programom. Táto funkcia je na riadku 9500 a v prípade, že ju nepožadujete, môžete ju vynechať. Ostatné premenné sú použité ako pomocné pre proces konverzie.

```
9470 REM Prevod z desiatkovej sústavy do šestnástkovej
9475 LET dec=number
9480 LET h$=""
9485 LET n1=INT (dec-(INT (dec/16)*16))
9490 LET h$=CHR$ (n1+48+(7 AND n1>9))+h$
9495 IF INT (dec/16)<> 0 THEN LET dec=INT (dec/16): GO TO 9485
9500 IF INT (LEN h$/2)*2<>LEN h$ THEN LET h$="0" +h$
9505 RETURN
```

16. Prevod hexadecimálnych čísel na dekadické

riadky: 9510-9540
premenné: h$, h, value, dec

Program zaisťuje opačnú funkciu, menovite prevádza hex reťazec h$ na dekadické číslo dec.
H$ môže byť tak dlhý, pokiaľ stačí aritmetická kapacita Spectra pre výsledné dekadické číslo. H$ musí byť určené ešte pred volaním podprogramu, po jeho priebehu je dekadický výsledok v dec.

```
9510 REM Prevod z hexadecimálnej sústavy do desiatkovej
9515 LET value=1: LET dec=0
9520 FOR h=LEN h$ TO 1 STEP-1
9525 LET dec=dec+(CODE h$h)-48-(7 AND h$ (h)>"9")-(32 AND h$ (h)>="a"))
     *value
9530 LET value=value*16
9535 NEXT h
9540 RETURN
```

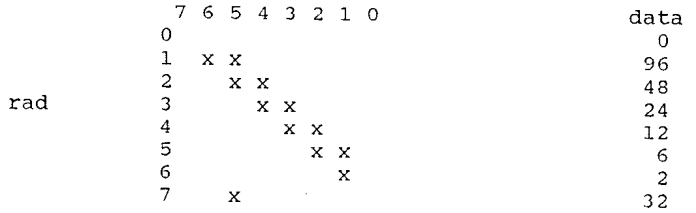
Hex reťazec sa musí skladať z číslic 0 až 9 vrátane a veľkých písmen A až F vrátane alebo malých písmen a až f vrátane.

D I Z A J N É R U Ž Í V A T E Ľ S K E J G R A F I K Y

Tento program vám umožní navrhovať vlastnú grafiku a potom vám oznámi hodnotu (dekadickú), ktorú použijete ako data pri zavádzaní takto vytvorených znakov do pamäte. Nepredstavuje žiadny luxus, ako iné podobné programy, ale pre splnenie svojho účelu je postačujúci. Všetky príkazy a doplnky sú po celú dobu tvorby na obrazovke.
Pokiaľ vlastníte tlačiareň, tak máte i možnosť vidieť, ako vyzerá znak v skutočnej veľkosti. Tu je príklad, ako vypadá obrazovka pri tvorbe, aby ste si mohli urobiť predstavu.

Ovládanie

0 medzera v bode-farba PAPER
1 vyfarbený bod-farba INK
2 vyčisti displej
3 zastavenie programu
4 opis na tlačiareň
5 vľavo
6 dole
7 nahor
8 vpravo



Veľký štvorec je zobrazovací priestor, kde vidíte znak, ktorý tvoríte. Ukazovateľ (+) ukazuje na miesto, ktoré sa má určiť, či bude mať farbu atramentu alebo papiera. S týmto ukazovateľom môžete pohybovať pomocou kláves 5,6,7 a 8.
Stlačenie 0 spôsobí, že miesto, kde je ukazovateľ, zostane nevyfarbené (nezávisle na tom, aké bolo toto miesto predtým: biele alebo čierne). Naopak, stlačenie 1 spôsobí, že toto miesto bude čierne. Poznamenávam, že ukazovateľ sa na čiernom pozadí objaví biely, takže je stále vidieť. Bokom vľavo od tohoto veľkého štvorca sa objavia 4 kópie práve tvoreného znaku, aby ste mohli vidieť, ako bude vyzerať v skutočnej veľkosti.
Stlačenie klávesy 2 vymaže zobrazený užívateľský znak, nastaví ukazovateľ do ľavého horného roku a vynuluje ukazatele Data.
Klávesa 3 program zastaví, pokiaľ ste so svojou prácou skončili. Pokiaľ chcete program reštartovať (napr. pri chybe), stačí dať len CONTINUE.
Klávesa 4 vám vypíše obrazovku na papier, avšak musí byť pripojená tlačiareň. Keď máte návrh znaku ukončený, opíšte si dáta, aby ste ich mohli použiť ako DATA príkaz pri zápise UDG do pamäte. Dáta sú dekadické, nie je teda treba používať BIN.

Výpis programu:

```
 1 REM UDG
10 GO SUB 9000
20 IF INKEY$<>"" THEN GO TO 20
30 LET i$=INKEY$
40 IF i$<"0" OR i$>"8" THEN GO TO 30
50 PRINT AT 2+VAL i$,1; INVERSE 1;">"
60 IF i$="0" OR i$="1" THEN GO SUB 500
70 IF i$="2" THEN GO SUB 1000
80 IF i$="3" THEN STOP
90 IF i$="4" THEN GO SUB 1500
100 IF i$>= "5" AND i$<="8" THEN GO SUB 2000
110 PRINT AT 2+VAL i$,1;" "
120 GO TO 20
500 REM INK or blank out
510 LET d$(cy,cx)=i$
520 LET d$(cy)=VAL ("BIN "+d$(cy))
530 POKE USR "A" -1+cy,d (cy)
540 PRINT AT 12+cy,16+cx; INVERSE VAL d$(cy,cx);"+";INVERSE 0;AT 16,1;
    "A A A A";AT 12+cy,27;d(cy);"   " (TO 3-LEN STR$ d (cy))
550 RETURN
1000 REM Clear the display
1010 FOR a=0 TO 7
1020 POKE USR "A"+a,0
1030 LET d$ (a+1)="00000000"
1040 LET d (a+1)=0
1050 PRINT AT 13+a,17;" 8 medzier ";AT 16,1;"A A A A";
     AT 13+a,27;"0    "
1060 NEXT a
1070 LET cy=1: LET cx=1
1080 PRINT AT 12+cy,16+cx,"+"
1090 RETURN
1500 REM Copy to printer
```

```
1510 PRINT AT 12+cy,16+cx; OVER 1;"+"
1520 COPY
1530 PRINT AT 12+cy, 16+cx; OVER 1;"+"
1540 RETURN
2000 REM Move cursor
2010 LET oldcx=cx: LET oldcy=cy
2020 LET cx=cx-(i$="5" AND cx>1)+(i$="8" AND cx<8)
2030 LET cy=cy-(i$="7" AND cy>1)+(i$="6" AND cy<8)
2040 IF oldcx<> cx OR oldcy<> cy THEN PRINT AT 12+oldcy,
     16+oldcx; INVERSE VAL d$(oldcy,oldcx);" ";AT 12+cy,
     16+cx; INVERSE VAL d$(cy,cx);"+"
2050 RETURN
9000 REM Initialise
9010 BRIGHT 0: FLASH 0: INVERSE 0: OVER 0
9020 INK 0: BORDER 7: PAPER 7: CLS
9030 FOR a=0 TO 7: POKE USR "A"+a,0: NEXT a
9040 PRINT "Ovládanie" ''
9050 PLOT 0,167:DRAW 63,0
9060 PRINT "0 medzera v bode-farba PAPER "
9070 PRINT "1 vyfarbený bod-farba INK"
9080 PRINT "2 vyčisti displej"
9090 PRINT "3 zastavenie programu"
9100 PRINT "4 opis na tlačiareň"
9110 PRINT "5 vľavo"
9120 PRINT "6 dole"
9130 PRINT "7 hore" ;TAB 17;"Bit": PLOT 152,95: DRAW 23,0
9140 PRINT "8 vpravo"
9150 PRINT TAB 17;"76543210  DATA" '': PLOT 216,79: DRAW 31,0
9160 FOR a=0 TO 7: PRINT TAB 15;a;TAB 27;"0": NEXT a
9170 PLOT 135,72: DRAW 65,0: DRAW 0, -65: DRAW -65,0: DRAW 0,64
9180 PRINT AT 17,11;"Rad": PLOT 88,31: DRAW 23,0
9190 PRINT AT 13,17;"+"
9200 DIM d(8): DIM d$(8,8)
9210 FOR a=1 TO 8: LET d$(a)="00000000": NEXT a
9220 LET cy=1: LET cx=1
9230 RETURN
```

Pri zápise programu do počítača dbajte, prosím, na tieto upozornenia:

riadok 520: slovo "BIN" je funkcia, a nie len tri písmena B I N
riadok 530: "A" po USR je grafické A
riadok 1050: tu je "8 medzier";"AAAA" sú grafické A a "0 ", teda 0 a 4 medzery
riadok 2040: jedna medzera v úvodzovkách
riadok 9150: "76543210 DATA"'' teda 2 medzery medzi číslami a DATA a po úvodzovkách 2 apostrofy

Program je navrhnutý ako séria podprogramov, volaných od počiatočnej slučky. Prvá prevedená vec je spustenie podprogramu na riadku 9000. Ten pozostáva z nastavenia trvalých atribútov (papier, atrament atď.) a vyvolania uživateľského A a vymazanie jeho atramentových bodov, aby ste na počiatku objavili čistý veľký štvorec. To je prevedené štandardným postupom POKE USR osemkrát v slučke.

Riadky 9040 až 9140 tlačia inštrukcie. Príkazy DRAW slúžia pre podtrhnutie. Riadky 9150 až 9180 vytvárajú štvorec, v ktorom budete

zostavovať váš znak. Čísla nad ním udávajú, ktorý bit z bajtu DATA odpovedá tomu ktorému bodu. Čísla na boku štvorca vľavo udávaju, ktorý bajt sa bude v patričnom okamihu zapisovať, napr. rad 0 ide od USR "A"+0 atď.

Riadok 9190 tlačí ukazovateľ do jeho východzej pozície. Riadok 9200 nastavuje numerické pole d a reťazcové pole d$. D uchováva hodnoty DATA a d$ obsahuje príslušnú binárnu verziu, aby bolo možné zistiť, ktorý jednotlivý bod má farbu atramentu a ktorý nie. Ak chcete mať výpis vo dvojkovom tvare pre použitie funkcie BIN pri ukladaní daného znaku do pamäte, potom musí byť tlačené pole d$ a nie d.

Riadok 9210 píše 0 do všetkých d$(), aby bol na začiatku štvorca prázdny. Premenné na riadku 9200 sú cy-udáva súradnice y ukazovateľa (ale nie od hornej hrany obrazovky) a cx-udáva x súradnicu ukazovateľa (ale tiež nie od ľavého okraja obrazovky). Po návrate z tohoto podprogramu prevedie program hlavnú slučku > ako ukazovateľ na možnosti voľby funkcii. Riadky 60 až 100 vyberajú zvolený podprogram činnosti.

Podprogram na riadku 500 je volaný, ak stlačíte klávesu 0 alebo 1. Odpovedajúci reťazec poľa d$() je nastavený na 0, ak má byť miesto pod ukazovátkom čisté, alebo na 1, ak má tam byť farba atramentu. Ten je potom na riadku 500 prevedený do dekadického tvaru použitím VAL a reťazca obsahujúceho BIN a dvojkový reťazec d$() - výsledok je uložený v odpovedajúcom prvku poľa d() ako Data. D() je menený zakaždým, keď niečo menite na UDG. Táto hodnota je potom zapísaná príkazom POKE do priestoru pre UDG v pamäti, takže grafické A potom vyzerá ako znak, ktorý prešiel vašou úpravou. Riadok 540 tlačí ukazovátko + do štvorca na obrazovke a určuje, či bude normálny alebo invertovaný, v závislosti na i$ - , čo zodpovedá tomu, ktorú voľbu ste stlačili na klávesnici. Posledný PRINT príkaz na riadku 540 určuje, koľko medzier sa bude tlačiť do stĺpca DATA v závislosti na tom, ak je k tlači 1 alebo 2 alebo 3 znaky (medzeru preto, aby sa vymazali číslice, ktoré tam boli predtým). Riadok 540 vytlačí tiež vľavo od štvorca 4 grafické A, teda nami práve upravené znaky.

Ďalší podprogram na riadku 1000 maže to, čo bolo vo štvorci, aby ste mohli tvoriť ďalší znak.

Podprogram na riadku 1500 kopíruje obrazovku na tlačiareň. Tu je vidieť, ako je užitočné použitie PRINT/OVER 1.

Ďalší podprogram na 2000 zabezpečuje pohyb ukazovateľa. Všimnite si na riadku 2040, ako je dvojkový reťazec d$() spolu s VAL využitý k zabezpečeniu, či je alebo nie je niečo k tlači inverzne. Inde neužitočné, ale tu ideálne.

Poznamenávame, že v tomto programe nie je funkcia opakovania pri trvalom stlačení klávesy. Ak sa chcete dostať s ukazovateľom od jedného okraja štvorca ku druhému, potom musíte stlačiť patričnú klávesu 8x. To zabezpečuje riadok 20, ktorý spôsobí, že program čaká, pokiaľ stlačenú klávesu nepustíte. Pokiaľ sa Vám to nezdá, skúste túto funkciu vynechať. Zistíte, že ukazovátko sa pohybuje tak rýchlo, že je obtiažné ho zastaviť tam kde chcete. Ak sa Vám to bude páčiť, môžete skúsiť dať na riadok 20 konštantné oneskorenie, napr. 20 FOR a=1 TO 50: NEXT a, čo vám dá krásne malé opakovanie. Neskúšajte použiť PAUSE, pretože tlač akejkoľvek klávesy vám PAUSE usekne a vy potom môžete ukazovateľ naháňať niekde po obrazovke.

H O T O V É N Á V R H Y U D G

Tu máte navrhnutých niekoľko grafických symbolov, ktoré sú vhodné pre vaše grafické hry. Ušetríte si tak čas s ich návrhom. Pozostávajú zo

spoločného programu, ktorý zapíše príkazom POKE príslušné dáta v príkazoch DATA na požadované miesto v oblasti pamäte určené pre UDG. Uvedené čísla riadkov u DATA sú náhodné, je potrebne, aby ste si ich upravili podľa vašej potreby.

Napr. vytvorenie znaku votrelca:

```
8010 REM Zmena grafického A na votrelca
8020 RESTORE
8030 FOR a=USR "A" TO USR "A"+7
8040 READ udg
8050 POKE a, udg
8060 NEXT a
8070 REM Čísla riadkov DATA sú vzaté
8080 REM ako príklad, prečíslujte si ich
8090 REM tak, aby vyhovovaly vašemu programu
8100 DATA 66,60,90,126,60,24,36,66
8110 PRINT "A": REM A v grafickom režime
```

Určite budete chcieť REM vynechať pre zjednodušenie. Všetko čo je k tomu treba - napísať čisto program bez riadkov s REM.

votrelec 1

```
.#....#.  66
..####..  36
.#.##.#.  90
.######.  126
..####..  60
...##...  24
..#..#..  36
.#....#.  66
```

votrelec 2

```
.#....#.  66
..#..#..  60
#.####.#  183
#.#..#.#  165
########  255
..####..  60
..#..#..  36
###..###  231
```

votrelec 3

```
..#####.  62
..#.#.#.  42
..#####.  62
...###..  28
....#...  8
.### ###  119
.#.....#  65
.#.....#  65
```

zlomok 1/4

```
.#...#..  68
.#..#...  72
.#.#..#.  82
..#..##.  38
.#..#.#.  74
#..#..#.  146
...#####  31
......#.  2
```

zlomok 1/2

```
.#...#..  68
.#..#...  72
.#.#....  80
..#.##..  44
.#....#.  66
#....#..  132
....#...  8
....####  15
```

zlomok 3/4

```
##....#.  225
..#...#.  34
.#...#..  68
..#.#.#.  42
####.##.  246
..#.#.#.  42
.#..####  79
#.....#.  130
```

Karty: kára

```
....#...  8
...###..  28
..#####.  62
.#######  127
..#####.  32
...###..  28
....#...  8
........  0
```

piky

```
...#....  16
..###...  56
.#####..  124
#######.  254
#######.  254
.#.#.#..  84
...#....  16
..###...  56
```

srdcia

```
.#...#..  68
###.###.  238
#######.  254
#######.  254
#######.  254
.#####..  124
..###...  56
...#....  16
```

kríže

```
...##...  24
..####..  60
...##...  24
.#####..  90
##=#####  255
.#.##.#.  90
...##...  24
..####..  60
```


Spojky: vpravo

```
..####..  60
.######.  126
######..  252
####....  240
###.....  224
######..  252
.######.  126
..####..  60
```

vľavo

```
..####..  60
######..  252
..######  63
....####  15
....####  15
..######  63
######..  252
..####..  60
```

nahor

```
.#....#.  66
.#....#.  66
###..###  231
###..###  231
########  255
########  255
.######.  126
..####..  60
```

dole

```
..####..  60
.######.  126
########  255
########  255
###..###  231
.#....#.  66
.#....#.  66
```

duch

```
..###...  56
.#####..  124
##.#.##.  214
##.#.##.  214
#######.  254
#######.  254
#.#.#.#.  170
#.#.#.#.  170
```

zdroj energie

```
........  0
...##...  24
..####..  60
.######.  126
.######.  126
..####..  60
...##...  24
........  0
```

značka

```
........  0
........  0
........  0
...##...  24
...##...  24
........  0
........  0
........  0
```

Auto: vpravo

```
........  0
###.###.  238
.#...#..  68
########  255
########  255
.#...#..  68
###.###.  238
........  0
```

dole

```
.#.##.#.  90
.######.  126
.#.##.#.  90
...##...  24
.#.##.#.  90
.######.  126
.#.##.#.  90
...##...  24
```

vľavo

```
........  0
.### ###  119
..#...#.  34
########  255
########  255
..#...#.  34
.###.###  119
........  0
```

nahor

```
...##...  24
.#.##.#.  90
.######.  126
.#.##.#.  90
...##...  24
.#.##.#.  90
.######.  126
.#.##.#.  90
```

Lietadlo: 1

```
...##...  24
...##...  24
..####..  60
.######.  126
########  255
...##...  24
...##...  24
..####..  60
```

2

```
...#....  16
...##...  24
#..###..  156
########  255
########  255
#..###..  156
...##...  24
...#....  16
```

3

```
..####..  60
...##...  24
...##...  24
#########  255
.######.  126
..####..  60
...##...  24
...##...  24
```

4

```
....#...  8
...##...  24
..###..#  57
########  255
########  255
..###..#  57
...##...  24
....#...  8
```

loď

```
....#...  8
...##...  24
..####..  60
....#...  8
########  255
.######.  126
..####..  60
```

človek

```
..###...  56
..###...  56
...#....  16
.#####..  124
...#....  16
..###...  56
.#...#..  68
.#...#..  68
```

výbuch

```
#..#...#  145
.#.#..#.  82
........  0
##......  192
......##  3
........  0
.#..#.#.  74
#...#..#  137
```

Zdroj streľby pre všeobecné použitie v 8-mich smeroch:

```
...##..    24        ......##    3        ##......  192
...##..    24        ....##.#   13        #.##....  176
..#..#.    36        ..##..#.   50        .#..##..   76
..#..#.    36        ##....#.  194        ..#...##   35
.#....#.   66        ..##.#..   52        ..#...##   35
.#.##.#.   90        ...#.#..   20        .#..##..   76
#.#..#.#  165        ....#...    8        #.##....  176
##....##  195        ....#...    8        ##......  192

....#...    8        ##....##  195        ...#....   16
....#...    8        #.#..#.#  165        ...#....   16
...#.#..   20        .#.##.#.   90        ..#.#...   40
..##.#..   52        .#....#.   66        ..#.##..   44
##....#.  192        ..#..#..   36        .#....##   67
..##..#.   50        ..#..#..   36        .#..##..   76
....##.#   13        ...##...   24        #.##....  176
......##    3        ...##...   24        ##......  192
```

```
......##    3          ##......  192
....##.#   13          #.##....  176
..##..#.   50          .#..##..   76
##...#..  196          .#....##   67
##...#..  196          ..#.##..   44
..##..#.   50          ..#.#...   40
....##.#   13          ...#....   16
......##    3          ...#....   16
```

V Y U Ž I T I E F U N K C I Í S C R E E N $ A A T T R

Využívanou činnosťou pri grafických hrách je kontrola prítomnosti daného znaku na určenej pozícii obrazovky, napr. pre detekciu zásahu. BASIC počítača Spectrum má pre tieto účely funkciu SCREEN$, ktorá pracuje takto:

Funkcia SCREEN$ (y,x) vracia ako výsledok svojej činnosti string (reťazec), ktorý zodpovedá znaku na obrazovke v súradniciach y (riadok) a x (stĺpec). Pritom nezáleží na tom, či je zobrazený znak inverzný alebo nie. Y a x môžu byť čísla alebo výrazy, podobne ako je tomu v príkazoch PRINT. Napr. skúste tento program:

```
10 PRINT AT 0,5;"+"
20 PRINT "Znak na pozícii 0,5 je ";SCREEN$ (0,5)
```

Na ďalšom programe vidíte, že INVERSE nemá vplyv na výsledok funckie SCREEN$. Tá vráti napr. + pre inverzné + alebo SPACE pre znak GRAPHICS SHIFT 8.

```
10 PRINT AT 0,5; INVERSE 1;"+"
20 PRINT "Znak na pozícii 0,5 je ";SCREEN$ (0,5)
```

Je zrejmé, že atribúty (farba, blikanie, jas) nemajú vplyv na výsledok, rozhodujúca je iba kombinácia bodov na testovanej pozícii obrazovky.

Malý príklad využitia funkcie SCREEN$ pre grafické hry. Riadite vesmírnu loď (inverzné V) tak, aby sa nezrazila s niektorou hviezdou (hviezdička), ktoré sa pohybujú. Pri kolízii sa hra zastaví a je ukázané vaše skóre. Klávesa 5 pohybuje loďou vľavo a klávesa 8 vpravo. Loď je uprostred obrazovky, pretože hviezdy sa pohybujú nahor okolo Vás.

Riadok 30 určuje, v ktorom mieste vaša loď začína, riadok 40 počíta vaše skóre a riadok 50 spôsobuje scroll, bez toho aby sa objavoval nápis scroll? Riadok 60 tlačí tri hviezdičky na spodok obrazovky a potom vymazáva nastávajúcu pozíciu lode. To preto, aby mohlo byť prevedené rolovanie (scroll). Zaisťuje to posledný príkaz PRINT na riadku 60. Riadok 70 číta z klávesnice pozíciu, kam loď tlačiť. Riadok 80 kontroluje, ktorý znak je na novej pozícii (či je to loď alebo hviezdička-v prípade kolízie). Ak to nie je hviezdička, potom sa program vracia na riadok 40 a hra pokračuje. V prípade kolízie hra končí a objaví sa skóre a blikajúca loď.

```
 1 REM Hviezdy
10 RANDOMIZE
20 LET score=0
30 LET across=INT (RND*32)
40 LET score=score+1
50 POKE 23692,255: REM Scroll
60 PRINT AT 21,RND*31;"*";AT 21,RND*31;"*";
   AT 21, RND*31;"*";AT 10,across;" ";AT 21,31''
70 LET across=across-(INKEY$ ="5" AND across>0)+(INKEY$=
   "8" AND across< 31)
80 IF SCREEN$ (10,across)<>"*" THEN PRINT AT 10, across;
   INVERSE 1;"V": GO TO 40
90 PRINT AT 0,0;"Body: ",score;AT 10,across;
   FLASH 1;"V"
```

SCREEN$ má ale len obmedzenú možnosť v rozpoznávaní znakov. Dokáže rozlíšiť len znaky zo sady znakov v ROMe, teda od medzery (Space) po symbol (c) , tj. od CHR$ (32) po CHR$(127). Užívateľom definovanú grafiku a blokovú grafiku na číslicových klávesách nedokáže rozlíšiť. To je však podstatné obmedzenie funkcie SCREEN$, pretože práve tieto znaky sú najčastejšie používané v grafických hrách.

Cesta k zavŕšeniu tohto problému je v použití funkcie ATTR a PRINT v rôznych farbách. Napr. strielate rakety na nejaký objekt. Objekt môže byť zelený, raketa červená a nosič rakiet trebárs modrý. K tomu, aby ste zistili, či ste zasiahli, stačí kontrolovať, či sa na miestach, kadiaľ sa raketa pohybuje, objaví zelená farba, pomocou funkcie ATTR.

Tak je možné veci zariadiť tam, kde funkcia SCREEN$ z najrozličnejších dôvodov zlyháva. Tak napr., ak je testovaný objekt modrý na žltom pozadí, pri normálnom jase a neblikajúci, môžete prítomnosť uvedeného objektu testovať takto:

```
IF ATTR (Y,X)=49 THEN ....
```

Pričom berieme na vedomie, že funkcia ATTR vracia ako výsledok číslo, ktoré je: (blikanie *128)+ (jas*64) +(farba papiera *8)+(farba atramentu).

Jedna vec, ktorú musíme mať na pamäti, je, že všetky voľné miesta (medzery) musia mať inú hodnotu ATTR ako ostatná použitá grafika. Ak bola použitá všeobecne trebárs farba atramentu modrá a raketa i medzery po

vyplnení voľného priestoru boli tlačené tiež modro, potom funkcia ATTR nemôže rozoznať, čo je raketa a čo voľný priestor!

Ľahká cesta, ako obísť tento problém, je špecifikovať všeobecnú farbu atramentu rovnakú ako je farba podkladového papiera ešte pred CLS, napr. takto:

```
FLASH 0: BRIGHT 0: INK 6: BORDER 6: PAPER 6: CLS.
```

Tým sa nastaví hodnota ATTR pre všetky volné miesta na 54 (obrazovka bude celá žltá). Nemôžete potom však použiť v priebehu programu žltý INK na žltom PAPER, pretože to nie je vidieť. Inú farbu ale pre INK nie je možné využiť, pretože to bude tlačené v tejto práve zvolenej farbe, zatiaľ čo voľný priestor bude mať trvale nastavenú farbu (viď vyššie). Jediný problém je, že po skončení programu nemôžete vidieť LIST (rovnaká farba INK a PAPER), pokiaľ neurobíte niečo ako INK 9: STOP, keď program príde na koniec.

Tu je príklad vývoja hviezdneho programu, ktorý využíva uživateľskú grafiku a funkciu ATTR pre určenie, či došlo ku kolízii alebo nie. Uživateľská grafika sú tri grafické A v riadku 70 a grafické B v riadkoch 100 a 110. Riadok 20 nastavuje globálne atribúty na 0, takže miesta na obrazovke, kde nebude žiadny PRINT, budú mať ATTR=0. Hviezdy sú tlačené biele oproti čiernemu pozadiu, čím majú ATTR iné než 0. Teda keď program na riadku 100 kontroluje či nedošlo ku kolízii, potom iba zistený prázdny priestor a teda ATTR=0 dovolí pokračovať ďalej. Akýkoľvek iný ATTR program zastaví. Tu je výpis programu:

```
  5 GO SUB 1000
 10 RANDOMIZE
 20 FLASH 0: BRIGHT 0: INK 0: BORDER 0: PAPER 0: CLS
 30 LET score=0
 40 LET across=INT (RND*32)
 50 LET score=score+1
 60 POKE 23692,255: REM Auto scroll; nasl. tri A sú v graf. režime
 70 PRINT INK 7;AT 21,RND*31;"A";AT 21,RND*31;"A"; AT 21,RND*31;"A"
 80 PRINT AT 10,across;" ";AT 21,31''
 90 LET across=across-(INKEY$="5" AND across>0)+(INKEY$= "8" AND
    across<31)
100 IF ATTR (10,across)=0 THEN PRINT AT 10,across; INK 4;"V": GO TO 50
110 PRINT AT 0,0; INK 7;"Body: ";score;AT 10,across;INK 4;FLASH 1;"V"
120 INK 9: STOP
1000 REM Vytvor grafickú hviezdu
1010 FOR a=0 TO 15
1020 READ udg
1025 REM A v nasl. riadku je v graf. režime
1030 POKE USR "A" +a, udg
1040 NEXT a
1050 DATA 16,56,254,124,56,108,130,0,195,165,90,66,36,36,24,24
1060 RETURN
```

Občas je potrebné použiť buď SCREEN$ alebo ATTR a my musíme medzi oboma zvoliť ten pravý. Voľba je pomerne jednoduchá, väčšinou býva určujúca rýchlosť. Spustite oba nasledujúce programy a zmerajte, ako dlho každý z nich trvá.

```
10 FOR a=1 TO 1000
20 LET b$=SCREEN$ (2,2)
30 NEXT a
```

```
10 FOR a=1 TO 1000
20 LET b=ATTR (2,2)
30 NEXT a
```


Ten, ktorý používa SCREEN$, trvá asi 12 sec., ten s ATTR asi 9 sec. Z toho vidíte, že je rýchlejšie získať atribúty určitého miesta obrazovky ako získať znak, ktorý je v tomto mieste. Občas sa môžete dostať do ťažkostí pri získavaní farby papiera alebo atramentu, ak je miesto na obrazovke zjasnené (BRIGHT) alebo bliká (FLASH). Funkcia ATTR vracia atribúty zvoleného miesta obrazovky ako jedno číslo od 0 do 255, ktoré obsahuje všetky štyri atribúty. Pre ich jednotlivé rozlíšenie musíme poznať, ako sú atribúty uchovávané v pamäti.

Tabuľka ukazuje len bajt atribútov:

bit 7	bit 6	bit 5 bit 4 bit 3	bit 2 bit 1 bit 0
blikanie	jas	papier	atrament

```
bity 0 až 2    farba INK dvojkovo (binárne)
bity 3 až 5    farba PAPER dvojkovo
bit 6          jas (BRIGHT) -ak je 1, potom je miesto zjasnené
bit 7          blikanie (FLASH) -ak je 1, potom miesto bliká
```

K získaniu jednotlivých atribútov možno využiť užívateľom definovaných funkcií (DEF FN). Tieto nasledujúce FN udávajú atribúty blikania (FN f), jasu (FN b), papiera(FN p) a atramentu (FN i). Ak využívate niektorý výraz len raz v programe, potom nemusíte písať všetky 4 definície, stačí len napísať tu potrebnú v plnom tvare v okamihu, kedy ju potrebujete. Číslo, ktoré funkcia vráti, je potom číslo farby, napr. ak bol papier modrý, funkcia FN p vráti číslo 1.

X a y sú normálne používané súradnice na obrazovke (x-stĺpec, y-riadok). Ľavý roh hore má súradnice (0,0).

```
10 DEF FN f(y,x)=INT (ATTR (y,x)/128)
20 DEF FN b(y,x)=INT ((ATTR (y,x)-INT (ATTR (y,x)/128)*128)/64)
30 DEF FN p(y,x)=INT (ATTR (y,x)-INT (ATTR (y,x)/64)*64)/8)
40 DEF FN i(y,x)=INT (ATTR (y,x)-INT (ATTR (y,x)/8*8)
```

Občas sa objaví situácia, kde potrebujete určiť na obrazovke znak s užívateľom definovanej grafiky (UDG), k čomu funkciu ATTR nemožno použiť. Možno však použiť SCREEN$, pokiaľ rozumieme jej funkcii. Funkcia SCREEN$ získava zo systémovej premennej hodnotu, ktorá jej umožní nájsť v ROM miesto, kde začína generátor (sada) znakov, v ktorom potom funkcia nájde vzorku bodov, ktorá zodpovedá tým bodom, ktoré sú na skúmanom mieste obrazovky.

Ak sme zmenili hodnotu v tejto systémovej premennej, čím sme funkcii SCREEN$ umožnili, aby si myslela, že pracuje s normálnym generátorom znakov v ROM, potom potrebujeme malú aritmetickú manipuláciu aby sme získali správnu hodnotu CHR$ (umožňuje myslieť si, že CHR$ 32 je UDG A, i keď normálne je CHR$ 32 začiatok generátora znakov).

Systémová premenná prichádzajúca v hru je 23606/23607, ktorá vo

svojich dvoch bajtoch obsahuje číslo, ktoré je o 256 menšie ako adresa začiatku generátora znakov v ROM. Existuje tiež dvojbajtová systémová premenná 23675/23676, ktorá nám hovorí, kde začína (udáva počiatočnú adresu) UDG (presne).

Takže všetko, čo potrebujeme urobiť, je to, že prenesieme obsah 23675/23676 do 23606/23607, pričom od vyššieho bytu odčítame 1, aby sme získali rozdiel 256.

Toto je najlepšie napísať ako podprogram. Ten potom zmení ukazovateľ generátora znakov na UDG, kontroluje zvolené miesto na obrazovke, vyťahuje hodnotu CHR$ tohoto miesta a potom vracia všetko do pôvodného stavu (ukazovateľ späť na začiatok generátora znakov). X a y sú normálne PRINT pozície, určujúce zvolené miesto na obrazovke.

```
8000 POKE 23606,PEEK 23675: POKE 23607, PEEK 23676-1
8010 LET a$=SCREEN$ (y,x)
8020 IF a$<>"" THEN LET a$=CHR$ (CODE a$+112)
8030 POKE 23606,0: POKE 23607,60
8040 RETURN
```

Pomocou tohoto podprogramu však nebudú presne rozpoznané voľné miesta na obrazovke, pokiaľ nie je v UDG definované niečo ako medzera (Space). Je to len malé upozornenie na jemnú nuansu, pretože medzera (space) sa využíva v hojnej miere - všetky voľné miesta na obrazovke sú vlastne medzery. K vyriešeniu problému sú dve cesty: buď definovať v UDG medzeru alebo podprogram upraviť, asi takto:

```
8000 LET a$=SCREEN$ (y,x): IF a$= " " THEN RETURN
8010 POKE 23606,PEEK 23675: POKE 23607, PEKK 23676-1
8020 LET a$=SCREEN$ (y,x)
8030 IF a$<>"" THEN LET a$=CHR$ (CODE a$+112)
8040 POKE 23606,0: POKE 23607,60
8050 RETURN
```

Dostať SCREEN$ k rozlíšeniu bodovej grafiky CHR$ 128 až CHR$ 143 už nie je také ľahké. Tu už neexistuje vzorka bodov pre ich výrobu, ale sú "vypočítavané" až keď je potrebné ich tlačiť (alebo zobraziť na obrazovke). Aby bolo možné vyriešiť i tento problém, je potrebné vytvoriť niekde v pamäti bodový vzorec blokovej grafiky a potom zmeniť ukazovateľ začiatku generátora znakov na toto miesto pamäte alebo nadefinovať UDG ako blokovú grafiku. Potom možno použiť funkcie SCREEN$ tak, ako bolo popísané vyššie. Pamätajte, aj keď je všetkých znakov blokovej grafiky 16, je nutné kontrolovať len 8. Tie ostatné sú ich inverzné obrazy. SCREEN$ to zvládne, ako ukazuje nasledujúci príkaz (môže to byť priamy príkaz alebo programový riadok).

```
CLS:PRINT CHR$ 143: PRINT SCREEN$ (0,0), CODE SCREEN$ (0,0)
```

CHR$ 143 je Graphics Shift a 8 alebo čierna krabička (alebo krabička plná bodov vo farbe INK). SCREEN$ ich rozozná ako riadnu medzeru (ona je to vlastne inverzná medzera).

Existuje ešte jedna celkom ľahká cesta, ako vec vyriešiť. Prekopírovať niekam do pamäte celý generátor znakov a málo využívané znaky (napr. hranaté a zložené zátvorky apod.) predefinovať na blokovú grafiku. Ukazovateľ začiatku genarátora znakov potom nastaviť na začiatok týchto znakov a je to.

N E V Y M A Z A T E Ľ N É P R O G R A M O V É R I A D K Y

Nebolo by to krásne, mať v programe riadok ako napr.:

```
10 REM (c) Juro Jánošik 1713
```

a vedieť, že sa nedá zmeniť alebo vymazať, čím zabránime cudzím ľuďom program kopírovať (aspoň morálne, keď už nie fakticky). Vymazanie riadku hore ja ľahké: 10 a ENTER. Všetko, čo je potrebné, je nájsť metódu, ktorá umožní do listingu (výpisu) vložiť riadok, ktorý možno odstrániť len ťažko.

Prvá časť odpovede znie, že pokiaľ sa Vám podarí vložiť tieto údaje na riadok 0, tak tento nemôže byť normálnou cestou vymazaný, pretože je spájaný s priamymi príkazmi napr. priamy príkaz PRINT po vykonaní uvedie správu 0 OK 0:1, čo znamená: všetko je OK v prvom príkaze riadku 0 - priameho príkazu). Keď ale skúsite napísať

```
0 REM (c) Juro Jánošik 1713
```

budete odmenený krásnou správou C, Nonsence in Basic (C, nezmysel v Basicu). Tak to by bolo.

Všetko, čo potrebujeme urobiť, je napísať riadok s normálnym číslom (napr. 10) a potom toto číslo zmeniť na 0. Ťažké? Ani nie. Môžeme to urobiť tak, že hľadáme v programe číslo riadku, nasledované REM a potom urobíme POKE, pokiaľ nedostaneme to, čo hľadáme. Je to zdĺhavé, zaiste.

Lepší spôsob je použiť systémové premenné NXTLIN uložené v adresách 23637/8, ktoré obsahuje adresu začiatku ďalšieho programového riadku (pozor! riadku, nie príkazu).

Spectrum manuál nás informuje, že každý riadok programu v Basicu začína číslom riadku, ktoré je uložené v dvoch bytoch v poradí vyšší bajt (More Significant Byte-MSB), nasledovaný nižším bajtom (Less Significant Byte-LSB). Teda riadok 1 bude dvojbajtovo 0,1 a riadok 258 bude 1,2 (1*256 +2). Takže ak urobíme POKE 0 do oboch bytov, získame náš zdanlivo nevymazateľný riadok. Tu je postup:

```
1 LET a=PEEK 23637+256*PEEK 23638: POKE a,0: POKE a+1,0: STOP
2 REM (c) Juro Jánošik 1713
```

Spustite tento program. Teraz urobte LIST a zistíte, že riadok číslo 2 sa zmenil na riadok číslo 0. Nie sú však zaradené podľa čísel. Nevadí.. Každý ďalší zapísaný riadok bude zaradený správne. Riadok číslo 1 už nepotrebujeme, tak ho normálnou cestou vymažeme, aby sme ostatným znemožnili urobiť to tak ako my. Teraz bude výpis vyzerať:

```
0 REM (c) Juro Jánošik 1713
```

Skúste riadok vymazať alebo zmeniť. Bezpečné, že ? Ak ho chcete vymazať, musíte prejsť celé toto POKEovanie znovu. Keď ale budete premýšľať, ako to realizovať, narazíte na problém - nemôžete teraz použiť systémovú premennú NXTLIN, pretože riadok 0 je teraz prvý riadok programu, všetky ďalšie riadky prechádzajú triedením a radia sa za riadok 0. NXTLIN dá správnu adresu pre POKE jedine v prípade, že sa použije pred riadkom, ktorý má byť POKEovaný. Tvrdý oriešok. Nechávame vás, aby ste si postup vypracovali sami.

Existuje niekoľko ciest, všetky riešia problém obchvatom. Nemá totiž

cenu prezrádzať niečo, čomu sme chceli zabrániť. Môžete nevymazateľný riadok umiestniť do všetkých častí programu, do všetkých stránok výpisu a ak ste trpezlivý, potom môžete pridať i BRIGHT, FLASH, rôzne farby apod.

Pre bežné používanie môžete program dať na pásku a príkazom MERGE ho začleniť do vašich programov. Môžete použiť buď:

```
SAVE "copyright" LINE 1
```

čo spôsobí, že program sa sám spustí od riadku 1 a vytvorí riadok 0. Alebo môžete mať na páske len riadok 0 a ten vsunúť do programu ako jeho prvý riadok. MERGE sa s riadkom 0 ľahko vysporiada. Keď vytvárate tento "copyright" riadok 0, doporučujeme aby ste zadali PAPER biely, BRIGHT 1, INK čierny, FLASH 1. To je skutočne výrazné.

Keď už sme pritom, riadok 0 nie je jediný zábavný, ktorý môžete POKEovať. Skúste POKE napr. 50 a 0 v riadku 1. Prečo je kurzor pred číslom riadku a ukazuje na opačnú stranu? Je to tým, že číslo riadku je väčšie ako 9999 a interpréter programu nedokáže toto číslo dekódovať presne. Používa potom znak menší namiesto čísla.

Ešte väčšia zábava je, ak urobíte POKE 64 a 0 v riadku 1. Kam zmizol program? Akékoľvek číslo riadku väčšie ako 16383 spôsobí, že program nemožno "vylistovať", pretože LIST nemôže ísť za riadok 16383. Program je stále v počítači, ale ho nevidieť, pokiaľ nezistíme správne čísla a pomocou POKE uvedieme veci do normálneho stavu naspäť.

S T L A Č N E J A K Ú K L Á V E S U

Upozornenie: Výrobca Didaktiku M, v.d. Skalica previedol na Didaktiku M dielčie zmeny, preto údaje v tejto kapitole sa líšia od skutočného stavu na Didaktiku M. Je to spôsobené tým, že na ZX Spectre sú najvyššie tri bity portov klávsnice vždy jednotky a na Didaktiku sú tu premenné hodnoty. Ak chcete, aby popísané metódy z tejto kapitoly fungovali aj na Didaktiku M, musíte vždy hodnotu z príslušného portu klávesnice pevne nastaviť s bitmi 5.,6.,7. na jednotku. Je to možné zrealizovať manipuláciou s bitmi ako to robíme pri práci v bajtoch pre farbu.

Spoločným príznakom využitia je riadenie chodu programu v závislosti na príkaze od operátora. Ako príklad môže byť výpis inštrukcií a potom upozornenie operátorovi, aby stlačil čokoľvek po skončení čítania, aby program pokračoval. Tato časť programu môže vyzerať napr. takto:

```
.......(inštrukcie)
1000 PRINT "Stlač ľuboľnú klávesu okrem SHIFT, aby program pokračoval"
1010 IF INKEY$="" THEN GO TO 1010
```

To je v poriadku, ale keď stlačíte Caps Shift alebo Symbol Shift, bude vás program ignorovať a ostatný ľudia si povedia: "To je ale hlúpy program". Tu sú príklady, ako to obísť:

```
.......(inštrukcie)
1000 PRINT "Stlač ľuboľný kláves."
1010 IF INKEY$="" THEN GO TO 1010

1000 PRINT "Stlač ENTER."
1010 INPUT a$
```


Určite ste zistili, že INKEY$ nereaguje na stlačenie niektorej z kláves Shift, ale na súčasné stlačenie oboch. Ich súčasné stlačenie (akoby EXTEND mód) vyrobí CHR$ 14.

Oba príklady vyššie uvedené sú dobré, ale nebolo by krásne, keby sme mohli pre pokračovanie stlačiť skutočne ktorúkoľvek klávesu? Akúkoľvek, čo samozrejme znamená ktorúkoľvek zo 40 kláves na klávesnici počítača ZX Spectrum, vrátane oboch Shift kláves. Tu je príklad:

```
1000 PRINT "Stlač ľubovoľný kláves..."
1010 IF INKEY$="" AND IN 65278=255 AND IN 32766=255 THEN GO TO 1010
```

Poznámka: Hodnoty IN 65278 a IN 32766 si na Didaktiku M musíte najprv upraviť tak, ako sme spomenuli v úvode kapitoly, čiže nastaviť tri bity 7.,6. a 5. bit na jednotku.

Klávesnica (jej logika) je umiestnená v niečom, čomu sa hovorí I/O priestor, mienené INPUT/OUTPUT (VSTUP/VÝSTUP). Je to súbor metód pre získanie informácií z a do počítača a od vonkajšieho okolia.

Konektory MIC a EAR, vnútorný reproduktor, klávesnica, tlačiareň, mikrodrajvy, disková jednotka, interfejs RS232 - to sú všetky príklady činnosti I/O. Čo sa týka užívateľa /operátora, je najvýznamnejší rozdiel v adresovaní pamäte ten, že PEEK a POKE pracujú iba s pamäťou, či už v ROM alebo RAM. Príkazy I/O (vstupno - výstupných zariadení) systémov IN a OUT sa týkajú získavania informácií do a z počítača z a do okolitého sveta. Existuje 65536 týchto I/O portov, akoby bolo k dispozícii 65536 pamäťových miest, ale tie môžu byť alebo nie sú využité všetky, podobne ako nie je využitý celý pamäťový priestor 16 K počítača Spectrum.

V Basicu sú dva príkazy, ktoré obsluhujú tieto I/O porty. Sú to IN a OUT, ktoré si môžeme zhruba predstaviť ako PEEK a POKE. Funkcie ako INKEY$ majú teda prístup k týmto I/O portom, ale využívajú strojový kód odpovedajúci IN a OUT svojou vlastnou cestou. Ďalšia otázka znie, ako sa dozvieme, ktorý port sa využíva pre ten ktorý účel. Hlava 23 Spectrum manuálu o tom podáva podrobné vysvetlenie, ale to najdôležitejšie, čo je v tomto článku, sa týka klávesnice.

Ako príklad použijeme OUT a budeme sa hrať s PORT 254 (s portom - bránou - vstupom 254), ktorý okrem iného riadi farbu BORDERu a reproduktor. Môžeme si to znázorniť týtmo krátkym programom:

```
10 OUT 254, INT (RND *256)
20 GO TO 10
```

Pri jeho spustení budete počuť klopavé zvuky z reproduktora a BORDER na obrazovke akoby sa zbláznil. Farby sa menia tak rýchlo, že ich môžete vidieť niekoľko naraz. Zistujete zaiste, že pokiaľ tento program beží, dolné dva riadky obrazovky farbu nemenia (a normálne by mala byť rovnaká ako BORDER). BORDER mení totiž farbu dolnej časti obrazovky len ak niečo píše. Pokiaľ rozumiete dvojkovým číslam, následujúci diagram vám pomôže objasniť, ako môže 8 bitov portu 254 riadiť niekoľko činností súčasne.

Všetky I/O porty sú osembitové bajty, podobne ako pamäťové miesta.

128	64	32	16	8	4	2	1	hodnota
D7	D6	D5	D4	D3	D2	D1	D0	bit
			riadi repro.	riadi MIC	ovláda BORDER			

Pretože sú použité iba bity 0 až 4, zmeníme riadok 10 predchádzajúceho programu:

```
10 OUT 254,INT(RND*32)
20 GO TO 10
```

pretože použité bity môžu načítať od 0 do max. 31.

Pre nás najužitočnejšie sú porty, ktoré sú spojené s klávesnicou. Je to osem portov, každý obsluhuje 5 kláves v ľavej prípadne v pravej polovici klávesnice. Napr. PORT 61438 je spojený s radou piatich kláves, a to 6 (bit 4), 7 (bit 3), 8 (bit 2), 9 (bit 1) a 0 (bit 0). Viď. obr. na nasledujúcej strane (51).

Skúste nasledujúci program, ktorý tlačí opakovane hodnotu v porte 61438. Tlačte klávesy 6 až 0, abyste videli, čo sa deje. Stlačte viac kláves súčasne.

```
10 PRINT IN 61438
20 PAUSE 100
30 GO TO 10
```

Spustte tento program a pozorujte, že sa tlačí stále 255, pokiaľ nestlačíte niektorú z kláves 6 až 0. Získané hodnoty vyzerajú celkom náhodne, pokiaľ nepochopíte, ako sú vytvárané. Už iste viete, že 255 zodpovedá stavu "žiadna klávesa nestlačená". Iste tiež viete, že 255 je dvojkové 11111111. Takže keď číslo, ktoré sme získali po stlačení niektorej klávesy, bolo vždy menšie ako 255, dokážete si predstaviť, že stlačenie klávesy spôsobí, že zodpovedajúci bit sa zmení z 1 na 0.

Preštudujte diagram na nasledujúcej strane, ktorý ukazuje, ako je ktorý bit toho ktorého portu spojený s príslušnou klávesou.

Spustite znovu program hore a zakaždým, keď stlačíte klávesu, odčítajte číslo pod klávesou od 255. Mali by ste dostať rovnaké číslo, ako je to, ktoré píše program na obrazovku.

Celá vec nie je možno ešte úplne jasná, ale dúfam, že v praxi to bude iné. Symboly D0 a D4 nadpísané v diagrame nad klávesami predstavujú jednotlivé bity I/O portov. V tomto prípade sú využité len bity 0 až 4, pretože port kontroluje len 5 kláves.

Ukážme si niekoľko jednoduchých príkladov využitia funkcie IN:

Kontrola, či je stlačená klávesa R:

```
IF IN 64510=(255-8) THEN PRINT "R je stlačené."
```

Kontrola, či je stlačená klávesa Y:

```
IF IN 57342=(255-16) THEN PRINT "Y je stlačené."
```


	PORT BIT					PORT BIT					
	D0	D1	D2	D3	D4	D4	D3	D2	D1	D0	
PORT 63486	1	2	3	4	5	6	7	8	9	0	PORT 61438
	1	2	4	8	16	16	8	4	2	1	
PORT 64510	Q	W	E	R	T	Y	U	I	O	P	PORT 57342
	1	2	4	8	16	16	8	4	2	1	
PORT 65022	A	S	D	F	G	H	J	K	L	ENTER	PORT 49150
	1	2	4	8	16	16	8	4	2	1	
PORT 65278	Caps shift	Z	X	C	V	B	N	M	Symb. shift	Space	PORT 32766
	1	2	4	8	16	16	8	4	2	1	

hodnoty, ktoré musia byť odčítané od 255, či je stlačená klávesa.

Kontrola, či je stlačená klávesa SPACE:

```
IF IN 32766=(255-1) THEN PRINT "SPACE je stlačená."
```

Samozrejme, že výrazy v zátvorkách sa nemusia písať v tvare ako je v príklade, ale stačí napísať len výsledné číslo. Spôsob, ako to napíšte, záleží len na vás, dôležité je,aby výsledok bol správny a vy ste porozumeli popisovaným veciam. Dôležité je pamätať si, že príslušný bit na odpovedajúcom porte je 0 vtedy, ak je klávesa stlačená. To vysvetľuje, ako získate hodnotu 255, keď nie je nič stlačené - všetky bity sú 1, čo je desiatkovo 255. Vezmeme napr., že stlačíme K. Port I/O spojený s príslušnou polradou je PORT 49150 (viď. diagram hore).

Každý bajt alebo port má 8 bitov:

```
BIT7 BIT6 BIT5 BIT4 BIT3 BIT2 BIT1 BIT0
1    1    1    1    1    1    1    1
```

Takto to vyzerá, keď nič nie je stlačené. Ak stlačíme K, potom príslušný port bude vyzerať nasledovne:

```
BIT7 BIT6 BIT5 BIT4 BIT3 BIT2 BIT1 BIT0
1    1    1    1    1    0    1    1
```

Teraz má port hodnotu BIN 11111011, čo je desiatkovo 255-4=251, čo je rovnaké ako 128+64+32+16+8+2+1.

Technicky vzaté, je načítanie jednotlivých bitov, správnejšie ako nami skôr použité odčítanie. Avšak pre našu aplikáciu to vyhovuje a je to ľahšie na pochopenie.

Príklad:

```
IF IN 49150=251 THEN PRINT "K je stlačené."
```

je to :

```
IF INKEY$="K" OR INKEY$="k" THEN PRINT "K je stlačené."
```

Ale je tu určitý pokrok, pretože IN môže kontrolovať, či je stlačený niektorý zo Shift kláves, čo pomocou INKEY$ testovať nemožno.

```
IF IN 65278=254 OR IN 32766=253 THEN PRINT "SHIFT je stlačený."
```

INKEY$ teda robí rozdiel medzi malými a veľkými písmenami, takže IF INKEY$="k" THEN ... nie je to isté ako IF INKEY$="K" THEN, kdežto IF IN 49150=251 THEN ... iba zisťuje, že bola stlačená klávesa K, bez ohľadu na to, či bola stlačená niektorá z kláves CAPS LOCK alebo CAPS SHIFT.

Využitie IN k prehliadaniu stavu klávesnice nám teda umožňuje kontrolovať, či bola stlačená len jedna alebo viac kláves:

```
IF IN 49150=(255-2-4) THEN PRINT "K a L bolo stlačené."
```

Jedna z možných aplikácií tohoto môže byť v hrách, kde sa používajú napr. klávesy pre pohyb kurzora (5,6,7 a 8) pre riadenie smeru pohybu v hre. Väčšina hier dovoľuje len pohyb jedným smerom. Ak využijeme práve získané vedomosti, môžeme sa pohybovať i diagonálne, podobne ako s joystickom.

Skúste nasledujúci program pre kreslenie čiary vľavo, dole, vpravo a nahor. Ovládanie je 5,8 - vľavo,vpravo a 6,7 - dole,hore. Stlačením oboch sa pohybujete diagonálne napríklad šikmo vpravo nahor. Pri použití INKEY$ by to nebolo možné, pretože nemôžete kontrolovať súčasne stlačenie oboch kláves.

```
10 LET x=0
20 LET y=0
30 PLOT x,y
40 LET a=IN 61438
50 LET x=x+(a=251 OR a=243)
60 LET y=y+(a=247 OR a=243)
70 GO TO 30
```

I keď možno klávesy 5,6,7 a 8 využiť pre riadenie pohybu na obrazovke, nešlo by to ešte ľahšie? Ich výhoda pri hrách je v tom, že sú pekne pohromade a že je možné ich ľahko "čítať" pomocou INKEY$ pre riadenie hodnôt premenných, napr. LET X=X+(INKEY$="8")-(INKEY$="5"). Avšak nevýhodou zasa je, že sú blízko a neprehľadne zoradené a vyžadujú pre ovládanie takmer ekvilibristickú virtuozitu prstov.

Systém popísaný ďalej nám umožňuje použiť pre riadenie pohybu všetkých 40 kláves, takže už sa nebudeme musieť hnevať, že sme v rýchlosti siahli vedľa a pod. Klávesnica sa rozdelí do 4 častí, každá po 10 klávesách pre riadenie jedného smeru:

1	HORE	0
VĽAVO Q - T		VPRAVO Y - P
VĽAVO A - G		VPRAVO H-ENTER
CAPS SHIFT	DOLE	SPACE

Tak spôsobí stlačenie niektorej klávesy v hornej rade pohyb nahor, v spodnej rade pohyb dole, v ľavej polovici oboch prostredných radov pohyb vľavo a v pravej polovici oboch prostredných radov pohyb vpravo. Stlačenie kláves z rôznych skupín spôsobí pohyb diagonálne.

Poznámka:

Samozrejme, že u rôznych variácií počítačov s odlišnou klávesnicou môže byť rozdelenie klávesnice mierne modifikované. Napríklad u Didaktiku M, Gama a Sinclair ZX Spestrum je rozmiestnenie klasické, presne podľa programu. Narozdiel od toho ZX Spectrum +, 128 K, +2, Sam Coupé, čo sú novšie modely odvodené od ZX Spectrum je klávesnica obohatená o ďalšie klávesy, čiže rozmiestnenie nových kláves spôsobuje menší prehľad pri použití tohoto programu.

Uvedený program je veľmi jednoduchý. Kreslí čiaru v smere, ktorý si zvolíte, stlačením príslušnej klávesy. Avšak pretože je jednoduchý, akonáhle narazíte na okraj obrazovky, nastanú ťažkosti. Pri rozbore programu, hlavne riadok 30 a 40, si všetko porovnávajte so spomínanými diagramami.

```
10 LET x=120
20 LET y=90
30 LET x = x + ( IN 49150 <> 255 OR IN 57342 <> 255) - (IN 64510<>255
   OR IN 65022<>255)
40 LET y = y - ( IN 65278 <> 255 OR IN 32766 <> 255) + (IN 63486<>255
   OR IN 61438<>255)
50 PLOT x,y
60 GO TO 30
```

V Ý P I S R E Ť A Z C O V Ý C H P O L Í

Skúste tento program:

```
10 DIM a$ (12,9)
20 FOR a=1 TO 12
30 READ a$ (a)
40 PRINT a$ (a);",";
50 NEXT a
60 PRINT CHR$ 8;"."
70 DATA "Január", "Február", "Marec", "Apríl", "Máj", "Jún", "Júl",
    "August", "September", "Oktober", "November","December"
```

Niektoré mená sú oddelené radou medzier, pretože každý reťazec v poli a$ má rovnakú dĺžku (tj. 9 znakov). Zakaždým, ak označujete niektorý z týchto reťazcov, označujete všetky znaky reťazca, i keď počítač musel doplniť nejaké medzery. Napr. pre a$(5), čo je Máj, teda len 3 platné písmená, počítač doplní 6 medzier, aby bola dodržaná dĺžka 9 znakov. Výsledkom je hŭf neužitočných medzier, vytlačených za znakmi, čo na obrazovke nevyzerá práve pekne. Problém môže byť vyriešený dvoma hlavnými cestami:

1. Pri tlačení sa odzadu hľadá prvý platný znak, pričom sa medzery vynechávaju:

```
 10 DIM a$ (12,9)
 20 FOR a=1 TO 12
 30 READ a$ (a)
 40 REM Výpis bez koncových medzier
 50 FOR b=LEN a$(a) TO 1 STEP-1
 60 IF a$(a,b)<>" " THEN PRINT a$ (a,TO b);",";: GO TO 80
 70 NEXT b
 80 NEXT a
 90 PRINT CHR$ 8;"."
100 DATA "Január", "Február", "Marec", "Apríl", "Máj", "Jún", "Júl",
     "August", "September", "Október", "November","December"
```

Výsledok výpisu:

Január,Február,Marec,Apríl,Máj,Jún,Júl,August,September,
Október,November,December.

Všetko čo program robí je, že načíta mená 12 mesiacov do reťazcového poľa a$ (12,9) a potom prevádza tlač pomocou slučky FOR-NEXT b, ktorá postupuje pri prehľadávaní jednotlivých reťazcov od konca a hľadá prvý znak, ktorý je iný ako medzera. Ten potom tlačí v tomto mieste pomocou a$ (a, TO b). Všimnite si čiarky. Tento výraz je skratka od a$ (a,1 TO b).

2. Využitím SLI (String Lenght Inicator-indikátor dĺžky reťazca) k záznamu dĺžky reťazca, čo ušetrí čas pri prehľadávaní pred tlačou. Táto metóda teda umožňuje správnu tlač slov, ktorá končí medzerami. SLI sa normálne ukladá ako prvý znak v každom reťazci.

Program pre demonštráciu:

```
10 DIM a$(12,10)
20 FOR a=1 TO 12
30 READ s$
40 LET a$(a)=CHR$(LEN s$+1)+S$
50 PRINT a$(a,2 TO CODE a$(a));",";
60 NEXT a
70 PRINT CHR$ 8;"."
80 DATA "Január", "Február", "Marec", "Apríl", "Máj", "Jún", "Júl",
    "August", "September", "Október", "November","December"
```

Všimnite si, že v riadku 10 bol ku každému reťazcu poľa a$ pričítaný ďalší prvok - k uloženiu SLI. Číslo predstavujúce dĺžku reťazca musí byť celé číslo. Rozsah 0 až 255 je postačujúci pre väčšinu aplikácií. Ak je

treba väčšie číslo (až do 65535), potom je pre jeho uloženie potreba dvoch bajtov.

Slučka a číta mená mesiacov z DATA do obecne použitej premennej s$. Pretože reťazcove premenné v s$ sú len tak dlhé, ako dlhé majú byť, použijeme LEN s$, aby sme zistili dĺžku každého reťazca a uložili ju potom ako prvý znak do reťazcov poľa a$ pomocou CHR$ (LEN s$+1). Toto je ten pridaný prvok, o ktorom sme hovorili skôr. Zbytok reťazca je potom kópia premennej s$.

Takto vyzerá potom uloženie slova Marec:

	1	2	3	4	5	6	7	8	9	10
a$(3)	6	M	a	r	e	c	SPACE	SPACE	SPACE	SPACE

SLI .....A$(3,2 TO CODE A$(3))

Takže prvý prvok reťazca A$(3) obsahuje znak 6. Dĺžka slova Marec je 5 písmen. Začína na druhom prvku a končí na šiestom. Prečo bola pridaná 1 k dĺžke slova Marec, je jasné - SLI je prvý znak. Bolo by totiž nerozumné dávať SLI až na koniec, pretože by sa mohlo stať, že niektoré dlhé slovo by ho mohlo prepísať. Jediný problém, ktorý môže nastať, je, že SLI bude väčšie než je deklarovaná dĺžka reťazcov poľa a$, čo spôsobí hlásenie chybného indexu. Toto môže byť odstránené pridaním riadku:

```
45 IF CODE A$(a)>=LEN a$(a) THEN LET a$(a,1)=CHR$ (LEN
    a$(a)-1)
```

Reťazce sú tlačené ako PRINT a$(a,2 TO CODE a$(a)), teda od znakov za SLI až do posledného, ktorý nie je medzera. Veľká výhoda metódy 2 oproti 1 je v tom, že môžete tlačiť slova končiace medzerami správne a pritom, ak je potrebné kopírovať takýto reťazec do iného, je to možné pomocou uloženého SLI previesť veľmi rýchlo.

Ak chcete využiť rýchlosť a vhodnosť SLI, bez toho aby ste používali vyššie uvedené "zmiešané" reťazce, nie je žiadny problém uložiť si SLI do oddeleného reťazcového alebo numerického poľa.

Najskôr reťazcové pole:

```
 1 REM Použitie reťazcového poľa
10 DIM a$(12,9)
20 DIM b$ (12)
30 FOR a=1 TO 12
40 READ s$
50 LET a$(a)=s$
60 LET b$(a)=CHR$ (LEN s$)
70 PRINT a$(a, TO CODE b$(a));",";
80 NEXT a
90 PRINT CHR$ 8;".";
100 DATA "Január", "Február", "Marec", "Apríl", "Máj", "Jún", "Júl",
     "August", "September", "Október", "November","December"
```

```
  90 REM
 100 REM
 110 REM
 120 REM
 130 REM
 140 REM
 150 REM
 160 REM
 170 REM
 180 REM
 190 REM
 200 REM
9000 RETURN
```

Program 2 - 7.5s

```
10 FOR a=1 TO 1000
20 GO SUB 9000
30 NEXT a
40 STOP
9000 RETURN
```

Program 1 trvá asi 9 sekúnd v porovnaní so 7.5 sekundami programu 2. Je to v tom, že program 1 musí pri hľadaní riadku 9000 preskakovať jeden po druhom všetky riadky pred tým. Je treba poznamenať, že vlastné číslo riadku nemá na rýchlosť vplyv, ale jeho umiestnenie v programe.

Podľa toho môžeme teda navrhnúť vhodnejšie rozloženie programu, ako napr.:

```
   1 REM Doporučené rozloženie programu
  10 GO TO 1000: REM preskočenie podprogramov
 100 REM Často používané GO SUB apod.
1000 REM Hlavný program
8000 REM Málo používané GO SUB apod.
```

Takéto rozloženie programu je vhodné tam, kde je dôležitý ušetrený čas. Ďalšiu možnosť ako ušetriť čas, je pozorné programovanie dlhých slučiek. Keď sme o slučkách hovorili, zistili sme, že existujú dva druhy, ktoré majú obdobné funkcie: slučky FOR/NEXT a slučky IF ... THEN GO TO:

Porovnajte rýchlosť oboch:

Program 3 - 9s

```
10 LET a=1
20 LET a=a+1
30 IF a <= 1000 THEN GO TO 20
```

Program 4 - 4.5s

```
10 FOR a=1 TO 1000
20 NEXT a
```

Takže slučka FOR/NEXT je asi 2 krát tak rýchla ako slučka IF ...THEN GO TO . Ako zaujímavé porovnanie skúste porovnať program 4 a program 5, ktorý používa viacnásobný príkaz na jednom riadku.

Program 5

```
10 FOR a=1 TO 100: NEXT a
```


Zistujeme, že na rozdiel od iných verzií BASICu nemôže Spectrum BASIC ponúknuť veľké množstvo ušetreného času pri použití viacnásobných príkazov na jednom riadku. Program 5 beží približne rovnako dlho ako program 4. Ale pri príkaze PRINT je možné skrátiť čas, ak použijeme viac príkazov na jednom riadku.

Program 6 - 45s

```
10 FOR a=1 TO 1000
20 PRINT AT 0,0;"Rozmýšľam"
30 PRINT AT 0,0;"Rozmýšľam"
40 PRINT AT 0,0;"Rozmýšľam"
50 PRINT AT 0,0;"Rozmýšľam"
60 NEXT A
```

Program 7 - 41s

```
10 FOR a=1 TO 1000
20 PRINT AT 0,0;"Rozmýšľam";
   AT 0,0;"Rozmýšľam";
   AT 0,0;"Rozmýšľam";
   AT 0,0;"Rozmýšľam";
30 NEXT a
```

Ak chcete zmeniť PRINT pozíciu na nový riadok, použite radšej apostrof nez TAB 0, je rýchlejší.

Program 8 - 23s

```
10 FOR a=1 TO 300
20 POKE 23692,255
30 PRINT "+";TAB 0;
40 NEXT a
```

Program 9 - 18s

```
10 FOR a=1 TO 300
20 POKE 23692,255
30 PRINT "+" '
40 NEXT a
```

Použitie riadiaceho znaku CHR$ 13 (čo je ENTER) namiesto apostrofu neprinesie žiadaný úžitok. Naopak včlenene nejakého riadiaceho znaku do reťazca často veci spomalí. Program 10 zaplňuje obrazovku znamienkami +. Program 11 zaplňuje obrazovku inverznými znamienkami +, čo musí byť zapísané do programu ako "INV. VIDEO +TRUE VIDEO".

Program 10 - 4.7s

```
10 FOR a=1 TO 704
20 PRINT "+";
30 NEXT a
```

Program 11 - 5.5s

```
10 FOR a=1 TO 704
20 PRINT "+";
30 NElT a
```

Naviac porovnajte program 11 s programom 12, ktorý používa funkciu INVERSE. To je ešte pomalšie než 11. Skúste experimentovať podobne s FLASH a BRIGHT.

Program 12 - 6.2s

```
10 FOR a=1 TO 704
20 PRINT INVERSE 1;"+";
30 NEXT a
```

Niektoré počítače majú premenné, nazývané celočíselné premenné (integer). Tie sú potom obsluhované omnoho rýchlejšie než premenné,

pracujúce v plávajúcej desatinnej čiarke. Na Spectre takéto celočíselné premenné nie sú, ale Spectrum môže mať špeciálne zastupenie pre malé celé čísla, ktoré sa používa automaticky pre čísla, ktoré ležia medzi -65535 a +65535 a sú celé.

Program 13 - 11.3s

```
 5 POKE 23692,255
10 FOR a=1 TO 600
20 PRINT a;
30 NEXT a
```

Program 14 - 15.3s

```
 5 POKE 23692,255
10 FOR a=1.3 TO 600.3
20 PRINT a;
30 NEXT a
```

Programy 13 až 18 ukazujú, ako používanie celých čísel zrýchli chod programov. Pretože nemáte možnosť ovládať toto "automatické zceločíselňovanie", je najlepšie, ak zaistíte aby sa v programoch negenerovali čísla s desatinými miestami, ktoré spomalujú chod programov.

Program 15 - 12.1s

```
 5 POKE 23692,255
10 FOR a=1 TO 600
20 PRINT 1.3;
30 NEXT a
```

Program 16 - 9.1s

```
 5 POKE 23692,255
10 FOR a=1 TO 600
20 PRINT 1;
30 NEXT a
```

Program 17 - 12.3s

```
 5 POKE 23692,255
 7 LET b=1.3
10 FOR a=1 TO 600
20 PRINT b;
30 NEXT a
```

Program 18 - 9.2s

```
 5 POKE 23692,255
 7 LET b=1
10 FOR a=1 TO 600
20 PRINT b;
30 NEXT a
```

Keď už sme pri tlači, pozrime sa, ako je to s tlačou čísel na

obrazovke. Programy 19, 20 a 21 porovnávajú tlač reťazcových konštánt, celočíselných numerických konštánt a neceločíselných numerických konštánt.

Program 19 - 9.8s

```
 5 POKE 23692,255
10 FOR a=1 TO 1000
20 PRINT "69";
30 NEXT a
```

Program 20 - 17.2s

```
 5 POKE 23692,255
10 FOR a=1 TO 1000
20 PRINT 69;
30 NEXT a
```

Program 21 - 24.8s

```
 5 POKE 23692,255
10 FOR a=1 TO 1000
20 PRINT 69.25;
30 NEXT a
```

Urobme teraz to isté pre premenné. Programy 22,23 a 24 tlačia reťazcové premenné, celočíselné numerické premenné a neceločíselné numerické premenné.

Program 22 - 10.3s

```
 5 POKE 23692,255
 7 LET b$="69"
10 FOR a=1 TO 1000
20 PRINT b$;
30 NEXT a
```

Program 23-17.9 sec

```
 5 POKE 23692,255
 7 LET b=69
10 FOR a=1 TO 1000
20 PRINT b;
30 NEXT a
```

Program 24 - 25.2s

```
 5 POKE 23692,255
 7 LET b=69.25
10 FOR a=1 TO 1000
20 PRINT b;
30 NEXT a
```

Záver znie: Použite reťazce k tlači všade, kde je to možné. Pokiaľ to má byť číslo, tak radšej celé, tlačí sa ovela rýchlejšie. A teraz sa pozrime z hľadiska tlače na VAL a CODE.

Program 25 - 26.1s

```
 5 POKE 23692,255
10 FOR a=1 TO 1000
20 PRINT VAL "69";
30 NEXT a
```

Program 26 - 18.3s

```
 5 POKE 23692,255
10 FOR a=1 TO 1000
20 PRINT CODE "E";
30 NEXT a
```

Tam, kde chcete uložiť informáciu ako znak v reťazci, je lepšie ju dekódovať pomocou CODE ako VAL. Použitie mien premenných, ktoré sa skladajú z viacerých znakov, je pomalšie než mená jednoznakové.

Program 27 - 6.9s

```
10 FOR a=1 TO 1000
20 LET m=10
30 NEXT a
```

Program 28 - 7.8s

```
10 FOR a=1 TO 1000
20 LET magaziny=10
30 NEXT a
```

Definujte najskôr najviac používané premenné. Čas pre ich nájdenie závisí od toho, ako hlboko musí počítač nahliadať do tabuľky premenných.

Program 29 - 8.4s

```
10 LET b=10
20 LET c=20
30 LET d=4
40 LET e=6
50 LET f=7
60 FOR a=1 TO 1000
70 LET g=b
80 NEXT a
```

Program 30 - 8.8s

```
10 LET b=10
20 LET c=20
30 LET d=4
40 LET e=6
50 LET f=7
60 FOR a=1 TO 1000
70 LET g=f
80 NEXT a
```


Teraz sa pozrime na príkaz REM. I keď je v priebehu programu ignorovaný, interpret jazyka Basic musí však cez neho prísť, a to zaberá určitý čas. Použitie REM tu a tam v programe nevadí, ale ak je REM vo veľkej slučke, potom je to už poznať. Porovnajte program 4 a program 31:

Program 31 - 5.2s

```
10 FOR a=1 TO 1000
20 REM Toto je poznámka za príkazom REM...
30 NEXT a
```

Program 4, prázdna slučka FOR/NEXT trvá asi 4,5 sec, pričom program 31, kam bol zabudovaný prázdny príkaz REM, trvá už 5,2 sec. REM teda zaberá čas. A to platí pre všetko, čo je vo vnútri slučky, ale je to preskakované a mohlo by to byť mimo slučku. Počítač prechodom cez preskakované príkazy stráca čas pri každom prechode slučkou.

Teraz sa pozrime na rôzne možnosti zápisu výrazov, ktoré sa používajú v programe viac krát. Napr. LET r=INT (RND*9)+1. Najskôr ho zapíšeme v plnej forme zakaždým, keď je to potrebné.

Program 32 - 13.6s

```
10 FOR a=1 TO 500
20 LET r=INT (RND*9)+1
30 NEXT a
```

Alebo ho budeme definovať ako funkciu:

Program 33 - 13.9s

```
 5 DEF FN r()=INT(RND*9)+1
10 FOR a=1 TO 500
20 LET r=FN r()
30 NEXT a
```

Alebo zadáme výraz ako podprogram:

Program 34 - 14.9s

```
10 FOR a=1 TO 500
20 GO SUB 50
30 NEXT a
40 STOP
50 LET r=INT (RND*9)+1
60 RETURN
```

Alebo použijeme funkciu VAL k reťazcu obsahujúcemu náš výraz:

Program 35 - 19.6s

```
5 LET a$="INT (RND*9)+1"
10 FOR a=1 TO 500
20 LET a=VAL a$
30 NEXT a
```

Z toho vidíte, že najrýchlejšie je zapísať výraz v plnej forme vždy vtedy a tam, kde je priorita rýchlosti vyššia než úspornosť programu.

A teraz o ATTR a SCREEN$. Ak máte voliť medzi ATTR a SCREEN$ tam, kde je možno použiť obe funkcie, potom ATTR je rýchlejšie ako ukazuje porovnanie programov 36 a 37.

Program 36 - 9.5s

```
10 FOR a=1 TO 1000
20 LET b=ATTR (1,1)
30 NEXT a
```

Program 37 - 12.5s

```
10 FOR a=1 TO 1000
20 LET b$=SCREEN$ (1,1)
30 NEXT a
```

V Y U Ž I T I E S Y S T É M O V Ý C H P R E M E N N Ý C H

Systémové premenné sú bajty pamäti od adresy 23552 do 23732, ktoré poamáhajú počítaču pamätať si rôzne veci, ktoré potrebuje pre svoju činnosť. Tieto informácie sú uchované v systémových premenných na ich adresách a počítač si ich môže vybrať alebo, ak je to potrebné, zmeniť.

My môžeme tieto informácie využiť rôznymi spôsobmi v našich programoch: čítame informáciu, ktorá tu už je, meníme ju, aby sme počítač prinútili niečo robiť, alebo mu zabránili v nejakej činnosti, ktorá nám vadí, alebo aby sme mohli niečo robiť oveľa ľahšie ako normálnou cestou.

Avšak nie všetky systémové premenné môžeme využívať a tiež nie všetky môžeme meniť. Niektoré zmeny môžu spôsobiť, že sa systém počítača zrúti alebo nás bude počítač celkom ignorovať. Niečo môže byť menené len za určitých okolností a väčšina len pri dodržaní prísnych obmedzení. Chceme vám dať návod, ako a kedy možno tu ktorú premennú využiť, ale dokonale sa tie všetky PEEKy a POKEy naučíte až časom.

23552 až 23559 KSTATE - číta klávesnicu

Keď je procesor prerušený vo svojej činnosti (a to je v Európe normálne 50 krát za sekundu, vďaka frekvencii 50 Hz), jedna z vecí, ktorú robí, je, že číta klávesnicu a výsledok tohto čítania ukladá v týchto bajtoch. Tie majú rôzne využitie.

Nie všetko môže byť využité programátorom. Použite nasledujúci program, aby ste videli, čo ide do týchto bajtov KSTATE. Prejdite ho a tlačte rôzne klávesy, aby ste videli aký vplyv majú jednotlivé klávesy (je potrebné klávesu SHIFT) a čo sa robí pri prechode z jednej klávesy na druhú.

```
10 FOR a=23552 TO 23559
20 POKE 23692,0: REM Nekonečný scroll
30 LET b=PEEK a
40 PRINT  a;TAB 10;b;TAB 20;CHR$ b AND b>31
50 NEXT a
60 GO TO 10
```


Prvé štyri bajty KSTATE sa zaoberajú niečím, čo je nazvané "prekrytie dvoch kláves", čo vám umožňuje stlačiť druhú klávesu predtým, než prvú skutočne pustíte. Popis, odovzdávaný hlavným štyrom bajtom, 23556 až 23559, sa teda bude dotýkať prave len jednej z týchto dvoch prekrývaných kláves, a to tej, ktorá bola stlačená prvá. PEEK 23556 nám tak vráti CODE veľkých znakov za stlačené klávesy, takže keď stlačíte SYMBOL SHIFT A, dostanete CODE od "A", nie CODE od "a" alebo CODE od STOP. Toto je užitočné tam, kde je treba realizovať rozhodnutie medzi malými a veľkými písmenami.

Efekt stlačenia klávesy je dočasný a trvá len potiaľ, pokiaľ je klávesa stlačená. Hodnota v 23556 bude 255, ak nebola v okamihu prerušenia procesu žiadna klávesa stlačená. Pre klávesu ENTER tam bude hodnota 13, pre SPACE 32. Stlačenie oboch kláves SHIFT súčasne dá hodnotu 14.

Tento program vám to predvedie:

```
10 LET a=PEEK 23556
20 POKE 23692,0
30 PRINT a,CHR$ a AND a>31
40 GO TO 10
```

23557 slúži pre časovanie k zabráneniu medziklávesového dotyku, známemu ako klávesnicový preskok.

23558 je časovač automatického opakovania, ktorý meria čas prestávky, kým sa začne stlačená klávesa automaticky opakovať a potom pauzu medzi jednotlivými opakovaniami, pokiaľ toto opakovanie už začalo. Určený čas pre toto oneskorenie je v systémovej premennej 23561/2 (viď. ďalej).

23559 obsahuje CODE znaku, ktorý bol na klávesnici stlačený ako posledný. Závisí od toho, či bola stlačená niektorá SHIFT klávesa alebo nie. Toto číslo je rovnaké ako to, ktoré by sme dostali po PRINT CODE INKEY$, len s tým rozdielom, že je to CODE poslednej klávesy, ktorá bola stlačená, nie tej, ktorá je práve teraz stlačená. Skúste tento program, aby ste videli, čo sa robí - tlačte rôzne klávesy i s použitím SHIFT.

```
10 LET a=PEEK 23559
20 POKE 23692,0
30 PRINT a,CHR$ a AND a>31
40 GO TO 10
```

23560 LASTK - novo stlačená klávesa

Zakaždým, keď je prezeraný stav klávesnice a je nájdená novo stlačená platná klávesa, zmení sa hodnota uložená v tejto premennej. Jej obsah je CODE naposledy stlačenej klávesy. S tým sa ale ďalej nedá moc robiť, podobne ako s INKEY$, okrem prípadu, že si chcete napísať znak už "dopredu" (akoby do zásoby). Ak si skúsite nasledujúci program a stlačením klávesy sa pokusíte túto činnosť vyvolať, uvidíte, že znak sa na obrazovke objaví len krátko, pretože celý program pracuje pomalšie než len riadok 50. V tejto premennej je uložený CODE poslednej stlačenej klávesy a to do tej doby, než je stlačená iná. To je potrebné pre test, či stlačená klávesa je novo stlačená, a to skúmaním bitu 5 premennej FLAGS 23611. Ten musí byť pri novostlačenej klávese 1.

```
10 PRINT "Teraz stlač kláves."
20 FOR a=1 TO 900
```

```
30 NEXT a
40 CLS
50 LET a=PEEK 23560
60 PRINT a: IF a>31 THEN PRINT CHR$ a
```

Toto môže byť využité pre test situácie áno/nie, ak viete, čo príde, môžete odpovedať ešte než sa program dostane k tomuto miestu. Teda, ak stlačíte súčasne dve klávesy, program zareaguje už po uvoľnení jednej z nich, bez toho, aby čakal na uvoľnenie celej klávesnice.

Riadiace znaky môžu byť generované pomocou CAPS SHIFT v súčinnosti s klávesami číslic. ENTER vracia 13. Oba SHIFT dohromady dajú 14. Ako ukážku si spustite program:

```
10 LET a=PEEK 23560
20 PRINT a,CHR$ a AND a>31
30 GO TO 10
```

23561 REPDEL - opakovacie oneskorenie

Táto premenná označuje dobu, po ktorú musí byť klávesa stlačená, než začne automatické opakovanie. Oneskorenie je v Európe v pädesiatinách sekundy a je na začiatku nastavené na 35/50. Oneskorenie možno zmeniť pomocou POKE, ak chcete napr., aby opakovanie začalo okamžite. Potom budete ale ťažko riadiť kurzor, ak udáte trebárs POKE 23561,1. POKE 23561,0 zdanlivo opakovanie vypína, v skutočnosti dáva oneskorenie okolo sekundy, rovnako ako POKE 23561,255.

23562 REPPER - oneskorenie medzi opakovaním

Táto premenná riadi dĺžku doby medzi opakovaním, pokiaľ už toto opakovanie začalo. V Európe je tento čas v 1/50 sec. Ak chcete zdanlivo tento čas zrušiť, urobte POKE 23562,0 alebo POKE 23562,255, čím dostanete oneskorenie asi 5 sec. Ak potrebujete editovať dlhé programové riadky (napr. dlhé PRINT príkazy), použite POKE 23562,1, čím urýchlite pohyb kurzoru na potrebné miesto. Ale pozor, nemeňte súčasne príliš čas v 23561, pretože potom sa vám bude kurzor zle ovládať. Normálna hodnota 23562 je 5/50 sekundy, teda 1/10 sekundy.

23563/23564 DEFADD

Adresa argumentu užívateľom definovanej funkcie v programe, tj. ak máte v programe riadok ako: DEF FN a(b), potom hodnota v 23563/4 bude adresa písmena b v zátvorkách, avšak len v tom momente, keď je funkcia použitá. Najlepšia cesta k tomu, aby ste videli obsah tejto premennej, je vložiť PEEK 23563/4 ako časť FN, pretože pokiaľ nie je FN v činnosti, je v 23563/4 stále 0. Tak riadok

```
10 PRINT PEEK 23564+256*PEEK 23564
```

vráti vždy 0. Avšak na druhej strane

```
10 DEF FN a(b)=PEEK 23563+256*PEEK 23564
20 PRINT FN a(999)
```

vydá adresu znaku b, 999 nie je významné, iba ako číslo, aby v b bola

nejaká hodnota pre zamedzenie chyby. V prípade funkcie bez argumentu, ako

```
10 DEF FN a()=PEEK 23563+256*PEEK 23564
20 PRINT FN a()
```

dá program adresu uzatvárajúcej zátvorky ()).

23568 až 23605 STRMS

Prvých 14 bytov na základnom Spectre obsahuje adresy ukazujúce na kanály a prúdy. Prúdy -3 až +3 sú uložené každý v dvoch bajtoch.

23606/23607 CHARS
Tieto premenné majú normálne hodnoty:

23606 obsahuje 0
23607 obsahuje 60

Táto systémová premenná ukazuje na začiatok sady znakov, ktoré počítač používa pre výpis na obrazovke alebo na tlačiarni. Tiež funkcia SCREEN$ používa túto premennú. Adresa normálne uložená v tejto premennej je 15360, čo je o 256 menej ako je počiatočná adresa generátora znakov v ROM. O 256 menej z toho dôvodu, že generátor znakov je dostupný podobným procesom ako je PEEK 23606+256*PEEK 23607+CODE"A"*8 a vzhľadom na to, že prvý znak v ROM je SPACE - medzera.

CODE znaku SPACE je 32 a teda 8*32=256. Generátor znakov je 768 bajtov dlhý, takže ak chcete nastaviť novú sadu znakov, potom to musí byť mimo tento počet bajtov.

Bola zmienka o tom, že SCREEN$ využíva túto systémovú premennú. To je pravda, ale v prípade, že budete chcieť identifikovať UDG pomocou SCREEN$, potom normálnou cestou nepochodíte. A to preto, že SCREEN$ dokáže normálne prehľadávať len generátor znakov v ROM. My ale môžeme dočasne zmeniť ukazovateľ začiatku generátora znakov na začiatok UDG a tam potom hľadať znak, ktorý sa má identifikovať. Ešte ale musíme urobiť jednu vec: existuje systémová premenná, ktorá ukazuje na začiatok UDG, a to presne, takže musíme odčítať 256. To znamená, odčítame 1 od vyššieho bajtu.

Nasledujúci program to predvádza:

```
10 FOR x=144 TO 164
20 PRINT AT 0,0;CHR$ x
30 POKE 23606,PEEK 23675
40 POKE 23607,PEEK 23676-1
50 PRINT AT 20,0;SCREEN$ (0,0)
60 PAUSE 40
70 POKE 23606,0
80 POKE 23607,60
90 NEXT x
```

Tým, čo sme urobili, sme prinútili počítač, aby si myslel, že UDG je normálny generátor znakov. SCREEN$ bude ale stále produkovať len znaky s CODE 32-127, čo ale ľahko obídeme. Pretože SCREEN$ začína s CHR$ 32 a UDG začíname na 144, tak potrebujeme pridať 112, aby sme dostali znaky v rozsahu UDG. Tu je jedna cesta, ako to urobiť.

X a y sú normálne súradnice znakov, ktoré máme pomocou SCREEN$

preskúmať. Kontrola je to prvé, čo je treba urobiť, pretože SCREEN$ nemôže teraz rozlišovať normálny znak. Znak zo súradníc y,x bude v a$. Riadok 8025 je treba len vtedy, ak používate inú sadu znakov než je to v ROM. Ak používate sadu v ROM, potom odstráňte riadok 8025 a riadky 8070 a 8080 nahraďte tými, ktoré sú uvedené za programom.

```
8000 REM SCREEN$ pre užívateľom definovanú grafiku
8010 LET a$=SCREEN$ (y,x)
8020 IF a$<>"" THEN RETURN
8025 LET a=PEEK 23606: LET b=PEEK 23607
8030 POKE 23606,PEEK 23675
8040 POKE 23607,PEEK 23676-1
8050 LET a$=SCREEN$ (y,x)
8060 IF a$<>"" THEN LET a$=CHR$ (CODE a$+112)
8070 POKE 23606,a
8080 POKE 23607,b
8090 RETURN

8070 POKE 23606,0
8080 POKE 23607,60
```

Príbeh však tu nekončí. UDG má len 21 znakov. Pokiaľ SCREEN$ medzi nimi nenašiel ten hľadaný, bude pokračovať v hľadaní ďalej. A za UDG môžu byť z najrôznejších dôvodov ďalšie znaky, takže by SCREEN$ mohol nájsť i nejaký nezmysel.

Aby sme tomu zabránili, pridáme riadok:

```
8065 IF a$>CHR$ 164 THEN LET a$=""
```

Pri použití PRINT, LIST atď. však si musíme byť istý, že ukazovateľ uložený v 23606/7 ukazuje na začiatok správnej sady znakov. Ak nenastavíme totiž ukazovateľ späť na správnu sadu, môžeme získať výpis, ktorý nie je dobre čitateľný.

23608 RASP

Riadenie dĺžky bzučania, ktoré sa ozve po naplnení pamäte. Po zapnutí počítača má hodnotu 64. Možno ju POKEom zmeniť. POKE 23608,0 dá len veľmi krátky zvuk, skôr klepnutie ako bzukot. Oproti tomu POKE 23608,255 dá veľmi dlhé bzučanie, pričom sa i zablokuje klávesnica, čo nám zabráni v ďalšom zápise.

23609 PIP

Riadenie dĺžky zvuku, ktorý sa ozve pri stlačení klávesy v režime príkazov, alebo pri INPUT. Počiatočná hodnota je 0, ale dá sa meniť. Ktorákoľvek hodnota medzi 30 a 130 dáva príjemné pípnutie, ktoré je lepšie počuť. Hodnoty vyššie ako 130 už spomaľujú vašu činnosť, pretože v priebehu pípnutia sa zastavuje činnosť počítača. Optimálna hodnota je 100.

23610 ERR NR

Riadi číslo chybovej správy a pokiaľ sa nič nedeje, obsahuje 255. V prípade chyby potom obsahuje číslo, ktoré je o 1 menšie ako číslo vypísanej chybnej správy. Chybová správa je uložená v ROM počínajuc

adresou 5010. Koniec správy je indikovaný v jej poslednom znaku, ktorý má bit 7 nastavený na 1. Za chybovými správami nasleduje v ROM (c) 1982 Sinclair Research Ltd., ktorá sa objavuje po zapnutí alebo po NEW. Pomocou POKE 23610 si môžete generovať chybové zakončenie programu, ale pretože chybová správa je pevne daná v ROM, budete na pochybách, či je to skutočná chyba alebo vaše programové zakončenie.

23611 FLAGS

Táto systémová premenná obsahuje rôzne indikátory, ktoré riadi celý systém a obecne nemôžu byť menené pomocou POKE. Niektoré z nich však môžeme využiť s pomocou PEEK.

BIT 0: 1 oznamuje, že pred nasledujúcim kľúčovým slovom nebude tlačená medzera

BIT 1: 1 znamená, že výpis bude na tlačiareň, 0 znamená, že výpis bude na obrazovku

BIT 5: Každá novo stlačená klávesa má uložené svoje CODE v 23560 (LASTK) a bit 5 23611 je nastavený na 1, čo signalizuje, že bola stlačená nová klávesa.

BIT 7: indikátor syntaxe. Táto systémová premenná má oveľa väčšie využitie pri používaní rutín ROM v strojovom kóde než pri programovaní v BASICu.

23613/23614 ERR-SP

Uchováva adresu stopy strojového zásobníka, kde leží návratová adresa. Skúste zavolať niekoľko GOSUB bez RETURN a pozorujte, ako sa zmenšuje obsah pamäte. Teraz môžete vidieť, čo a kedy sa stane, keď ste v situácii, že máte nedostatok pamäte (out of memory). Skúste PEEK troch adries, ich základ je v 23613 a 23614, aby ste videli, z akých dát pozostáva skutočný návrat (return).

```
10 LET a=PEEK 23613+256*PEEK 23614
20 PRINT PEEK a;TAB 10;PEEK (a+1);TAB 20;PEEK (a+2)
30 PRINT a
```

23617 MODE

Určuje, ako bude vyzerať kurzor. Je 0, 1,2 alebo 4 pre L/C mode, E mode, G mode alebo K mode. POKE do tejto premennej ovplyvní vzhľad kurzora. Môže to byť číslica, písmeno alebo ľubovoľný znak z klávesnice. To je veľmi vhodné hlavne pri INPUT. Hodnota sa vráti do normálneho stavu, akonáhle vznikne potreba, tak napr. pri normálnej zmene módu z klávesnice. Takže ak sa dostanete do ťažkostí, stlačte dvakrát oba SHIFT klávesy súčasne a dostanete normálny L/C mód. Skúste nasledujúci program, ktorý robí POKE všetkých možných hodnôt do 23617. Väčšina sú to varianty štyroch kurzorov používaných bežne. 255 dá v móde L/C blikajúce <, ktoré vám ukazuje, kde práve ste.

```
10 FOR a=0 TO 255
20 PRINT a
```

```
30 POKE 23617,a
40 INPUT a$
50 NEXT a
```

23618/9 NEWPPC a 23620 NSPPC

23618/9 NEWPPC je dvojbajtová systémová premenná, ktorá obsahuje číslo riadku, na ktorý sa má skákať. 23618 obsahuje nižší bajt čísla riadku a 23619 vyšší bajt, takže obsiahnuté číslo riadku sa číta ako PEEK 23618+256*PEEK 23619. Ako urobiť POKE, povedzme, riadok číslo x:

```
POKE 23618,x-256*INT (x/256)
POKE 23619,INT (x/256)
```

Teraz sa dostávame k systémovej premennej 23620. Pomocou 23618, 23619 a 23620 môžeme simulovať GO TO na určitý príkaz vo vnútri programového riadku, čo môže byť vždy potrebné. GO TO nemôže normálne smerovať k určitému príkazu v dlhom programovom riadku, ale vždy len na začiatok riadku. Pre zaistenie napr. skoku na príkaz č. 4 na riadku x budeme postupovať tak, ako bolo popísané vyššie a potom POKE 23620,4 a skok je prevedený.

```
10 STOP
50 PRINT "Ukážka č.1":PRINT "Ukážka č.2": STOP

POKE 23618,50 : POKE 23619,0 : POKE 23620,2
```

23624 BORDER

Jednotlivé bity tejto systémovej premennej ovládajú atribúty dolnej časti obrazovky a farbu BORDER tak, ako ukazuje tabuľka:

bit

7	6	5 4 3	2 1 0
FLASH dolná časť obrazovky	BRIGHT dolná časť obrazovky	BORDER a dolná časť obrazovky PAPER	dolná časť obrazovky INK

Ak budete pomocou príkazu POKE vkladať do tejto premennej rôzne hodnoty, môžete dosiahnuť napr. blikajúcu, jasnú, mnohofarebnú dolnú časť obrazovky alebo urobiť rovnaký PAPER a INK, aby iní ľudia nemohli opísať váš program - akákoľvek zmena zostane bez úžitku. Alebo môžete INPUT urobiť zvlášť jasný, aby vynikol.

23629/23630 DEST

Adresa premennej, ktorá bola označená. Ak premenná bola nastavená už skôr, potom táto systémová premenná ukazuje na miesto, odkiaľ je označená premenná uložená v tabuľke premenných. Ak bola premenná definovaná poprvý

raz, potom systémová premenná ukazuje adresu, kde začína názov tejto premennej v programe, tj. 10 LET a=5 - systémová premenná obsahuje adresu písmena a. Táto systemová premenná môže byť využitá k nájdeniu adresy v pamäti uloženia numerickej premennej, ak použijete LET a=a, ako napr.:

```
10 LET a=5
20 LET a=a
30 PRINT PEEK 23629+256*PEEK 23630
```

23631/2 CHANS

Uchováva adresu, kde začína priestor kanálových informácií.

23633/4 CHURCHIL

Adresa vstupných/výstupných (INPUT/OUTPUT) informácií, práve využívaných. Normálne ukazuje v priebehu nejakej výstupnej/vstupnej operácie na 5 bajtový blok dát v priestore kanálových informácií. Použite nasledujúci program k prevedeniu obsahu 23633/4.

```
 1 FOR x=0 TO 3: PRINT #1;PEEK 23633+256*PEEK 23634: NEXT x: PAUSE 0:
   STOP
```

23627/23628 VARS

Ukazuje na začiatok miesta, kde sú v pamäti uložené premenné. Bez ohľadu na to, že môžete s pomocou tejto systémovej premennej nájsť cestu do oblasti premenných, môžete pomocou nasledujúceho výrazu zistiť dĺžku programu v BASICu (tj. okrem obrazovky, systémových premenných, zásobníka a premenných).

```
LET bajty=PEEK 23627+256*PEEK 23628-PEEK 23635-256*PEEK 23636
```

23635/23636 PROG

Adresa začiatku priestora v pamäti, odkiaľ je uložený program v Basicu. Ukazuje na prvý bajt čísla prvého riadku programu. Môže to byť užitočné pri konverzii programov na iné počítače Sinclair i s informáciou uchovanou v REM príkaze v prvom riadku programu. Viď. tiež VARS vyššie.

Ak chcete "bezpečne zablokovať" nejaký riadok programu, potom v zmysle tejto systémovej premennej urobte POKE 0 do oboch bajtov čísla riadku, na začiatku programu. Programové riadky začínajú dvojbajtovým číslom.

23637/23638 NXTLIN

Adresa začiatku ďalšieho programového riadku. Môžete ju využiť k dosiahnutiu strojového kódu uloženého niekde v programe v príkaze REM, prípadne k nahratiu knižnice podprogramov z pásky pomocou MERGE. Volanie strojového kódu môže byť napr.:

```
9000 LET a=USR (PEEK 23637+256*PEEK 23638+5)
9010 REM Strojový kód
9020 RETURN
```

Je tu jedno obmedzenie, a to, že do REM nemôžete dať žiadne riadiace znaky pre farbu, blikanie, jas apod., pretože by mohli byť interpretované ako strojový kód, čo je chyba. Avšak v prípade knižnice podprogramov to môže byť použité ľubovoľne.

23639/23640 DATADD

Obsahuje adresu čiarky, ukončujúcu poslednú položku v DATA príkaze. Ak nebolo zo zostavy nič čítané (napr. po RUN, RESTORE apod.), potom je v 23639/40 držaná adresa bajtu pred programovým priestorom, normálne CHR$ 128 na konci priestoru kanálových informácií.

Pre demonštráciu spustite program:

```
10 DATA "1","2","3","4","5"
20 LET a=PEEK 23639+256*PEEK 23640
30 PRINT a;TAB 9;PEEK a;TAB 18;CHR$ PEEK a AND PEEK a>31
40 READ b$
50 GO TO 20
```

```
23754     126
23763     44    ,
23767     44    ,
23771     44    ,
23775     44    ,
23779     13
```

Adresa v tejto dvojbajtovej premennej môže ukazovať na znak ENTER alebo dvojbodka, označujúca koniec riadku alebo príkazu, obsahujúceho DATA - adresu koncového znaku posledného prvku dát.

23641/2 E LINE

Táto systémová premenná ukazuje na začiatok priestoru nad premennými. Tu môžeme získať poňatie o tom, koľko pamäte v bajtoch zaberá obrazovka, systémové premenné, program a premenné, jednak po RUN, jednak po nastavení premenných atď. Napíšte ako priamy príkaz:

```
PRINT PEEK 23641+256*PEEK 23642-16384
```

Môžeme tiež zistiť, koľko zaberajú premenné, či bol program spustený po ich nastavení. Priamy príkaz:

```
PRINT PEEK 23641+256*PEEK 23642-PEEK 23627-256*PEEK 23628
```

23653/23654 STKEND

Táto systémová premenná obsahuje adresu, kde začína voľná pamäť. Čítaním tejto systémovej premennej získame poňatie o tom, koľko voľnej pamäte nám zostáva, keď túto prečítanú odčítame od RAMTOP. Nebude obsahovať pamäť použitú pre zásobník strojového kódu a GOSUB, ale bude obsahovať dĺžku príkazu PEEK. Je to teda pomerne presný návod, ktorý je platný vo väčšine okolností.

```
PRINT PEEK 23730+256*PEEK 23731-PEEK 23653-256*PEEK 23654
```


23658 FLAGS 2

Táto systémová premenná obsahuje niektoré indikátory, využívané (normálne) počítačom k indikácii určitých stavov. My môžeme mať najväčší úžitok z využitia stavu indikovaného bitom 3. Tento, ak je 1, ukazuje, že je zapnutý CAPS LOCK (teda veľké písmená). Aký to má úžitok? Povedzme, že máme v programe INKEY$. Operátor má odpovedať hoci A ako ÁNO a N ako NIE. Väčšine ľudí je to jedno, ak použijú pre ÁNO veľké A alebo malé a. Počítaču to ale jedno nie je. Ak očakáva pre ÁNO ako odpoveď A, potom a nie je správna odpoveď. Aby sme túto situáciu vyriešili, musíme program kompletovať (IF INKEY$=A" OR "a" THEN...). Jednoduchšie je urobiť príslušný POKE, ktorým automaticky zapneme veľké písmená a je to.

Tak POKE 23658,8 zapína CAPS LOCK a POKE 23658,0 CAPS LOCK vypína. Toto ale môže ovplyvniť ostatné indikátory, a preto sa pred POKE musíme presvedčiť, či v 23658 nebola už nejaká dielčia hodnota a tú potom prípadne zväčšiť (pri zapnutí) alebo zmenšiť (pri vypnutí) o 8. Normálne v móde L je v 23658 hodnota 0, takže je všetko OK pre POKE, ako bolo uvedené vyššie. Ak urobíme POKE bez ohľadu na okolnosti, systém sa síce nezrúti, ale môžu nastať zaujímavé efekty. Ak je vyrovnávacia pamäť tlačiarne prázdna, potom bit 1 je 0.

23659 DF SZ

Táto systémová premenná obsahuje počet riadkov dolnej časti obrazovky, normálne využívanej pre INPUT, chybové hlásenie atď. Normálne by to malo byť 2, okrem situácie, keď je zobrazený dlhý INPUT apod. Ak urobíme POKE 0 do tejto systémovej premennej, aby sme mohli využívať všetkých 24 riadkov, tak sa systém zrúti. Za určitých obmedzení sa to však dá urobiť. Toto obmedzenie spočíva v tom, že než opäť využijeme dolnú časť obrazovky pre INPUT alebo chybové hlásenia, musíme všetko vrátiť do pôvodného, normálneho stavu.

BREAK programu by spôsobil katastrofu. Práve tak i chyba, ktorá vznikne v priebehu programu, pretože sa nemá kde objaviť. Nasleduje krátky program, ukazujúci využitie riadkov 22 a 23. Bohužiaľ možno na tie riadky robiť len PRINT a PRINT TAB, nie však PLOT. Stav obrazovky sa potom vracia do normálneho stavu pomocou POKE 23659,2 vo vnútri vlastného programu.

```
10 POKE 23659,0
20 FOR a=0 TO 23
30 PRINT a
40 NEXT a
50 PAUSE 0
60 POKE 23659,2
```

Aby sme videli, ako môže všetko dopadnúť zle, pridáme k programu riadok:

```
45 PRINT chyba
```

Ojoj! Ak chcete len PRINT na spodné dva riadky, potom je lepšie použiť PRINT # 1;"test", čo pracuje rovnako dobre, ak nie ešte lepšie, bez nebezpečia, že sa systém zrúti. Ak urobíte do DF SZ POKE hodnotu väčšiu než 2, potom horná časť obrazovky bude menšia. Tak po POKE 23659,y bude horná časť obrazovky len 24-y riadkov. Nasledujúci program ukazuje, ako možno riadiť rolovanie časti obrazovky za pomoci DF SZ a SCR CT. Objavujú

sa náhodné čísla a robí sa scroll len horných 14 riadkov obrazovky.

```
10 POKE 23692,0: POKE 23659,10
20 PRINT RND
30 GO TO 10
```

23670/1 SEED

Ak je použité RANDOMIZE (číslo), potom je do tejto systémovej premennej uložené číslo (konštanta alebo premenná). Toto číslo určuje nasledujúce náhodné číslo. Otvára možnosť získania ďalšieho (predpokladaného) náhodného čísla s tým, že využívate svojich znalostí k obráteniu šťastia na svoju stranu. Napr. po RANDOMIZE 1 bude nasledujúca hodnota RND 0,0022735596. Alebo INT (RND*6)+1 bude 1.

23672/3/4 FRAMES

Toto je čítač televíznych snímkov, ktorý môže byť použitý ako časovač. Zvyšuje svoj obsah 50x za sec, teda každých 0,02 sec. Čas získaný čítaním a výpočtom z týchto troch bytov však nie je presný. Manuál Spectrum na to upozorňuje vo svojej kapitole 18. Doporučujeme čítať tento čas 2 krát za sebou a vziať väčšiu hodnotu. Počas čítania (PEEK) a výpočtu čítač totiž totiž stále počíta. Jeho celková kapacita je 3 dni a 21 3/4 hod.

Je nutné upozorniť, že 3 bajty sú v opačnom poradí hodnôt: najvýznamnejší bajt je 23674, teda ich hodnota bude:

```
65536*PEEK 23674+256*PEEK 23673+PEEK 23672
```

čo nám udá čas v pedesiatinách sec. Je tu ešte niekoľko vecí, ktoré ovplyvňujú presnosť časovača. BEEP zastavuje jeho činnosť. LOAD a SAVE majú tiež vplyv na presnosť. Len PAUSE čaká zadaný čas, bez toho aby sa časovač zastavil.

23675/6 UDG

Adresa začiatku UDG je normálne na 16K Spectre 32600 a na 48K Spectre 65368. Toto číslo je rovnaké ako USR "a", takže PRINT USR "a" zodpovedá

```
PRINT PEEK 23675+256*PEEK 23676
```

POKEri z donútenia si s tým užijú zábavy. Manuál totiž doporučuje zmeniť hodnotu v týchto systémových premenných pre úsporu miesta v pamäti, keď máte viac sád UDG. Je však nutné sa na to pozrieť i obrátene: nastaviť viac sád UDG zaberie miesto v pamäti, pretože v činnosti môže byť len jedna sada súčasne. Pamätajte, že existuje 21 znakov UDG, takže si musíte v pamäti rezervovať 21*8 alebo 168 bytov pre každú ďalšiu sadu UDG a potom vložiť príkazom POKE jej počiatočnú adresu do 23675/6 až v okamihu, keď ju bude využívať.

Pre zábavu skúste:

```
POKE 23675,96: POKE 23676,127 (16K Spectrum)
POKE 23675,96: POKE 23676,255 (48K Spectrum)
```

Potom skúste pomocou UDG (čo sú pred nadefinovaním veľké písmená)

napísať nejaku správu.

Užitočný tip: Akonáhle ste si vytvorili svoju sadu UDG, urobte si SAVE. Väčšina ľudí napíše niečo ako:

```
SAVE "znaky" CODE 32600,168
```

Fajn, ale máte zadať počiatočnú adresu. Môžete použiť SAVE "znaky" CODE (PEEK 23675+256* PEEK 23676),168 a tým dostanete na pásku práve používanú sadu UDG, bez toho aby ste vedeli, kde začína. To vám umožní LOAD UDG, ktoré bolo zhrané príkazom SAVE na 16K Spectre do 48K Spectra. Aby ste dostali UDG na správne miesto na počítači s odlišným rozsahom pamäte, použite jednoducho LOAD "znaky" CODE (PEEK 223675+256*PEEK 23676),168. To vám automaticky uloží UDG na správne miesto. Je to rovnaké, ako keď napíšete:

```
LOAD "znaky" CODE USR "a",
```

čo ušetrí trochu ťukania do klávesnice, i keď to vyzerá zvláštne.

23679 P POSN

Obsahuje informáciu o tom, ako ďaleko ste pri LPRINT vo vyrovnávacej pamäti tlačiarne. Obsahuje (33-číslo stĺpca) pre stĺpce 0 až 31. LPRINT pozíciu nemôžete zmeniť pomocou POKE.

23680 PROC

Obsahuje spodný bajt adresy, do ktorej príde ďalší znak pre tlačiareň. Je to 23296 + LPRINT číslo stĺpca (čo je 0 až 31). Pretože je to adresa horného radu bodov každého znaku, môžete do nej zasahovať pomocou POKE, aby ste zmenili LPRINT pozíciu vyrovnávacej pamäte, čo vám umožní zmeniť hodnotu v P POSN (23679).

Môže sa zdať, že to pracuje, i keď nechcete, ale problémy sa zrušia na konci riadku.

23681 UNUSED SYSTEM VARIABLE

Táto systémová premenná, striktne nazývaná "nepoužitá", v skutočnosti obsahuje 91. Je to horné byte adresy LPRINT vyrovnávacej pamäte (91* 256 = 23296, kde vyrovnávacia pamäť začína). Môžete sem vkladať pomocou POKE pre vlastné použitie, ale akékoľvek použitie tlačiarne sem vráti hodnotu 91. Premenná 23680/1 dohromady obsahuje adresu LPRINT pozície vo vyrovnávacej pamäti. Nemôžete ovplyvniť činnosť tlačiarne pomocou POKE 23681, ale čokoľvek sem vložené môže byť prepísané pri tlačiarenskej rutine.

23677/8 COORDS

23677 obsahuje x súradnicu posledného nakresleného bodu. Po CLS začína na hodnote 0. Premenná 23678 obsahuje y súradnicu posledného bodu. Obe premenné obsahujú skutočnú hodnotu, takže ak bol posledný bod nakreslený na súradniciach 3,3, obsahujú obe premenné hodnotu 3.

Do oboch môžete vkladať príkazom POKE platné hodnoty súradníc x, resp. y. I keď tento POKE neurobí v skutočnosti žiadny PLOT na obrazovke, je to vhodná cesta pre pohyb PLOT kurzora. Mohlo by sa to urobiť i pomocou PLOT OVER 1;x,y, to však nie je "čisté". Medzi iným je možné pomocou POKE takto stimulovať príkaz MOVE, ktorý používajú iné verzie BASICu. Je to

užitočné, ak chcte kresliť čiaru od zvláštneho bodu.

23684/5 DF CC

Adresa PRINT pozície v pamäti obrazovky. Môžete meniť príkazom POKE, aby ste pohli s PRINT pozíciu niekam inam, ale predpokladom je znalosť organizácie obrazovej pamäti.

23688/9 SPOSN

23688 obsauje informáciu o tom, ako ďaleko naprieč obrazovkou došla PRINT pozícia. Začína na hodnote 33 pre ľavý okraj obrazovky a s každou pozíciou vpravo sa znižuje o 1. Po použití PRINT AT y,x (za predpokladu, že x i y sú platné) bude 23688 obsahovať 33-x. Toto môže byť užitočné ak chceme zabrániť rozdeleniu slova na konci riadku. Ak uvážite, že hodnota v 23688 sa s postupom k pravému okraju obrazovky znižuje k 0 (že teda ubúda miesta pre ďalšie slovo), môžete použiť porovnanie hodnoty v 23688 s dĺžkou slova, ktoré chcete napísať. K tomu, aby ste zistili, či sa na obrazovku ešte vojde celé alebo nie. Ak chcete zabrániť rozdeleniu, potom toto slovo dáte na nový riadok. Povedzme, že slovo k tlači je vo w$:

IF PEEK 23688<LEN w$+1 THEN PRINT

To však funguje len pre slovo kratšie ako 32 znakov.

23689 obsahuje informáciu o tom, ako ďaleko sa dostala PRINT pozícia dole obrazovkou. Začína na 24 pre horný riadok a znižuje sa o 1 s každým posunom PRINT dole. Ak chcete radšej CLS než rolovanie v momente, kedy sa s PRINTom dostanete blízko k dolnému koncu obrazovky, potom skúste:

IF PEEK 23689=3 THEN CLS

23692 SCR CT

Obsahuje údaj o tom, koľko rolovaní sa prevedie dovtedy, kým sa rolovanie zastaví s otázkou scroll? Tak napr. POKE 23692,255 dá 255 tlačových riadkov, než sa objaví scroll? POKE 23692,0 pôsobí podobne, ale máte ešte jeden tlačový riadok k dobru. Ak potrebujete zabrániť scroll? na viacej riadkov, potom ak máte PRINT príkaz v slučke, musí táto slučka obsahovať aj POKE.

23693 ATTR P

Obsahuje trvalé atribúty, tj. FLASH, BRIGHT, PAPER a INK, ktoré pôsobia celkovo a počas celého programu. Miestne farby v príkazoch PRINT môžu byť rozdeľované kdekoľvek. Je treba poznamenať, že väčšina ROM zatiaľ využíva hodnôt systémovej premennej, ktorá uchováva dočasné atribúty, ktoré sú rovnaké ako trvalé, pokiaľ nebol zmenený nejaký miestny parameter. CLS nastaví obrazovku, ale podľa hodnôt v ATTR P. Funkcie jednotlivých bitov sú uvedené v nasledujúcej tabuľke:

bit 7	bit 6	bit 5 bit 4 bit 3	bit 2 bit 1 bit 0
FLASH	BRIGHT	farba PAPER binárne	farba INK binárne

Bit 7=1 pre FLASH 1.
Bit 7=0 pre FLASH 0.
Bit 6=0 pre BRIGHT 0.
Bity 5, 4 a 3 obsahujú farbu PAPER binárne, napr. pre PAPER 7 bity 5,4 a 3=111.
Bity 2, 1 a 0 obsahujú farbu INK binárne, tj. pre INK 3, bity 2, 1 a 0=011.
Atribúty 8 a 9 nie sú tu prideľované. Ak trvalé atribúty sú 8 a 9, potom ak sú uložené v ATTR P, nebudú platné.

23694 MASK P

Táto systémová premenná pomáha Spectru určiť atribúty tlače, ak bol zadaný parameter 8. Takže ak zadáte celkovo BRIGHT 8, bude bit 6 v 23694 nastavený na 1, čím bude v budúcnosti počítaču pripomínať, že bolo určené BRIGHT 8: Počítač potom pri tlači vždy nahliadne do tejto premennej a zmení príslušný atribút. Alebo presnejšie povedané, zmení starý znak za nový a atribút nechá tak, ako bol.

Význam bitov:

bit 7	bit 6	bit 5 bit 4 bit 3	bit 2	bit 1	bit 0
FLASH 8	BRIGHT 8	PAPE 8?	INK 8?		

Tam, kde sa berie do úvahy viac než jeden bit, ako INK a PAPER, iba nastavené bity (v 1) berú svoje atribúty podľa obrazovky.

To môže viesť k neželaným efektom. Skúste:

```
10 INK 8
20 POKE 23694,BIN 00000011
30 PRINT AT 0,0; INK 5;"5555555"
40 PRINT AT 0,0;"1111"
```

Ak je zadané INK 8, potom by ste očakávali, že jednotky budú tlačené v rovnakej farbe ako päťky (v našom programe svetlomodrá), ale chyba! Namiesto aby počítač kontroloval, celý INK atribút, kontroluje iba, ktorý bit v 23694 je 1 a ktorý nie. Pozrime sa, či zistíte farbu, v ktorej sa vytlačia jednotky. Príjemnú zábavu!

23695 ATTR T

Táto systémová premenná obsahuje atribúty dočasných farieb tak, ako boli miestne zadané vo vnútri príkazov PRINT. Môžete to vidieť napr. s pomocou týchto dvoch priamych príkazov:

```
PRINT PEEK 23695
PRINT INK 7; PAPER 0;PEEK 23695
```

Toto je vrátane PEEKu v príkaze PRINT pod vplyvom miestneho riadenia farieb. Normálne, pokiaľ nebol zadaný nejaký miestny atribút, bude táto systémová premenná obsahovať atribúty trvalé, celkové. Farba a ostatné atribúty, ktoré budú použité v mieste tlače, sa berú z tejto premennej (23695), a to iba tie atribúty, ktoré sú rozdielne od atribútov trvalých, tj. uložených v 23693 (ATTR P).

Funkcie jednotlivých bitov:

bit 7	bit 6	bit 5 bit 4 bit 3	bit 2 bit 1 bit 0
dočasný FLASH	dočasný BRIGHT	dočasný PAPER	dočasný INK

23696 MASK T

Pracuje podobne ako MASK P (23694) s tým rozdielom, že parametre tu uložené sú dočasné. Normálne obsahuje to isté čo MASK P, ak je však nastavená miestna farba apod., obsah MASK T sa zodpovedajúcim spôsobom zmení.

Možno to študovať pomocou programu:

```
PRINT PEEK 23696, INK 8;PEEK 23696, INK 0;PEEK 23696, FLASH 8
```

Jednotlivé bity majú tieto funkcie:

bit 7	bit 6	bit 5 bit 4 bit 3	bit 2 bit 1 bit 0
dočasný FLASH 8?	dočasný BRIGHT 8?	dočasný PAPER 8?	dočasný INK 8?

23697 P FLAG

Táto systémová premenná obsahuje ako je zrejmé z názvu, ukazovatele slúžiace počas tlače. Po určení PAPER 9 bude nastavený na 1 bit 6 a 7. Po INK 9 bude 1 bit 4 a 5. Po INVERSE 1 bude 1 bit 2 a 3. A po OVER 1 bude 1

bit 0 a 1. Účinok bude celkový, ak je nastavený na jeden príslušný neparný bit (tj. 1, 3, 5 a 7) a miestny, ak je 1 v párnom bite.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
celkový PAPER 9	dočasný PAPER 9	celkový INK 9	dočasný INK 9	celkový INVERSE 1	dočasný INVERSE 1	celkový OVER 1	dočasný OVER 1

23681,23728/9

Tieto tri bajty systémových premenných nie sú normálne na Spectre využité. Môžete si z nich urobiť "zákaznícke premenné" a použiť ich vo vašich programoch pre uloženie najrozličnejších informácií (napr. premennej). Prístup na pevnú adresu je totiž rýchlejší než prehľadávajúcu tabuľky premenných, 23728/9 boli uvažované pre nemaskované prerušenie, to však nie je na našom Spectre použité.

23730/1 RAMTOP

Táto dvojbajtová systémová premenná udáva adresu posledného bajtu RAM v priestore určenom pre BASIC. Nie je to však koniec pamäte, ktorú môže BASIC využiť, pretože za RAMTOP je ešte UDG. Ak posunieme RAMTOP nahor do časti pre UDG, túto síce prepíšeme, ale zato získame niekoľko bytov, ktoré sa môžu hodiť, zvlášť na 16K Spectre.

Jedna dôležitá vec je, že NEW pracuje iba po adresu, ktorá je v RAMTOP, takže za ňou môžeme uschovať dáta alebo strojový program a pod., ktoré majú vplynúť medzi nahrávané programy alebo ich chceme len ochrániť pred vymazaním.

23732/33 PRAMT

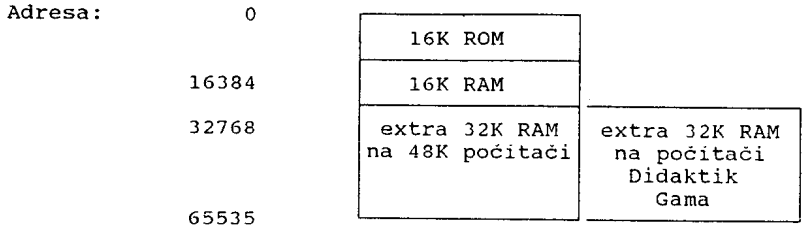
Obsahuje adresu, kde končí RAM na Spectre. Ak sa dostanete k počítaču, o ktorom neviete, koľko má zabudovanej pamäte, nie je nutné ho otvárať a pozerať do vnútra, ale stačí len

```
PRINT PEEK 23732+256*PEEK 23733-16384
```

16384 sa odčítava preto, aby sme z kapacity pamäte RAM vylúčili ROM. RAM začína na adrese 16384 a ide po adresu, ktorá je uložená v PRAMT.

O B S A D E N I E P A M Ä T E

Spectrum normálne pozostáva zo 16KB ROM a 16KB alebo 48KB RAM, viď. nasledujúca tabuľka:



Didaktik Gama, s pamäťou 80 kB má identické rozmiestenie pamäte ako ZX Spectrum, len má navyše jednu banku s 32 kB RAM, ktorá leží paralelne s 32 kB RAM od adresy 32768 do 65535. Súčasný prístup do oboch baniek nie je možný, vždy sa vyberie jedna z nich pomocou OUT 127,číslo.

ZX Spectrum 128 sa rozmiestnením základných pamäťových segmentov taktiež nelíši od svojho predchodcu. Priamo prístupných je rovnako iba 65536 bajtov. Ostatné sú rozdelené do 8 baniek RAM po 16 kB, ktoré sa umiestňujú pomocou OUT 32765,číslo na adresu 49152. Podobne aj dve 16 kB banky ROM sa striedajú podľa potreby od adresy 0 po 16383.

16K ROM obsahuje inštrukcie, dáta a tabuľky, ktoré Spectrum potrebuje pre beh BASIC programov, prevádzanie inštrukcií, dodávaných užívateľom, ovládanie tlačiarne, magnetfónu apod. Tento priestor je od adresy 0 do 16383. Potom nasleduje 16K RAM a prípadne ďalších 32K RAM (na 48K Spectre).

Hlavný rozdiel medzi ROM a RAM je ten, že obsah ROM je pevný a nedá sa meniť, ani pri vypnutom napájaní. Obsah RAM môže byť menený ľahko, väčšinou keď chceme, niekedy však i nechcene keď vypneme zdroj. RAM sa rozdeľuje na niekoľko častí, viď. nasledujúca tabuľka:

Adresa	Obsah pamäte
16384	Pamäť obrazovky
22528	Atribúty obrazovky
23296	Vyrovnávacia pamäť tlačiarne
23552	Systémové premenné
23734	Mapy microdrive (pokiaľ je pripojený)

CHANS	Kanálové informácie	PEEK 23631 + 256*PEEK 23632
	CHR$ 128	
PROG	Program BASIC	PEEK 23635 + 256*PEEK 23636
VARS	Tabuľka premenných	PEEK 23627 +
	CHR$ 128	256*PEEK 23628
E LINE	Editovaný príkaz alebo riadok	
	CHR$ 13	
	CHR$ 128	
WORKSP	Vstupné dáta	PEEK 23649 +
	CHR$ 13	256*PEEK 23650
	Dočasný pracovný priestor po CHR$ 13	
STKBOT	Zásobník kalkulátoru	PEEK 23651 + 256*PEEK 23652
STKEND	Zásobná pamäť	PEEK 23653 + 256*PEEK 23654
Z80A STACK POINTER	Strojový zásobník	
	Zásobník GOSUB	neprístupný pre BASIC
RAMTOP	CHR$ 62	PEEK 23730 + 256*PEEK 23731
UDG	UDG	PEEK 23675 + 256*PEEK 23676
P RAMT		PEEK 23732 + 256*PEEK 23733

1.Pamäť obrazovky (Display file) - adresa 16384 až 22627

Toto je kópia televízneho obrazu. Pre každý bod obrazu v hornej i dolnej časti obrazovky (teda 256 krát 192 bodov) sú v tomto priestore zopovedajúce bity pamäte. Pozostávajú zo 6144 bajtov, ktoré sú rozložené takpovediac netradične, kurióznejšie než udáva manuál k počítaču. Ak je na obrazovke niektorý bod nastavený na farbu INK, potom mu zodpovedá bit s hodnotou 1.

Tento program dáva vysvetlenie o rozložení obrazovej pamäti:

```
10 FOR a = 16384 TO 22527
20 POKE a,255
30 NEXT a
```

2. Atribúty obrazovky (Attributes File) - adresa 22528 až 23295

Tento priestor pamäti obsahuje informácie o farbe, jase a blikaní obrazovky. Ak si predstavíte, že napr. znak medzery pozostáva z matice 8 krát 8 bodov, potom tieto body vo vnútri tejto matice môžu mať iba jednu farbu INK a jednu farbu PAPER, pretože ich atribúty pochádzajú z rovnakého miesta - atribúty pracujú v rovnakej mierke ako súradnice PRINTu. To je dôvod, prečo nemôžete mať zelenú príšerku na žltom pozadí s červenými očami a bielymi zubami. Jeden bajt pozostávajúci z 8 bitov v priestore atribútov obsahuje informáciu o FLASH, BRIGHT, PAPER, INK pre jednu znakovú pozíciu na obrazovke. Je tu 768 bajtov (32*24), čiže spolu s dvomi dolnými dialógovými riadkami obrazovky. Sú uložené v tomto poradí: prvý rad 32 bajtov pre 1. riadok, druhý rad 32 bajtov pre druhý riadok, atď...

Tento krátky program nám to ukáže:

```
10 FOR a = 22528 TO 23295
20 POKE a,199
30 NEXT a
```

Tri bity sú použité pre PAPER, tri pre INK, jeden pre FLASH a jeden pre BRIGHT. Sú uložené binárne, teda jeden bit sám osebe dáva 0 alebo 1 a tri bity dávajú 0 až 7. Nie sú tu však uložené parametre 8 a 9. Iba výsledok činnosti počítača rozhoduje o tom, či INK 9 ukončí červenú, bielu alebo akúkoľvek.

Označenie bitov v bajte atribútu:

bit 7	bit 6	bit 5 bit 4 bit 3	bit 2 bit 1 bit 0
FLASH	BRIGHT	PAPER	INK

3. Vyrovnávacia pamäť tlačiarne (Printer buffer) - adr.23296 až 23551

Informácia o znakoch čakajúcich na odoslanie do tlačiarne. Pozostáva z 256 bajtov (čo je 32 x 8), takže môže byť tlačená ľubovolná grafika . Sú tu uložené znaky v tom tvare, v ktorom sú tlačené.

4. Systémové premenné (The system variables) - adresa 23552 až 23734

Bajty obsahujúce informácie, ktoré sa vzťahujú k najrôznejším oblastiam činnosti počítača, ukazujú, kde začínajú rôzne časti pamäti, kde končia a pod. Majú mená, ale počítaču je to jedno, slúžia len pre lepšiu orientáciu programátorovi. Tieto mená sú napr. SCR CT, VARS a pod.

5. Mapy microdrive (Microdrive maps)

Tento priestor obsahuje informácie, vzťahujúce sa k microdrive, pokiaľ je však pripojený. Keď nie je pripojený, nie je tu nič a systémová premenná CHARS ukazuje na adresu 23734.

6. Kanálové informácie (Channel information)

Na štandartnom 16K alebo 48K Spectre, ktoré nemajú pripojené žiadne prídavné zariadenia (okrem tlačiarne, ktorá však na tuto časť pamäti nemá vplyv), sú tu uložené informácie o štyroch vstupných/výstupných kanáloch. Tieto informácie o štyroch vstupných/výstupných kanáloch hovoria, aké dáta pôjdu odkiaľ a kam a udávajú adresy rutín v ROM, ktoré operácie vstup/výstup pre ktorý kanál zaisťujú.

```
244       9     246   16      75      [K]
244       9     246   21      83      [S]
129      15     131   21      82      [R]
244       9     245   21      80      [P]
128
```

Znaky vpravo sú "mená súborov", čiže máme kanály K, S, R, P.

Kanál K je "Klávesnica". Informácia môže prichádzať z klávesnice a odchádzať na dolnú časť obrazovky.

Kanál R uvoľňuje informácie do pracovnej časti pamäte a

Kanál P uvoľňuje informácie do tlačiarne.

Čo však znamenajú tie štyri bajty pred "menom súboru"? Keď si to prepíšeme ako dve adresy a meno súboru, vidíme, že sú to adresy rutín v ROM, ktoré príslušné činnosti ovládajú.

7. Priestor programu (Program area)

Toto je časť pamäte, kam sa ukladá program v BASICu. Na jeho začiatku ukazuje adresa v systémovej premennej PROG. Na začiatku je uložený prvý bajt prvého čísla riadku. Programový priestor končí na adrese, zmenšenej o 1, ktorá je uložená v systémovej premennej VARS. Riadok programu v BASICu je uložený takto:

dva bajty nesúce číslo riadku MSB LSB	dva byty nesúce dĺžku riadku+1 pre CHR$ 13 na konci riadku	text programového riadku	CHR$ 13 BIN 0000110 (ENTER)

Každý riadok končí znakom CHR$ 13, čo je ENTER. Viacnásobné príkazy sú oddelené dvojbodkou (CHR$ 58) a čísla sú uložené dvakrát, najprv ako CODE ich číslic a potom po CHR$ 14 (čo znamená číslo) nasleduje 5 bajtov, ktoré predstavujú toto číslo v celočíselnom formáte alebo s plávajúcou desatinnou čiarkou.

Nasledujúci jednoduchý program vám umožní pozrieť sa na ktorýkoľvek riadok, ktorý je zapísaný ako prvý v programe. Zistí kde v pamäti prvý riadok začína (riadok 10) a kde končí (riadok 20). K tomu využíva bajty 3 a 4 programového riadku, ktoré zachovávajú dĺžku programového riadku od 5. bajtu až do konca (po ENTER-vrátane). Všetky adresy sú získané pomocou PEEK a údaje sú vytlačené na obrazovku v troch stĺpcoch. Prvý sú adresy, druhý čísla uložené na týchto adresách a tretí zodpovedajúce znaky, pokiaľ

sú tlačené. Príklad ukazuje výpis príkazu REM:

```
23755       0
23756       1
23757       15
23758       0
23759       234         REM
23760       108         l
23761       105         i
23762       110         n
23763       101         e
23764       32
23765       111         o
23766       102         f
23767       32
23768       66          B
23769       65          A
23770       83          S
23771       73          I
23772       67          C
23773       13
```

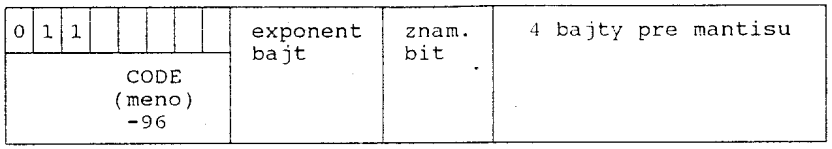
```
 1 REM Line of BASIC
10 LET start=PEEK 23635+256*PEEK 23636
20 LET finish=start+3+PEEK (start+2)+256*PEEK (start+3)
30 FOR a=start TO finish
40 LET char=PEEK a
50 PRINT a;TAB 8;char;TAB 16;CHR$ (char) AND char>31
60 NEXT a
```

8. Tabuľka premenných (Variables area)

Tento priestor v pamäti začína na adrese, ktorá je v systémovej premennej VARS a končí na adrese, ktorá je v E-LINE. V tomto priestore sú uložené všetky premenné,ktoré používa program BASIC. Aby sa zamedzilo zmätku v menách premenných, sú uložené podľa rôzneho formátu podľa druhov (t.j. iný formát pre reťazce, polia, číslicové premenné atď.). Iný druh bitov pre prvé písmeno v názve premennej udáva počítaču rozdiely medzi rôznymi typmi premenných. Tieto písmená sú obyčajne uložené ako malé, ale rozdielny druh bitov prvého písmena umožňuje, že sa nemusíme ďalej pozerať na CODE tohto písmena.

Teraz sa pozrieme detailne na rôzne typy premenných. Prvé písmeno mena všetkých premenných je uložené v jednom bajte ako CODE tohoto písmena - 96, pričom pre meno sú významné bity 0 až 4. Bity 5, 6 a 7 majú hodnotu, ktorá závisí od typu premennej a nie sú použité v mene. Zodpovedajúce hodnoty, ak sú bity 5 a 6 v 1, sú 32 a 64, čo je celkom 96. Skutočné prvé písmeno mena premennej je uložené ako CODE malého písmena v bitoch 0 až 4, zmenšené o hodnotu bitov 5 a 6. Tak bude prvé písmeno názvu premennej, ktoré bolo napr. "a" , uložené v bitoch 0 až 4 ako BIN 00001. Bity 5, 6 a 7 budú v 1 alebo v 0 podľa toho, o aký druh premennej sa jedná.

Číselná premenná , ktorej meno je len jedno písmeno: Tento typ premennej má bity 5, 6 a 7 v byte mena nastavené na 1, 1 a 0. Za menom nasleduje bajt exponentu a štyri bajty mantisy:



Bajt exponentu má hodnotu od 1 do 255 (môže byť aj 1 alebo 255). Udáva, kde leží desatinná bodka. Mantisa je uložená v štyroch bajtoch. Udáva číslice uloženého čísla a môže mať hodnotu od 0.5 do 1 (nikdy 1, ale môže byť 0.5). Číslo sa potom vypočíta pomocou vzorca:

(mantisa)*2 ^ (exponent-128).

Bit, ktorý je v mantise najviac vľavo, sa používa ako znamienkový bit. Ak je číslo záporné, je 1, a ak je číslo kladné, je 0.

Pre uloženie celých čísiel od -65535 do +65535, ktoré sa dajú zapísať do dvoch bajtov, existuje trochu odlišný spôsob:

bajt 1	bajt 2	bajt 3	bajt 4	bajt 5
0 nula	0, ak je číslo kladné alebo 255, ak je záporné	LSB MSB Číslo je uložené v bajtoch 3 a 4		0 nula

Prvý bajt uloženého čísla je vždy 0. To pomáha k identifikácii formátu uloženia čísla. Druhý bajt je znamienkový - bude 0, ak je číslo kladné, alebo 255, ak je uložené číslo záporné. Bajty 3 a 4 obsahujú hodnotu čísla v poradí "menej významný bajt" (LSB), nasledovaný "viac významným bajtom" (MSB). Číslo je uložené v tvare, ktorému sa hovorí "dvojkový doplnok".

Hodnota pre kladné číslo sa zistí zo vzťahu:

LSB+256*MSB

Hodnota záporného čísla sa zistí zo vzťahu:

LSB+256*MSB-65536

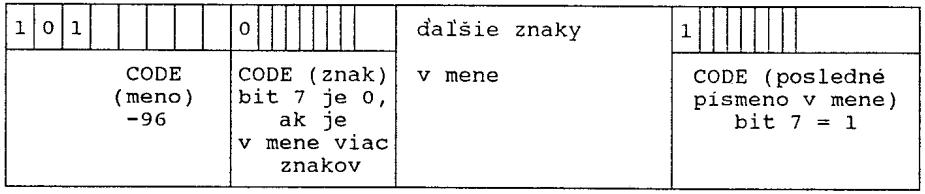
Všeobecne sa hodnota uloženého čísla zistí:

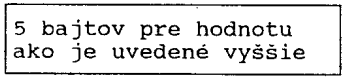
```
LET hodnota=LSB+256*MSB-(65536 AND (sign byte=255))
```

Piaty bajt je vždy 0.

Numerické premenné, ich mená skladajúce sa z viac ako jedného písmena:

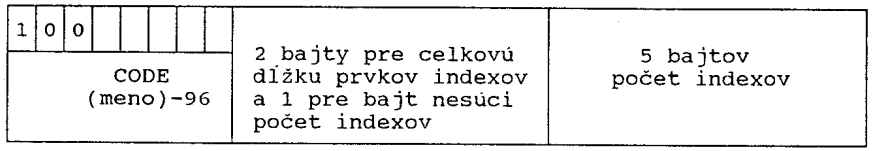
Tento typ premenných je uložený tak, že u prvého písmena názvu sú bity 5 a 7 nastavené na 1 a bit 6 na 0. Nasledujúce znaky mena sú uložené ako ich CODE, pričom je u všetkých, s výnimkou posledného, bit 7 nastavený na 0. Posledný znak mena je identifikovaný práve tým, že má bit 7 nastavený na jednotku. V dekadickom vyjadrení to znamená, že prvé písmeno názvu je uložené ako jeho CODE +64, nasledujúce znaky ako ich CODE a posledný znak ako jeho CODE+ 128. Príležitostne sa môžete pri rozbore vyššie uvedeného dostať v myslení do pekného zmätku zmiešaním dekadických a binárnych hodnôt, je lepšie použiť dvojkový zápis, ako by sme prideľovali jednotlivé bity.

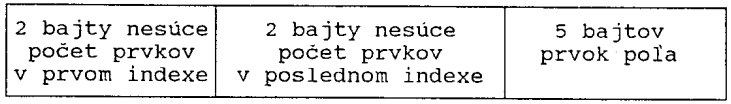




Numerické pole

Toto počítač rozozná tak, že v 1. bajte pre meno poľa sú bity 5 a 6 nastavené na 0 a bit 7 na 1. To zodpovedá, ak zoberieme všetkých 8 bitov mena ako CODE+32.

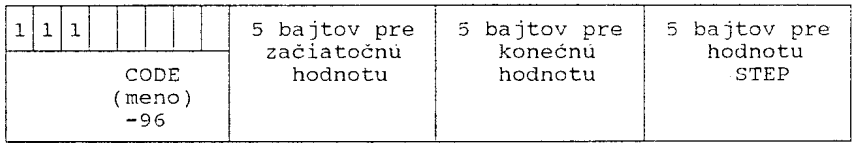


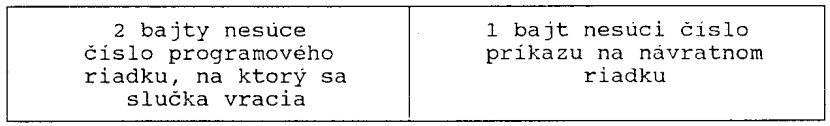


Prvky poľa sú ukladané tak, že prvky prvého indexu sú uložené najskôr, prvky druhého indexu za nimi atď., ako ste určite očakávali.

Riadiace premenné pre slučky FOR ... NEXT

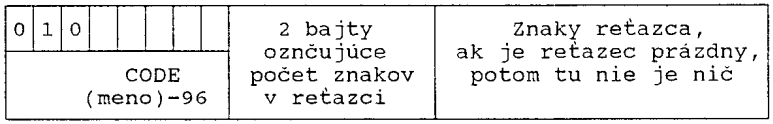
Rozoznávajú sa tak, že majú bity 5,6 a 7 v bajte mena nastavené na 1. Písmeno mena je opäť uložené do bitov 0 až 4 ako jeho CODE-96. Dekadicky, ak zoberieme do úvahy všetkých 8 bitov bajtu mena, je to CODE mena+128. Je tu jeden bajt pre meno, 5 bajtov pre začiatočnú hodnotu (číslo pred TO). 5 bajtov pre konečnú hodnotu (číslo za TO) a 5 bajtov pre hodnotu STEP. Nasledujú dva bajty, ktoré udávajú číslo programového riadku, na ktorý sa slučka vracia, a 1 bajte udávajúci číslo príkazu na návratovom riadku. Čo je spolu 19 bajtov.





Reťazcové premenné

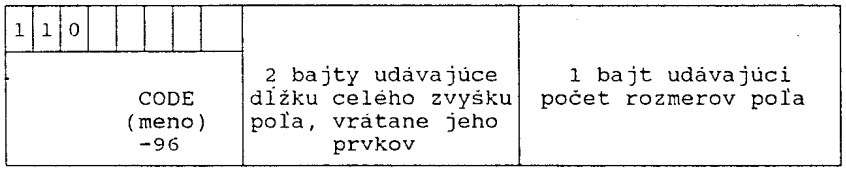
Rozoznávajú sa podľa toho, že majú v prvom bajte bity 5 a 7 na 0 a bit 6 na 1. Meno (len jedno písmeno) je uložené v bitoch 0 až 4 prvého bajtu ako jeho CODE-96, čo teda dekadicky pre celý bajt je CODE-32. Premenná sa skladá z jedného bajtu mena, nasledujú dva bajty udávajúce dĺžku reťazca (t.j. koľko znakov reťazec obsahuje znakov - obdoba funkcie LEN). Potom prídu znaky reťazca, uložené jednoducho ako ich CODE. Ak je reťazec prázdny, potom tu nič nie je.

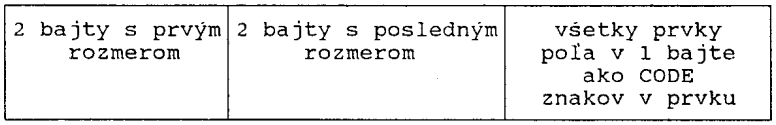


Reťazcové pole

Bity 6 a 7 prvého bajtu sú nastavené na jednotku a bit 5 na 0. Meno (len jedno písmeno) je uložené v bitoch 0 až 4 ako jeho CODE-96. Dekadicky to zodpovedá, vzťahuje sa na celý bajt, CODE+96. Premenná pozostáva z jedného bajtu mena, potom dvoch bajtov, v nich je uložená dĺžka celého zvyšku poľa vrátane jeho prvkov, rozmerov a jedného bajtu, ktorý udáva, koľko rozmerov pole má. V tomto bajte bude napr. 3 pre prípad DIM a$ (4,5,6). Rozmery potom samé nasledujú vo dvoch bajtoch v poradí, ako boli určené v programe pri zadaní DIM. Napr., ak je zadané DIM a$ (4,5,6), bude

193 meno poľa a potom budú čísla 127,0,3,4,0,5,0,6,0 a potom nasleduje CODE znakov jednotlivých prvkov poľa. Uvedené nuly sú potrebné pre využitie dvoch bajtov vo všetkých prípadoch, ktoré môžu nastať.





Editovaný príkaz/riadok, vstupné dáta a dočasný pracovný priestor

Táto časť pamäti je využívaná počítačom pri chode Basického programu, jeho editovanie a pre spracovanie informácie o premennej . Tento pristor sa môže rozširovať alebo zmenšovať až na 0 podľa toho, nakoľko ho práve potrebujeme. Ak budete chcieť tento priestor otestovať, zistíte, že neexistuje, pretože sa skutočne vytvára až v okamihu, keď je potrebný.

Zásobník kalkulátoru

Používa sa pri umiestnení čísiel s pohyblivou rádovou čiarkou celých čísiel a pre informáciu o reťazcoch. Všetko sa ukladá v piatich bajtoch- čísla v ich päťbajtovom tvare a pri reťazcoch päť bajtov s detailnou informáciou o nich (počiatočná adresa a pod.). Aritmetické operácie tiež využívajú tento zásobník.

Zásobná pamäť

Je to časť pamäti medzi zásobníkom kalkulátora a strojovým zásobníkom, jej využitie programátorom sa však nedoporučuje. Ak vzrastajú zásobníky, sú prevádzané zmeny na premenných atď. Uložiť sem niečo dôležité nie je bezpečné (môže to byť počítačom prepísané).

Strojový zásobník a zásobník GOSUB

Strojový zásobník je miesto, kam mikroprocesor ukladá informácie, ako napríklad obsahy registrov, ktoré potrebuje uchovať. Zásobník GOSUB uchováva informácie, ktoré počítač potrebuje, aby mohol previesť správny RETURN po prevedení GOSUB, teda aby sa po prevedení podprogramu vrátil späť na správne miesto.

RAMTOP

Toto je horná hranica pamäti, využívaná BASICom, oddeľuje UDG, ktora je uložená medzi RAMTOP a skutočným koncom (fyzickým) inštalovanej pamäte. Čokoľvek uložené nad RAMTOP je bezpečne pred všetkým z BASICu, vynímajúc POKE. NEW je účinné iba po adresu uloženú v RAMTOP. Toto zabezpečenie platí nielen pre UDG, ale môžeme tu mať aj dáta alebo strojový kód - stále je to chránené pred NEW (avšak nie pred POKE alebo vypnutím).

I N É V E R Z I E J A Z Y K A B A S I C

Táto kapitola vás bude určite zaujímať, ak máte previesť program písaný v inej verzii BASICu na vaše SPECTRUM. Ukazuje, ako možno nahradiť iné príkazy, funkcie a iné najrôznejšie verzie BASICu, ktoré však nefungujú priamo v SPECTRE. Niekedy sa jedná len o zmenu slova, inokedy je potrebná celá rutina.

ARRAYS - POLIA

V číslicových poliach môže vzniknúť problém len s indexom. Ten môže začínať nulou, pričom na SPECTRE začína 1. Všeobecné riešenie je pripočítať 1 ku všetkým indexom, aj pri príkaze DIM, ak uvažuje program s použitím indexu 0. Problémy môžu nastať s kalkulovanými indexami (napr. a(G*3-1)), pretože ich prevod môže byť klamný. Je nutné sa s nimi vysporiadať individuálne.

ASC

Zodpovedá na Spektre funkcii CODE - prinajmenšom pre znaky s CODE 32-126. Ak je použité pri uložení DATA v reťazci alebo poli a potom následne dekódované, potom buď nie je treba urobiť žiadnu zmenu, alebo je treba dôkladne preštudovať program prípad od prípadu a zistiť všetky hodnoty znakov líšiacich sa od hodnôt na Spektre. Neexistuje žiadne pevné a rýchle pravidlo.

CALL

Používa sa pre skok do strojového programu a môže byť nahradené USR. SPECTRUM pri návrate do BASICu vracia niektoré číslo, pomocou čoho ľahko zistíte, čo robiť. Napr. CALL (adresa) môže byť nahradené LET A=USR (adresa) alebo RANDOMIZE USR (adresa).

DEGREES AND RADIANS (STUPNE A RADIÁNY)

Trigonometrické funkcie SPECTRA pracujú v radiánoch. Prevod stupňov na radiány sa robí takto:

LET radians=(PI*degrees)/180

DIM

Program môže používať niekoľko polí nastavených jedným príkazom DIM (napr. DIM a(3), b(4), c(5)). Toto ide na SPECTRE nahradiť: DIM a(3): DIM b(4): DIM c(5). Treba však dávať pozor na reťazcové alebo znakové pole, pretože môže potrebovať ďalší zvláštny index: na SPECTRE sa to robí pri DIM, kde sa určí pevná dĺžka. Na iných počítačoch je dĺžka reťazca v poli len taká, aká je treba. Ak je v programe DIM a$(4), znamená to štyri reťazce a nie jeden reťazec dlhý 4 znaky, ako na SPECTRE. Prevod je možný tak, že musíme zistiť najdlhší reťazec (napr. 10 znakov) a potom zadať v príkaze DIM na Spectre DIM a$ (4,10).

DIV

Táto funkcia vracia celočíselnú časť po delení (takže výsledok 10 DIV 4 bude 2). Je možné prepísať to tak, že vložíme delenie do zátvoriek a použijeme INT, teda INT (10/4).

DO ... UNTIL (ROB ... POKIAĽ)

Program prevádza činnosť medzi príkazmi DO a UNTIL a zastaví sa len vtedy, ak nastane stav, ktorý sa vyžaduje po UNTIL. Je možné prepísať pomocou IF ... THEN GO TO ...

DRAW

Väčšina verzií BASICu kreslí čiaru po zadaní koncových súradníc, t.j. po DRAW 200,90 nakreslí čiaru, ktorá končí na 200,90. Basic Spectra ale potrebuje zadať súradnice v relatívnom tvare, t.j. koncové x - začiatočné x, koncové y - začiatočné y, t.j. počet bodov v smere x a y. Väčšina Basicov pred DRAW používa príkaz MOVE, pomocou ktorého nastaví začiatočný bod kreslenia. Takže konverzia MOVE a,b:

DRAW x,y s absolútnymi hodnotami x a y bude pre Spectrum vyzerať takto:

```
PLOT a,b: DRAW (x-a), (y-b)
```

ELSE

Je to časť príkazu IF ... THEN ... ELSE, ktorý spôsobí, že v prípade, ak nie je podmienka pre IF splnená, zostatok riadku sa nepreskočí, ale sa prevedie to, čo je za ELSE. Takýto zložený príkaz možno na Spectre preložiť do dvoch riadkov. Napr.:

```
IF x=1 THEN PRINT "x=1" ELSE PRINT "x nie je 1"
```

V Basicu Spectra:

```
IF x=1 THEN PRINT "x=1"
IF x<>1 THEN PRINT "x nie je 1"
```

Pre niektoré prípady je možné použiť AND pre spojenie podmienok, ale

rozpísanie do dvoch riadkov je najjednoduchší spôsob.

END

Môže byť väčšinou jednoducho vynechaný. Vo väčšine prípadov vykonáva to isté ako STOP.

Umocňovanie

Rôzne verzie používajú pre ne rôzne znaky. Stačí len nahradiť všelijaké ** alebo ^ známou šípkou hore - kláves H.

FOR ... NEXT SLUČKY

Rozdiel medzi verziami je v tom, kde program testuje, či je už koniec opakovania. Ak je test robený v príkaze NEXT, znamená to, že celá slučka po pozitívnom zistení, že je koniec prevedie priebeh cyklom ešte raz. Na Spectre je tento test robený v príkaze FOR a ak je dosiahnutá konečná hodnota, slučka sa už neprevedie, pretože by potom bola počiatočná hodnota vyššia ako konečná.

GET A GET$

Toto "číta" klávesnicu a zvyčajne program čaká na stlačenie niektorej klávesy. Celý výraz vyzerá asi takto: LET A=GET alebo GET A (a rovnako to platí aj pre GET$). Toto môže byť prevedené pomocou INKEY$ (ak má počítač čakať na stlačenie určitej klávesy, potom musíme urobiť špeciálnu úpravu) alebo pomocou INPUT (potom sa musí použiť ENTER). Použitie INKEY$ pre prevod LET A=GET alebo GET A:

```
1000 LET A=CODE INKEY$: IF A=0 THEN GO TO 1000
```

Prevod GET A$ (t.j. LET A$=GET A$):

```
1000 LET A$=INKEY$: IF A$="" THEN GO TO 1000
```

IF ... THEN ...

Niektoré verzie dovoľujú, že THEN môže byť vynechané. Pri Spectre ale musí byť. Napr. IF x=1 PRINT "1" musí byť zmenené na IF x=1 THEN PRINT "1".

INKEY

Toto číta CODE práve stlačenej klávesy. Je možné to previesť pomocou CODE INKEY$, napr. LET A=INKEY bude na Spectre LET A=CODE INKEY$.

Môžete sa stretnúť aj s verziou, ktorá má za INKEY číslo v zátvorke. Tak je zadaný čas, počas ktorého počítač čaká na stlačenie klávesy, než začne pokračovať v ďalšej činnosti. Najjednoduchšie je vložiť PAUSE pred miesto, kde sa klávesnica "číta". Takže:

```
LET A=INKEY(50) môže byť
PAUSE 50: LET A=CODE INKEY$
```

Ak nie sú používané rovnaké časové jednotky, je treba príslušne upraviť dĺžku PAUSE.

INSTR

Táto funkcia hľadá, či sa nejaký reťazec nachádza v inom. Na prevod tejto funkcie do Basicu Spectra bude potrebný krátky program. Program nižšie uvedený bude produkovať podobné výsledky ako LET y=INSTR (b$, c$), aj keď bude pracovať pomalšie. B$ je "veľký" reťazec, v ktorom hľadáme c$, teda "malý" reťazec. Ak je c$ "väčší" než b$ alebo sú obidva reťazce rovnako veľké, nevznikne žiadna chyba.

```
9200 REM INSTR
9205 LET p=0
9210 IF LEN c$=0 OR LEN b$=0 OR LEN c$>LEN b$ THEN RETURN
9215 FOR p=1 TO LEN b$-LEN c$+1
9220 IF b$ (p TO p+LEN c$-1)=c$ THEN RETURN
9225 NEXT p
9230 LET p=0
9235 RETURN
```

Po návrate z podprogramu bude p obsahovať pozíciu, na ktorej začína hľadaný reťazec. Ak nie je reťazec c$ vôbec v b$ nájdený, alebo je c$ väčšie ako b$, potom bude p=0. Všeobecne p=0 znamená, že v b$ neexistuje presná kópia c$.

CELOČÍSELNÉ PREMENNÉ

Normálne sú označené tým, že majú symbol % pripojený na konci názvu (napr. A%). Normálne môžete používať ľubovolné meno premennej, musíte si však byť istý, že číslo v premennej je celé (ak je A%=3/2 a potom PRINT A%, výsledok bude 1) Rovnaký výsledok na Spectre dostaneme: LET A=INT (3/2).

LEFT$

Funkcia LEFT (a$,b) vezme prvých b znakov z a$. Na Spectre je to možné napísať pomocou a$(TO b).

LINK

Toto je volanie strojového programu. Môže byť nahradené USR.

LOGICKÉ VÝRAZY

Stretnete sa v iných verziách s vyhodnotením TRUE (pravdivý) a FALSE (nepravdivý). FALSE bude mať hodnotu 0, ale TRUE môže byť 1 a -1. Tým je

myslené, že niečo ako PRINT (1=2) vytlačí vždy 0 (aj na Spectre), ale PRINT (1=1) môže byť pri niektorých verziách aj -1. Riešenie je v zmene znamienka v týchto výrazoch. Napr.

```
LET x=10-(score=6)+(time<300)
LET x=10+(score=6)-(time<300)
```

Pozor na použitie AND OR ako binárnych operátorov, ktoré pracujú s číslami bit po bite. Napr. tento druh výrazu:

```
LET a=a AND 4
```

Tu nie je prevod na Spectrum jednoduchý. Budeme musieť program celý prepísať alebo na to celé zabudnúť.

LOGS

Spectrum používa prirodzené logaritmy. Ak potrebujete logaritmy s iným základom (zvyčajne 10), potom:

```
LET LOGBASEx číslo=LN číslo/LN x
```

kde x je požadovaný základ a číslo je číslo, ktorého logaritmus hľadáme.

MAT

Skratka pre MATICE. Je to funkcia, ktorá bude pracovať so všetkými prvkami poľa:

```
10 DIM x(10)
20 DIM y(10)
30 MAT x=y
```

Spôsobí, že všetky prvky poľa x dostanú hodnoty zo zodpovedajúcich prvkov poľa y. Prepis na Spectrum je pomocou slučky:

```
10 DIM x(10)
20 DIM y(10)
25 FOR m=1 TO 10
30 LET x(m)=y(m)
35 NEXT m
```

MID$

MID$ (w$,t,u) vyberie prostredných u znakov z w$, začínajúc prvkom s poradím t. Pre Spectrum: w$ (t TO t+u-1)

MOD

a MOD b udáva zvyšok po delení a/b. Pre Spectrum:

```
zvyšok=a-(INT(a/b)*b)
```

MOVE

Všetko, čo táto funkcia vykonáva, je nastavenie PLOT kurzora, pričom sa na obrazovke nič neobjaví. Všeobecne sa používa k nastaveniu bodu, odkiaľ sa bude niečo kresliť a pre test nejakej určenej časti obrazovky. Je potrebné si pamätať, že na rozdiel od Spectra pri použití kreslenia vyplní prvý bod až pripojená funkcia DRAW. Pre simuláciu samotného MOVE x,y je treba urobiť POKE do príslušných systémových premenných:

```
POKE 23677,x: POKE 23678,y.
```

NEXT

V niektorých verziách môže byť meno premennej za NEXT vynechané. Ak je to tak, potom sa načíta riadiaca premenná, ktorá je najbližšia. Pri Spectre musí byť meno z NEXT udané vždy.

ON ... GOTO/GOSUB ...

Máva tvar: ON x GOTO 200,300,400,500. Čo znamená, že po x=1 ide na 200, x=2 na 300 atď. Najjednoduchšia cesta pre konverziu je:

```
IF x=1 THEN GOTO 200
IF x=2 THEN GOTO 300
IF x=3 THEN GOTO 400
IF x=4 THEN GOTO 500
```

Ale je možné použiť aj AND:

```
GOTO (200 AND x=1)+(300 AND x=3)+(500 AND x=4)
```

Ak čísla riadku idú za sebou v rozpätí, ktoré to dovoľuje, je možné ich vypočítať:

```
GOTO 100+(x*100)
```

PAINT

Toto je grafický príkaz, ktorý vyplní nejakú oblasť obrazovky farbou. Neexistuje jednoduchá cesta pre jeho prevod, pretože sa líši od počítača k počítaču. Tomuto a mnoho iným grafickým príkazom je najlepšie sa vyhnúť, pretože grafika je oblasťou, ktorá sa na jednotlivých počítačoch najviac odlišuje.

PEEK a POKE

Toto je ďalší veľký rozdiel medzi počítačmi. Sú potrebné zvláštne znalosti o použitom počítači a pritom je nutnosť konverzie častá.

PRINT

Na niektorých počítačoch je otáznik ? použitý ako skratka pre PRINT. Na Spectre ho však nemožno použiť, musí byť PRINT.

PROC, ENDPROC

PROC je skrátená Procedúre (postup), čo je tvar podprogramu, ktorý sa volá častejšie podľa mena než podľa čísla riadku.
Na Spectre sa PROC nahradí podprogramom volaným GOSUB (číslo riadku), ENDPROC sa nahradí RETURN.

RANDOM NUMBERS (NÁHODNÉ ČÍSLA)

Niektoré počítače generujú náhodné čísla pomocou výrazu RND(x), ktorý vracia celé číslo medzi 1 a x vrátane. Prevod na Spectrum je pomocou INT(RND*x)+1. Ak vidíte výraz RND(0), obyčajne to znamená zopakovať posledné náhodné číslo. RND(-x) je obyčajne to isté ako RAND na Spectre.

REPEAT ... UNTIL (OPAKUJ ... POKIAĽ)

Takto sa vytvára slučka bez odkazu na číslo riadku. Slučka sa opakuje tak dlho, pokiaľ nie je splnená podmienka za slovom UNTIL. Napr.:

```
10 LET x=0
20 REPEAT
30 LET x=x+1
40 UNTIL x=10
```

Je to možné vykonať pomocou IF ... THEN GOTO ... príkazu, ktorý dáme do riadku, v ktorom bolo UNTIL. GOTO má potom číslo riadku, v ktorm bolo REPEAT.

```
10 LET x=0
20 REM Začiatok slučky
30 LET x=x+1
40 IF x<10 THEN GO TO 20
```

RESET

Používa sa k vytvoreniu čierneho alebo bieleho bodu, alebo bloku na obrazovke. Tento grafický príkaz môže byť obyčajne nahradený PLOTom alebo PRINTom, v závislosti na použitom počítači a programe.

RESTORE

Spectrum má za príkazom RESTORE číslo riadku, čo často zjednodušuje program. Na iných počítačoch je celkom bežné naraziť na niečo ako:

```
100 RESTORE:FOR a=1 TO 4: READ a$: NEXT a
```

```
110 DATA "ryba","vták","cicavec","plaz"
120 DATA "mačka","pes"
```

Potom je nutné prejsť všetky dáta, až pokiaľ narazíme na to správne. Pre Spectrum zmeníme riadok 100 na:

```
100 RESTORE 120
```

RIGHT$ (R$,X)

Tento príkaz berie x znakov zprava v reťazci r$. Na Spectre ho nahradíme:

```
r$ (LEN r$-x+1 TO)
```

SCROLL

Príkaz SCROLL pre ZX 81 môžeme na Spectre nahradiť PRINT AT 21,31;"" (pozor na apostrofy). To má výhodu v uľahčení BASICu, ale nevýhodu v prekrytí otázky scroll?. LET a=USR 3582 vyvolá rutinu v ROMe, ktorá odroluje obrazovku bez toho, aby nastali problémy. Samozrejme ak sa použije nová ROM, potom môže byť adresa odlišná.

SET

viď RESET.

TAB(X,Y)

Ak neberieme do úvahy, že program môže byť napísaný pre počítač, ktorý má na riadku viac (alebo menej) znakov ako Spectrum, je to ako AT y,x na Spectre.

THEN

Môže byť na niektorých počítačoch vynechané, napr.:

```
IF x=2 GOTO 10
```

Na Spectre však musí byť:

```
IF x=2 THEN GO TO 10
```

NEDEFINOVANÉ PREMENNÉ (UNDEFINED VARIABLES)

Na niektorých počítačoch bude premenná, ktorá je v programe použitá, ale nebola definovaná pred týmto prvým použitím, nastavená na 0. Na Spectre sa však objaví chyba "2 Variable not found" (2-premenná nenájdená). Premenná musí byť na Spectre vždy definovaná pred svojim prvým

použitím (t.j. príkazom LET alebo DIM).

VAL

Pri použití VAL pri nenumerickom reťazci dá počítač hodnotu 0. Na Spectre ale interpreter Basicu objaví rôzne chyby, ako napr. 2 (viď vyššie) alebo C Nonsense in BASIC.

O B O Z N Á M E N I E S A S O B R A Z O V K O U

Po prečítaní kapitoly 24 manuálu Sinclair budete asi rozčarovaní klamne neužitočným rozložením vzoru bodov na obrazovke. Kópia obrazu je uložená v pamäti do dvoch blokov. Prvé sú body obrazovky, ktoré sú uložené v 6144 bajtoch od adresy 16384 do 22527. Každý bod obrazovky má svoj zodpovedajúci bit (8 bitov v bajte, 6144*8=49152 bodov/bitov), ale body nie sú uložené v rovnakom poradí na obrazovke ako v pamäti. To bude vysvetlené neskôr.

Druhý blok, dlhý 768 bajtov, od adresy 23295 obsahuje informácie o farbe, jase a blikaní. Tak a teraz sa pozrime na obidve časti. Najprv na pamäť obrazu (diplay file), teda na body, na ich štruktúru.

Skúsime niečo zapisovať do VIDEO RAM a budeme pozerať, čo sa deje. Pamätajte si, že každý bit v pamäti korešponduje s jedným bodom obrazovky a keďže poukujeme celý bajt, zmeníme 8 bitov/bodov naraz. Každý bod, ktorého bit je v 0, bude mať farbu PAPER. Budeme poukovať hodnotu 255, čo je BIN 11111111, čím nastavíme všetkých 8 bitov na 1. Inými slovami, na obrazovke sa to objaví ako čiara, jeden znak široká a jeden bod vysoká.

```
10 FOR a=16384 TO 22527
20 POKE a,255
30 NEXT a
```

Zistíme niekoľko vecí. Predovšetkým nemožno dosiahnuť rozumnú štruktúru, ak je obrazovka takto zaplňovaná. Ďalej obrazovka je zaplňovaná v troch blokoch, horných 8 riadkov (PRINT 0 až 7), stredných 8 riadkov (PRINT 8 až 15) a nakoniec dolných 8 riadkov (PRINT 16 až 23).

Z toho vyplýva, že aj dolná časť obrazovky je uložená v tomto bloku pamäte. Čiary sa objavujú najskôr v hornom rade bodov tlačového riadku 0, potom v hornom rade bodov tlačového riadku 1 atď. až po horný rad tlačového riadku 7. Potom nasleduje druhý rad bodov tlačového riadku 0, druhý rad bodov tlačového riadku 1 atď., pokiaľ nie je všetkých horných 8 tlačových riadkov zaplnených. Z toho môžeme odvodiť, ako nájsť adresu horného radu bodov ktoréhokoľvek znaku v ktoromkoľvek tlačovom riadku.

Použijeme PRINT súradnice y a x.

```
LET adresa=16384+y*32+x
```

Bude to ale fungovať len pre prvých 8 tlačových riadkov. Zistili ste, že po zaplnení hornej tretiny obrazovky sa zaplňuje prostredná tretina rovnakým spôsobom. A nasledujúca rovnica nám oznámi adresu horného radu bodov v tlačových riadkoch 8 až 15:

```
LET adresa=16384+2048+(y-8)*32+x
```

Číslo 2048 udáva, koľko bajtov predstavuje PRINT v radoch 0 až 7. Pamätajte, že máme 32 stĺpcov, 8 bajtov pre každý znak a 8 riadkov v tejto časti obrazovky. Teda 32*8*8=2048. Podobne potrebujeme ďalšiu rovnicu pre spodných 8 riadkov obrazovky:

```
LET adresa=16384+4096+(y-16)*32+x
```

Toto môžeme využiť aj v nasledujúcom programe. Ten vytlačí 8 dole obrazovkou v ľavom stĺpci a potom robí POKE linku nad každý vytlačený znak (akoby obrátené podčiarknutie).

```
10 LET x=0
20 FOR y=0 TO 21
30 PRINT AT y,0;"8"
40 IF y<=7 THEN LET adresa=16384+y*32+x
50 IF y>=8 AND y<=15 THEN LET adresa=16384+2048+(y-8)*32+x
60 IF y>=16 THEN LET adresa=16384+4096+(y-16)*32+x
70 POKE adresa,255
80 NEXT y
```

Je to trochu viac než nekonvenčné, mať v programe tri riadky (40,50 a 60), z ktorých môže pracovať len jeden. Ti, ktorí sú matematickejšie založení, si už všimli, že môžu byť zlúčené do jedného riadku:

```
LET adresa=16384+2048*INT(y/8)+32*y-(INT (y/8)*8))*32+x
```

alebo

```
LET adresa=16384+2048*INT (y/8)+32*y-256*INT(y/8)+x
```

alebo

```
LET adresa=16384+1792*INT (y/8)+32*y+x
```

a z toho konečne vyplýva:

```
LET adresa=16384+32*(56*INT (y/8)+y)+x
```

Vyzerá to divne, ale funguje to. A teraz môžeme riadky 40, 50 a 60 nahradiť našim jedným zázrakom:

```
10 LET x=0
20 FOR y=0 TO 21
30 PRINT AT y,0;"8"
40 LET adresa=16384+32*(56*INT (y/8)+y)+x
50 POKE adresa,255
60 NEXT y
```

To pracuje výborne. Budeme sa zaujímať, ako ale pracovať s ďalšími radmi bodov, nie iba s tou hornou. Pretože horná čiara bodov celého bloku ôsmich tlačových radov je v pamäti uložená za sebou, znamená to, že druhá čiara bodov nasleduje v pamäti o 256 bajtov (32 znakov na riadok*8 radov = 256) ďalej a tak podobne pre ďalšie čiary.

Teraz by sme si to mali urobiť pre všetky tri osemriadkové bloky. Aby sme to skrátili, zakončíme priamo rovnicou, ktorá nám umožní zaplniť

riadky znakov na pozícii y,x. Hodnota premennej k je 0 až 7 pre hornú či dolnú čiaru rovnako, ako pri definícii UDG.

```
LET adresa=16384+32*(56*INT (y/8)+y)+x+linka*256
```

Skúste tento demonštračný program:

```
10 LET x=1
20 FOR y=0 TO 21
30 PRINT AT y,0;"8"
40 FOR k=0 TO 7
50 LET adresa=16384+32*(56*INT (y/8)+y)+x+k*256
60 POKE adresa,255
70 NEXT k
80 NEXT y
```

Tento program veľmi pomaly a neefektívne zaplňuje obrazovku poukovaním do adries v pamäti obrazovky, a to v tom poradí, ako by sa dalo očakávať pri pohľade na obrazovku. Ak rozšírite hodnotu x až do 23, môžete akoby kresliť PLOTom i do riadku 22 a 23, čo sa normálne nejde. Je známe, že nemôžete poukovaním ovplyvniť stav jediného bodu obrazovky, vynímajúc prípad, že najprv zistíte stav celého bajtu (jeho hodnotu), ktorý príslušný bod ovláda a potom upravíte hodnotu tak, aby sa zmenil len požadovaný bit.

```
10 LET x=1
20 FOR y=0 TO 21
30 PRINT AT y,0;"8"
40 FOR k=0 TO 7
50 LET adresa=16384+32*(56*INT (y/8)+y)+x+k*256
60 POKE adresa,255
70 NEXT k
80 NEXT y
```

V binárnom vyjadrení existuje logická metóda pre prideľovanie obrazovej pamäti, ktorú ľahko pochopíte. Určité skupiny bitov obsahujú určité informácie, ktoré sa vzťahujú k pozíciam v obrazovej pamäti. To je využiteľné v strojnom kóde, pretože to umožňuje ľahký výstup textu a grafiky na obrazovku. Nasledujúci diagram ukazuje, ako to je organizované:

bit 15 bit 14 bit 13	bit 12 bit 11	bit 10 bit 9 bit 8
0 1 0 Pevné hodnoty ukazujúce na začiatok obrazovej pamäte BIN 01000000 00000000 = 16384 16384	Tieto 2 bity sú 0, ak je zobrazovaný bod v hornej tretine, ďalej 1, ak je v druhej tretine a 2, ak je v dolnej tretine (2048*INT(Y/8))	Určujú, o ktorú čiaru bodov v matici 8x8 sa jedná: 0 pre hornú, 7 pre spodnú (256*linka)

bit 7 bit 6 bit 5	bit 4 bit 3 bit 2 bit 1 bit 0
Týka sa čísiel tlačových pozícií zodpovedajúcej tretine obrazovky 0 pre riadky 0,8 a 16; 1 pre 1,9,17;2 pre 2,10,18 atď. až 7 pre 7,15,23	X súradnice tlačovej pozície vo zvolenom tlačovom riadku, teda 0-31

Ak dáme dohromady výrazy uvedené dole v každej sekcii diagramu, dostaneme už dobre známy výraz:

```
16384+32*INT (y/8)+y)-(INT(y/8)*8)+x+linka*256
```

Po druhé - atribúty obrazovky. Toto je, čo sa týka chápania oveľa jednoduchšie. Informácie o farbe, jase a blikaní pre každú tlačovú pozíciu sú uložené vždy v jednom bajte. Na obrazovke je 32x24 tlačových pozícií, čo je celkom 768 bytov. Tieto sú uložené v bloku pamäti, ktorému sa hovorí attributes file (súbor atribútov, atribúty obrazovky). Je od adresy 22528 po adresu 23295. Rozloženie týchto bytov je veľmi jednoduché na pochopenie v porovnaní s pamäťou obrazovky (display file).

Prvý bajt (adresa 22528) zodpovedá hornej ľavej tlačovej pozícii na obrazovke. Prvých 32 bajtov zodpovedá 32 tlačovým pozíciám v hornom rade obrazovky, ďalších 32 bajtov druhému radu obrazovky atď.

Ak chcete vidieť, že to je tak naozaj, skúste nasledujúci program, ktorý zaplní celú obrazovku čiernou farbou. Všimnite si poradie, v akom sa obrazovka zaplňuje, vrátane dolných dvoch riadkov. Spodné dva riadky sa tiež začiernia, aj keď len na chvíľu, pretože ich hneď prepíše správa OK stlačte nejakú klávesu (okrem SHIFT) na ukončenie programu.

```
10 FOR a=22528 TO 23295
20 POKE a,0
30 NEXT a
```

Informácia v bajte atribútov je držaná v takej forme, že umožňuje ovplyvniť jednotlivé bity, uchovávajúce hodnoty pre jas, blikanie, farbu papiera a atramentu.

bit 7	bit 6	bit 5, 4, 3	bit 2, 1, 0
FLASH 1 v prípade blikania 0,ak nie je blikanie	BRIGHT 1 v prípade jasu 0,ak je normálny	farba PAPER binárne	farba INK binárne

V prípade, že poznáme súradnice PRINT AT y,x, potom je jednoduché zistiť adresu, kam poukovať nami požadované atribúty pre pozíciu y,x.


```
POKE (22528+32*y+x), číslo
```

Môžeme to urobiť aj pomocou PRINT AT y,x; OVER 1;" " - ale to je už iná záležitosť.

Skúste tento program. Najskôr sa vás opýta na súradnice x a y, do ktorých potom vytlačí znak. Potom sa opýta na hodnotu atribútu a vy môžete pozorovať, ako sa menia farby, blikanie a jas v závislosti na hodnote, ktorú zadávate. Podľa diagramu hore môžete použiť pre zadanie atribút BIN. Ak chcete tlač neblikajúcu, jasnú, biely papier a modrý atrament, napíšete atribút ako BIN 01111001:

```
10 INPUT "Zadaj hodnotu pre x";x
20 INPUT "Zadaj hodnotu pre y";y
30 PRINT AT y,x;"+"
40 INPUT "Zadaj hodnotu atribútu";atr
50 POKE (22528+32*y+x),atr
```

Pre získanie hodnoty atribútu, ktorý ja na mieste tlače, nie je treba používať PEEK, pretože je k tomu určená funkcia ATTR:

Ak však chcete "píkovať", skúste

```
LET p=PEEK (22528+32*y+x)
```

DÔKLADNEJŠIE ZOZNÁMENIE S OBRAZOVKOU

Tu je prehľad niekoľko užívateľom definovaných funkcií, ktoré môžu byť užitočné:

1. Zistenie, či bit b (0 až 7) čísla n (0 až 255) je 1 alebo 0:

   ```
   DEF FN a(b,n)=INT ((n-INT (n/(2^ (b+1)))*(2^ (b+1)))/(2^b))
   ```

2. Pripočítanie percenta p k sume a:

   ```
   DEF FN s(a,p)=a*p/100+a
   ```

3. Udaním celkovej sumy t, vrátane percenta p možno zistiť pôvodnú sumu (pred pripočítaním percenta):

   ```
   DEF FN a(p,t)=(t*100)/(100+p)
   ```

4. Párna alebo nepárna. FN o() bude 1, ak je číslo n nepárne, alebo 0, ak je číslo n párne:

   ```
   DEF FN o(n)=n-2*INT(n/2)
   ```

5. Nájdenie farby PAPER na pozícii y,x:

   ```
   DEF FN p(y,x)=INT ((ATTR (y,x)-INT (ATTR (y,x)/64)*64)/8)
   ```

6. Nájdenie farby INK na pozícii y,x:

```
DEF FN i(y,x)=INT (ATTR(y,x)-INT (ATTR(y,x)/8)*8)
```

7. Zistenie stavu FLASH (či je 1 alebo 0) na pozícii y,x:

```
DEF FN f(y,x)=INT(ATTR (y,x)/128
```

8. Zistenie stavu BRIGHT na pozícii y,x:

```
DEF FN b(y,x)=((ATTR (y,x)-INT (ATTR (y,x)/128)*128)/64
```

9. PEEK 2 bajtov od adresy a:

```
DEF FN p(a)=PEEK a+256*PEEK (a+1)
```

Využitie tejto funkcie, napr. v tvare: PRINT FN (23641) - 16384 vám dá vo chvíli použitia prehľad o tom, koľko pamäte ste využili (vrátane časti pre premenné).

10. Počet bajtov pamäte, ktoré zostávajú voľné k dispozícii:

```
DEF FN n()=PEEK 23730+256*PEEK 23731-PEEK 23653-256*PEEK 23654
```

11. Zaokrúhlenie hodnoty x na najbližšie celé číslo:

```
DEF FN w(x)=INT (x+0.5)
```

12. Náhodné číslo medzi 1 a x:

```
DEF FN r(x)=INT (RND*x)+1
```

13. Výpis reťazca do stredu obrazovky (myslené stranovo):

```
DEF FN m(m$)=(16-LEN m$/2) AND LEN m$<33
```

FN vráti hodnotu pre TAB, aby reťazec bol v riadku umiestnený súmerne. Pracuje však len pre reťazce kratšie než 32 znakov. Ak je reťazec dlhší, vzdá FN svoju činnosť a TAB bude 0. Najlepšie využitie je v pridaní FN priamo do príkazu PRINT, ale nezabudnúť dať reťazec za TAB do zátvoriek, napr. PRINT TAB FN m(a$);a$ alebo PRINT TAB FN m("pes");"pes".

14. Nájdenie logaritmu čísla n so základom 10 (t.j. dekadického logaritmu) využitím prirodzených logaritmov, v ktorých počíta Spectrum:

```
DEF FN l(n)=LN n/LN 10
```

Funkcia môže byť využitá aj pre nájdenie logaritmu s iným základom, napr. b:

```
DEF FN L(n,b)=LN n/LN b
```

15. Zaokrúhlenie čísla a na dve desatinné miesta:

```
DEF FN r(a)=INT(a*100+0.5)/100
```


16. Zaokrúhlením čísla a na d desatinných miest:

```
DEF FN r(a,d) =INT (a*(10^d)+0,5)/(10^d).
```

V S T U P N É A V Ý S T U P N É K A N Á L Y

Niekedy je potrebné použitie určitých príkazov pre umožnenie výstupu (OUTPUT) na rôzne miesta určenia alebo pre akceptovanie vstupu (INPUT) z rôznych zdrojov. Napr. skúste INPUT #2;"To je #2";#1;x.

Uvádza nás to do systému kanálov, používaných pre vstup a výstup informácií do a zo Spectra. Existuje 19 kanálov. Kanály -3 až -1 používa operačný systém a sú prakticky pre programátora nepoužiteľné. Na základnom Spectre (t.j. nie je pripojené rozšírenie pamäte, microdrive atď.) sú kanály 0 až +3 realizované tak, ako ukazuje tabuľka. Všimnite si, že normálne len jeden kanál umožňuje vstup.

Kanál	INPUT	OUTPUT
0	nič	pracovný priestor pamäte (dolná časť obrazovky)
1	klávesnica	editačný priestor (dolná časť obrazovky)
2	nič	horná časť obrazovky
3	nič	tlačiareň

Kanály 4 až 15 nie sú v dobe písania tohto spisu využité (pokiaľ si ich užívateľ neotvorí). Budú pravdepodobne použité pre niektoré zariadenia, ako RS 232, microdrive a vytváranie počítačových sietí. Môžeme využiť určité kanále pre tlač, tak napr.:

```
10 FOR a=0 TO 3
20 PRINT #a;"Toto je #";a
30 NEXT a: PAUSE
```

Všimnite si, ako môže ísť tlačový výstup na obe časti obrazovky alebo tlačiareň. PRINT nie je jediným príkazom, ktorý môže byť touto cestou využitý. Skúste nasledujúce:

```
20 INPUT #2;"Vlož číslo ";#1;x
```

Reťazec správy pre INPUT sa objaví v hornej časti obrazovky (všimnite si, že potom nie je vymazaný), zatiaľ čo všetko ostatné, čo napíšete, sa objaví na normálnom mieste. Je to preto, že INPUT z klávesnice môže byť len cez kanál 1, tak ako to udáva ROM. OUTPUT môže ísť ku ktorémukoľvek zadanému kanálu. LLIST a LPRINT môžu by nahradené PRINT #3. Skúste LPRINT #2 - kto potrebuje ešte PRINT ? Obyčajne najpoužívanejšie je využitie PRINT #1, čo nám umožní tlač do dolných dvoch riadkov obrazovky. Aby ste videli, čo sa stane, ak skúsite získať INPUT z kanálov číslo 0, 2 alebo 3, skúste INPUT #3;x.

Príklad sa pokúša získať vstupné údaje z tlačiarne, čo sa zrejme nedá. Objaví sa chyba J Inalid I/O Device (neplatné vstupné/výstupné zariadenie). Môžete si otvoriť kanál príkazom OPEN #. Aby to fungovalo, musíte určiť, ktoré zariadenie je pripojené ku ktorému kanálu. Po napísaní OPEN príde číslo kanálu, potom čiarka a nakoniec písmeno označujúce pripojené zariadenie (v tejto dobe R-pracovný priestor/dolná obrazovka, K - editačný priestor/dolná obrazovka a klávesnica S -horná obrazovka alebo P-tlačiareň). Napr. OPEN #5,"P". To isté je treba urobiť pri uzatvorení kanála použitím CLOSE #, napr. CLOSE #5. Vyvarujte sa pokusu o uzatvorenie kanála, ktorý predtým nebol otvorený, ináč sa dočkáte tragických koncov.

ÚPRAVA ČÍSELNÝCH VÝPISOV

Ak je treba upravovať čísla na obrazovke do nejakého tvaru alebo formulára, môže sa z toho stať zložitá záležitosť. Môže sa požadovať výpis len určitého počtu desatinných miest, výpis čísiel pod sebou tak, aby boli vyrovnané desatinné čiarky a podobne.

Väčšinou sa to robí tak, že sa čísla prevedú do reťazca a ten sa potom znak po znaku kontroluje, až sa nájde desatinná čiarka, celočíselná časť sa študuje, aby sa zistilo, koľko je treba pridať vľavo núl alebo medzier atď. Ukážeme si rutiny, ktoré zabezpečujú často užívané spôsoby formátovania.

1. Zaokrúhlenie čísla na určený počet desatinných miest

Numericky je veľmi jednoduché zaokrúhliť číslo na dve desatinné miesta. Stačí na to vzorec:

```
LET a=INT (číslo*100+0.5)/100
```

Je to veľmi užitočné napr. pri práci s peniazmi. Nasledujúci krátky program vám ukáže, čo táto rutina vie a čo prišlo skrátka. Riadok 10 generuje číslo amount ako náhodné, riadok 20 ho redukuje do premennej twodec na dve desatinné čísla. Obe premenné sú v riadku 30 tlačené vedľa seba, pre porovnanie:

```
10 LET amount=RND*100
20 LET twodec=INT (amount*100+0.5)/100
30 PRINT amount,twodec
40 GO TO 10
```

```
34.892273      34.89          42.144775      42.14
16.993713      16.99          60.923767      60.92
74.623108      74.62          69.326782      69.33
96.762085      96.76          99.543762      99.54
57.159424      57.16          65.782166      65.78
87.005615      87.01          33.700562      33.7
25.434875      25.43          27.616882      27.62
 7.699585       7.7           71.348572      71.35
77.574158      77.57          51.174927      51.17
18.086243      18.09          38.174438      38.17
56.561279      56.56          63.153076      63.15
```

Rutina pracuje dobre, aj keď trochu neúhľadne. Čísla menšie ako 0.1

sú tlačené bez núl pred desatinnou čiarkou, čísla >= 0.1 a<1 sú tlačené s nulou pred desatinnou čiarkou. Môže sa tiež objaviť premenný počet číslic za desatinnou čiarkou, napr. číslo 2 (žiadna číslica za desatinnou čiarkou), 2.2 (1 číslica) a 2.24 (2 číslice za desatinnou čiarkou).

Ak chcete zaokrúhlovať číslo amount na počet desatinných miest udaný premennou places, použijete túto rutinu. Premenná rounded je amount po zaokrúhlení:

```
LET rounded=INT (amount*(10^places)+0.5)/(10^places)
```

Použitie umocňovania zpomaľuje túto rutinu. Ak sa má rutina používať viac ako jedenkrát, je lepšie uložiť umocňovanie do inej premennej:

```
LET mult=10 ^ d: LET rounded=INT amount*mult+0.5)/mult
```

2. Zaokrúhlenia na pevný počet desatinných miest s pridaním núl k desatinnej čiarke, ak je treba

Tu je amount číslo, ktoré sa bude tlačiť na dve desatinné čísla. Prevádza sa to reťazcom a$, kde sa doplnia číslice, ak je to nutné.

```
10 LET amount=RND*100
20 LET a$=STR$ (INT (amount*100+0.5)/100)
30 IF a$ (1)="." THEN LET a$="0"+a$
40 LET c=LEN a$-LEN STR$ INT VAL a$
50 LET a$=a$+"."00" (c+1 TO)
60 PRINT "+";amount, "+";a$
70 GOTO 10
```

```
 8.581543      8.58          59.408569      59.41
43.719482     43.72          55.688477      55.69
79.025289     79.03          76.886096      76.89
26.91803      26.92          51.483154      51.48
18.984631     18.93          61.294504      61.29
20.158904     20.19          96.907043      96.91
14.257913     14.26          68.031311      68.03
69.433594     69.43           2.3834229      2.38
 7.553100      7.55          78.868103      78.07
66.58783      66.59          15.130615      15.13
94.12536      94.13          34.892273      34.89
```

Výstup z programu je už niečo také, ako si predstavujeme napr. tabuľku cien. Riadok 10 nastavuje počiatočnú hodnotu a riadok 20 ju prvádza na reťazec a súčasne zaokrúhľuje na dve miesta. Riadok 40 odpočíta dĺžku celočíselnej časti od dĺžky celého čísla, čím zisti počet čísiel za desatinnou čiarku. To určuje, koľko 0 bude v riadku 50 za desatinnou čiarkou pridávať. Riadok 60 prevádza tlač.

3. Zarovnanie poľa podľa desatinnej čiarky

Tam, kde sa používajú tabuľky čísiel, je pre ich rýchly prehľad treba mať zarovnávanie podľa desatinnej čiarky.

```
10 LET amount=RND*1000
20 PRINT amount
```

```
30 GO TO 10
```

```
1.1291504           941.25356
85.81543            594.08569
437.19482           556.88477
790.25269           766.86096
269.1803            314.82184
189.34631           619.91504
201.88904           969.07043
142.57813           680.31311
694.33594           23.834229
75.1006             788.88103
665.8783            151.30615
```

Pre zarovnávanie desatinnej čiarky je treba zmeniť riadok 20:

```
10 LET amount=RND*1000
20 PRINT TAB 15-LEN STR$ INT amount;amount
30 GO TO 10
```

```
169.93713          421.44775          631.53076
746.23108          609.23767          365.21912
967.62085          693.25782
571.59424          995.43762
870.05615          657.82166
254.34875          337.00562
 76.99585          276.16882
775.74158          713.48572
180.86243          511.74927
565.61279          381.74438
```

Vidíte, aké je to úhľadné? Ak chcete takto zarovnať čísla zaokrúhlené na dve desatinné miesta, ktoré sú uložené v reťazci, nemožno vyššie opísané metódy použiť, pretože spätná konverzia reťazca na číslo pomocou VAL môže hodnotu reťazca ovplyvniť. Dá sa ale použiť:

```
20 PRINT TAB 15-LEN a$;a$
```

Všimnite si, že ak je číslo tak veľké, že STR$ začne používať vedeckú notáciu, začnú sa diať zvláštne veci. Samozrejme, keď ste v situácii, že môžete rozdeľovať stovky miliónov, potom vás to asi nebude zaujímať.

4. Prepisovanie stĺpcov čísiel

Ak máte zapísanú tabuľku do stĺpcov, stáva sa, že je treba niektoré z horných čísiel zmeniť na inú hodnotu. To prináša aj určité riziko, že nové číslo nebude mať rovnaký alebo väčší počet číslic ako to staré a potom tam zo starého čísla nejaká číslica zostane. Skúste ako jednoduchý príklad:

```
10 FOR a=110 TO 0 STEP -1
20 PRINT AT 0,0;a
30 NEXT a
```

Čísla tlačené na obrazovku začínajú ako trojmiestne, ale akonáhle sa má tlačiť dvojmiestne, zostáva v pravom stĺpci 0. Tomu možno zabrániť

tlačením niekoľkých medzier za číslom. Čo však v prípade, že máme vpravo od čísiel ďalšie čísla alebo text alebo grafiku? Prepíšeme ich.

Použite túto rutinu, ktorá doplní iba toľko medzier, aby bol dosiahnutý rovnaký počet tlačených miest ako má najdlhšie (v zmysle počtu číslic) číslo.

V riadku 20 má byť medzi úvodzovkami o jednu medzeru menej ako je najväčšia dĺžka čísla a číslo za TO v zátvorke musí byť maximálny počet číslic (dĺžka čísla).

```
10 FOR a=110 TO 0 STEP -1
20 PRINT AT 0,0;a;"  " (TO 3-LEN STR$ a)
30 NEXT a
```

R U T I N Y V R O M

V pamäti ROM je niekoľko užitočných rutín, ktoré sa dajú použiť vo vašich programoch, pokiaľ sú v strojnom kóde. Nasleduje popis niektorých z nich, i keď nie príliš detailný. To by zabralo samostatnú knihu. Všetky uvedené adresy sú dekadické, pokiaľ nie je uvedené inak.

16 PRINT rutina

Obsah A registra je poslaný na bežný (práve platný) výstupný kanál. V prípade, že je práve platný iný kanál ako ten, kde chcete PRINT poslať, použite rutinu 5633, pričom je v A registri číslo kanálu, ktorý má byť otvorený (horná obrazovka, kanál 2). PRINT rutinu môžete použiť na výstup riadiacich znakov tak,akoby to boli normálne znaky.

949 BEEP

Vyprodukuje zvuk podobne ako príkaz BEEP v Basicu. Po zavadení tejto rutiny musí registrový pár HL obsahovať kmitočet (nižšie hodnoty dávajú vyšší kmitočet) a registrový pár DE dĺžku tónu (nižšie hodnoty-kratší tón). Je treba poznamenať, že kmitočet vplýva na dĺžku tónu, takže ak zdojnásobíte kmitočet a máte rovnako nastavenú dĺžku tónu, nemusí byť táto dĺžka bezpodmienečne rovnaká. Opakované volanie tejto rutiny s nastavenou krátkou dobou trvania tónu pri rôznych kmitočtoch bude simulovať kĺzavý tón alebo len ovládať frekvenčnú obálku, čo v Basicu nie je možné.

3435 CLS

To isté ako Basic príkaz CLS.

3582 Scroll the screen

USR 3582 alebo CALL 3582 bude rolovať obrazovku po jednom riadku, pričom by sa ovplyvnila tlačová pozícia alebo systémová premenná SCR CT. Veľmi užitočné v hrách, kde potrebujeme priebežné rolovanie, bez toho aby sme museli zadávať niečo ako: POKE 23692,255: PRINT AT 21,31"", čo odroluje obrazovku a potom natlačí otázku scroll?

3742

Adresa začiatku riadku na obrazovke. Ak zadáme do A registru číslo

tlačového riadku (0 až 33), potom po skončení tejto rutiny bude v registrovom páre HL adresa hornej čiary bodov prvého znaku tohto radu.

3756

To isté ako BASIC príkaz COPY. Zaujímavé však je, že po zablokovaní prerušenia začína zadaním počtu riadkov hornej časti obrazovky, ktoré sa majú opísať na tlčiareň. Tento počet má byť v registri B. Rutina môže vyzerať takto:

```
DI              zablokovanie prerušenia
LD B, číslo     koľko riadkov opísať
CALL 3759       COPY na tlčiarni
```

3799 L PRINT

Tlačí to, čo je vo vyrovnávacej pamäti tlačiarne na tlačiarni ZX Printer a všetko v tejto pamäti potom maže.

6683

Tlačí obsah registrového páru BC ako dekadické číslo. Dovolené hodnoty sú 0 až 9999, pretože rutina normálne tlačí čísla riadkov.

7997 PAUSE

Registrový pár BC uchováva čas v päťdesiatinách sekundy. Podobne ako v príkaze PAUSE v Basicu tento čas skončí jeho vypršaním alebo stlačením nejakej klávesy.

8252 tlač bajtov

Táto rutina vytlačí na práve platnom výstupnom kanáli reťazec, začínajúci na adrese v registrovom páre DE, dlhom ako udáva registrový pár BC.

8855

Pre zmenu farby BORDERu stačí pred volaním rutiny 8855 uložiť do A registra číslo farby.

8874

Táto rutina nám povie, ktorý bit z adresy zodpovedá zadanému bodu obrazovky. Pri zadaní C register obsahuje x súradnicu a B register y súradnicu. Po návrate určuje číslo v A registri, ktorý bit adresy HL zodpovedá bodu so súradnicami X,Y na obrazovke.

8927 PLOT rutina

Pri zadaní dát do B registra Y súradnice a do C regitra X súradnice. Zodpovedá PLOT X,Y.

9402 DRAW rovné čiary

Pred volaním tejto rutiny musia byť nastavené štyri registre, ako je uvedené ďalej. Do B registra príde posun na osi Y (absolútna hodnota) a do C registra posun na osi X (absolútna hodnota). Registre D a E potom budu obsahovať 1 pre kladný alebo 255 pre záporný posun (SGN posunu). Register D pre X a register E pre Y.

S U P E R Z V U K Y

V Basicu sú schopnosti Spectra obmedzené na tvorenie jedného tónu s pevnou dĺžkou a kmitočtom:

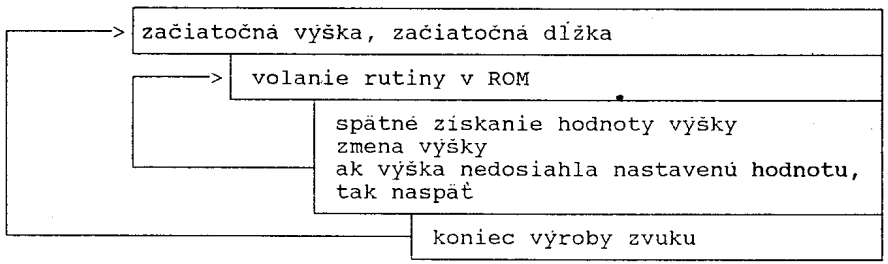
BEEP dĺžka, výška tónu

Toto môžeme čiastočne rozšíriť hraním niekoľkých krátkych tónov rýchle za sebou. Napr.:

```
10 FOR a=0 TO 21
20 BEEP 0.01,(40-(a/2))
30 NEXT A
```

Akonáhle sme ale nútení vytvoriť zložitejší zvuk, BASIC to už nie je schopný. Keď prejdeme do strojového kódu, sme stále obmedzení na riadenie dĺžky a výšky tónu. Čo ale strojový kód dokáže, je značné urýchlenie celého procesu, takže sme schopní hrať jeden krátky tón za druhým tak rýchle, že splynú v jeden dlhý tón. A ak to zariadime tak, aby sa tieto krátke tóny líšili mierne vo svojej výške, môžeme produkovať zvuky, ktoré znejú polyfonicky. Cesta k tomu, ako to urobiť, spočíva v opakovanom volaní ROM rutiny, ktorá normálne zabezpečuje prevedenie príkazu BEEP, pričom používame veľmi krátke tóny a meníme kmitočet.

Ilustračný diagram:



A tu je program v strojovom kóde, ktorý nám to prevedie. Pokiaľ neovládate strojový kód alebo Assembler, nasledujú úplné inštrukcie, ako všetko previesť na Spectre 16K a 48K. Zvuk môže byť použitý v ktoromkoľvek programe, ku ktorému pridáte túto rutinu:

Hexa-decimálne	Dekadicky		Assembler	Poznámky:
06 0A	6,10		LD B,10	čítač opakovania
C5	197	COUNT;	PUSH BC	zachovávanie čítača
21 00 00	33,0,0		LD HL,0	začiatočná výška
11 64 00	17,100,0	LOOP;	LD DE,100	dĺžka jedného zvuku
E5	229		PUSH HL	zachovanie výšky
CD 85 03	205,181,3		CALL 949	volanie rutiny v ROM
01 14 00	1,20,0		LD BC,20	krok zmeny výšky
11 64 00	17,100,0		LD DE,100	obmedzenie výšky
E1	225		POP HL	obnovenie výšky
C6 00	198,0		ADD A,0	vynulovanie ukazateľa prenosu
ED 4A	237,74		ADC HL,BC	nová výška
ED E5	229		PUSH HL	zachovanie novej výšky
C6 00	198,0		ADD A,0	vynulovanie ukazateľa prenosu
ED 52	237,82		SBC HL,DE	test limitu výšky
E1	225		POP HL	obnovenie výšky
38 EG	56,230	JR C,LOOP	(JR C,-26)	dosiahnutý limit výšky?
C1	193		POP BC	obnovenie čítača opakovania
10 DF	16,223	DJ NZ,COUNT	(DJ NZ-30)	ďalšie opakovanie?
C9	201		RET	návrat do Basicu

Relatívne skoky v Assembleri sú uvedené v obidvoch tvaroch: s návestiami pre prehľadnosť a relatívne (mínus) pre praktické použitie.Hodnoty čísiel vkladaných do registrov sú príklady. Poznámky vedľa Assemblera ukazujú, čo sa tu deje.

Detailne: 36 bajtová rutina hore opakovane vyrába tón s výškou HL a dĺžkou DE. Výška sa s každým opakovaním znižuje, a to tak dlho, až je dosiahnutá limitová hodnota výšky. Opakovanie sa prevádza toľkokrát, koľkokrát je zadané. Je treba poznamenať, že pri použití BEEP rutiny v ROM týmto spôsobom nasleduje to, že doba trvania (dĺžka) je závislá na výške tónu (predstavte si to radšej ako počet cyklov, než dĺžku tónu - vyšší kmitočet má kratšiu periódu). Výška a dĺžka môžu začínať s hodnotami 1 pre vysoký krátky tón, ak je však hodnota obidvoch príliš veľká, dostávame dlhý, veľmi hlboký neužitočný tón.

Chceme vám dať program, ktorý ukladá strojový kód v dekadickom tvare. Číta bajty strojového kódu z príkazu DATA a potom ich POKEuje do pamäte nad RAMTOP. Pre stojový program potrebujete vymedziť v pamäti 36 bajtov. Pre užívateľa 16K doporučujeme CLEAR 32563 - rutina bude začínať na adrese 32564. Pre užívateľa 48K CLEAR 65331 - rutina bude začínať na adrese 65332. Nasledujú dve verzie ukladacieho programu, jedna pre 48k, druhá pre 16k.

```
 1 REM 48K generátor zvukov
10 CLEAR 65331
20 LET address=65332: LET end=1000
30 READ byte: IF byte=end THEN STOP
40 IF byte>255 THEN STOP
50 POKE address,byte
60 LET address=address+1
70 GO TO 30
```



```
 1 REM 16K generátor zvukov
10 CLEAR 32563
20 LET address=32564: LET end=1000
30 READ byte: IF byte=end THEN STOP
40 IF byte>255 THEN STOP
50 POKE address,byte
60 LET address=address+1
70 GO TO 30
```

Riadky 10 nastavujú nový RAMTOP, nad ktorý príde strojár. Ak budete mať strojár dlhší ako 36 bajtov (kombinujete ich napríklad viac a pod.), potom musíte hodnotu nového RAMTOP vypočítať. Zo systémovej premennej RAMTOP zistíte zostávajúcu hodnotu (normálne 32599 na 16K a 65367 na 48K) a potom od nej odčítate počet bajtov vášho strojového kódu. To je nová hodnota RAMTOP, ktorú nastavíte pomocou CLEAR. Strojový program bude začínať na adrese o jednu vyššie ako je nový RAMTOP.

Sme na riadku 20. Premenná "address" je adresa, na ktorej strojový program začína, hneď za RAMTOP, ako bolo popísané. Premenná "end" je použitá ako označenie konca súpisu DATA. Myslí sa tým to, že hodnoty v príkaze DATA musia byť medzi 0 až 255. Ak je hodnota vyššia, znamená to koniec. Mohlo by to byť urobené pomocou nejakého čísla, ale použitie slova umožňuje okamžité vizuálne zistenie stavu (pritom nie všetky počítače môžu obsahovať slová v príkaze DATA ako v tomto prípade). To všetko urobili riadky 30 a 40. Riadok 50 ukladá bajty strojového kódu na príslušné miesto pamäte. Riadok 60 pričítava 1 k adrese pre uloženie ďalšieho bajtu.

Nasleduje niekoľko príkladov zvukov, ktoré môžu byť generované. Sú uvedené príkazmi DATA, ktoré sa používajú spoločne s ukladacím programom. Všetky obsahujú 36 bajtov strojového kódu, ale s rôznym nastavením výšky, dĺžky, opakovania atď. Zapíšte príkazy DATA podľa vášho výberu a prejdite ukladací program. Program by sa mal zastaviť zo správou 9 STOP statement, 30:3. Ak sa to nestane a vy ste v programe nerobili žiadne zmeny, stala sa nejaká chyba. Uložte program na pásku, ináč ho stratíte, keď skúsite prejsť strojový program. Ak ste pripravený počúvať zvuk, spustite strojový program:

```
LET a=USR 32564     (pre 16K)

LET a=USR 65332     (pre 48K)
```

Môžete použiť aj RANDOMIZE USR 65332, ale tým môžete ovplyvniť náhodnosť náhodných čísiel vo vašich hrách, v ktorých tieto zvuky nájdu určite najväčšie uplatnenie. Jedno zaujímavé odbočenie: Kde vidíte v programe, že je k volaniu strojového kódu použité RANDOMIZE a sú tiež generované náhodné čísla, je lepšie použiť RANDOMIZE (0*USR xxxxx), obíde sa tak problém s ovplyvnenou náhodnosťou. Ale späť k veci.

Tu je niekoľko príkazov DATA pre rôzne zvuky:

```
 98 REM BOMB FALLING
 99 REM SOUNDS DATA LIST
100 DATA 6,1,197,33,0,0,17,1
110 DATA 0,229,205,181,3,1,20,0
```

```
120 DATA 17,0,12,225,198,0,237
130 DATA 74,229,198,0,237,82
140 DATA 225,56,230,193,16,223
150 DATA 201,END

 98 REM PHASOR FIRE
 99 REM SOUNDS DATA LIST
100 DATA 6,1,197,33,0,0,17,1
110 DATA 0,229,205,181,3,1,1,0
120 DATA 17,100,1,225,198,0,237
130 DATA 74,229,198,0,237,82
140 DATA 225,56,230,193,16,223
150 DATA 201,END

 98 REM REPEATED PHASOR FIRE
 99 REM SOUNDS DATA LIST
100 DATA 6,10,197,33,0,0,17,1
110 DATA 0,229,205,181,3,1,1,0
120 DATA 17,100,1,225,198,0,237
130 DATA 74,229,198,0,237,82
140 DATA 225,56,230,193,16,223
150 DATA 201,END

 98 REM RASPBERRY
 99 REM SOUNDS DATA LIST
100 DATA 6,1,197,33,0,10,17,1
110 DATA 0,229,205,181,3,1,100,0
120 DATA 17,0,30,225,198,0,237
130 DATA 74,229,198,0,237,82
140 DATA 225,56,230,193,16,223
150 DATA 201,END

 98 REM ALIEN MACHINERY OR UFO
 99 REM SOUNDS DATA LIST
100 DATA 6,20,197,33,0,4,17,1
110 DATA 0,229,205,181,3,1,50,0
120 DATA 17,0,6,225,198,0,237
130 DATA 74,229,198,0,237,82
140 DATA 225,56,230,193,16,223
150 DATA 201,END

 98 REN MYSTERIOUS SOUNDS
 99 REM SOUNDS DATA LIST
100 DATA 6,5,197,33,0,10,17,10
110 DATA 0,229,205,181,3,1,0,2
120 DATA 17,0,19,225,198,0,237
130 DATA 74,229,198,0,237,82
140 DATA 225,56,230,193,16,223
150 DATA 201,END

 98 REM ALARM
 99 REM SOUNDS DATA LIST
100 DATA 6,10,197,33,0,0,17,100
110 DATA 0,229,205,181,3,1,0,1
120 DATA 17,0,3,225,198,0,237
```



```
130 DATA 74,229,198,0,237,82
140 DATA 225,56,230,193,16,223
150 DATA 201,END

 98 REM LASER BEAM
 99 REM SOUNDS DATA LIST
100 DATA 6,25,197,33,0,0,17,6
110 DATA 0,229,205,181,3,1,50,0
120 DATA 17,0,1,225,198,0,237
130 DATA 74,229,198,0,237,82
140 DATA 225,56,230,193,16,223
150 DATA 201,END

 98 REM DRUM SYNTH-TYPE  PEEOW
 99 REM SOUNDS DATA LIST
100 DATA 6,1,197,33,0,0,17,5
110 DATA 0,229,205,181,3,1,1,0
120 DATA 17,0,1,225,198,0,237
130 DATA 74,229,198,0,237,82
140 DATA 225,56,230,193,16,223
150 DATA 201,END

 98 REM BIRD (OR SEEING STARS !!)
 99 REM SOUNDS DATA LIST
100 DATA 6,14,197,33,0,0,17,40
110 DATA 0,229,205,181,3,1,25,0
120 DATA 17,240,0,225,198,0,237
130 DATA 74,229,198,0,237,82
140 DATA 225,56,230,193,16,223
150 DATA 201,END
```

Po vypočutí sú pre vás tieto zvuky aj pri všetkej vašej úcte k BEEPu Basicu určite prekvapením. Jediné, čo uvedená rutina nedokáže, je ostupný tón. Pretože sa tóny opakujú veľmi rýchlo, nebude v tom veľký zdiel. Ak chcete, tu je mierne upravená rutina, ktorá bude produkovať aj ostupný tón.

Hexadecimálne	Dekadicky	Assembler	Poznámka
06 01	6,1	LD B,1	čítač opakovaní
C5	197 REPT;	PUSH BC	uloženie čítača
21 E8 03	33,232,3	LD HL,1000	počiatočná výška
11 01 00	17,1,0 LOOP;	LD DE,1	dĺžka zvuku
E5	229	PUSH HL	uloženie výšky
CD B5 03	205,181,3	CALL 949	volanie BEEP
E1	225	POP HL	obnova dĺžky
01 01 00	1,1,0	LD BC,1	zmena výšky
C6 00	198,0	ADD A,0	nulovanie carry
ED 42	237,66	SBC HL,BC	nová výška
30 EF	48,239	JR NC,LOOP(JR NC,-17)	viac tohto zvuku?
C1	193	POP BC	obnova čítača
10 EB	16,232	DJ NZ,REPT(DJ NZ,-24)	nové opakovanie?
C9	201	RET	návrat do Basicu

Táto rutina je dlhá iba 27 bajtov a obsahuje dodatok "limit výšky", čím je myslené, že ak výška začína na akejkoľvek hodnote, končí vždy na najvyššej. Môžete si túto rutinu prepísať do DATA príkazu rovnako ako predchádzajúci, ale zostane tu niekoľko bajtov nevyužitých. Ak vás to hnevá, tak si prepočítajte hodnotu RAMTOP a upravte riadok 10 a 20 ukladajúceho programu. Pričom na druhej strane, ak to však necháte tak, získate určitý štandart, ktorý vám umožní meniť zvuky tak, ako si práve prajete. Ak chcete, aby táto kratšia rutina bola rovnako dlhá ako predchádzajúca, potom ju na potrebný počet bajtov pred END doplňte 0, čo je v strojovom kóde NOP (no operation-žiadna činnosť).

```
 98 REM ASCENDING TONE
 99 REM SOUNDS DATA LIST
100 DATA 6,1,197,33,232,3,17,1
110 DATA 0,229,205,181,3,225,1
120 DATA 1,0,198,0,237,66,48
130 DATA 239,193,16,232,201,END
```

```
 98 REM FASTER ASCENDING TONE
 99 REM SOUNDS DATA LIST
100 DATA 6,1,197,33,232,2,17,1
110 DATA 0,229,205,181,3,225,1
120 DATA 10,0,198,0,237,66,48
130 DATA 239,193,16,232,201,END
```

Ďalšia otázka je, ako pridať tieto zvuky do vašich programov. Obávame sa, že pokiaľ nemáte skúsenosti so strojovým kódom, tak si pri tom aj vlasy vytrháte. Sú tri hlavné možnosti:

1. Začleniť ukladací program a príkazy DATA do vášho programu. Keď je potom spustené vykonávanie vášho programu (vhodné je využiť možnosti SAVE LINE), potrebujete potom nastaviť strojový kód len jedenkrát. Nasledujúce RUNy prejdú len program (a nebude sa opäť nastavovať stroják). Môže to mať túto formu:

```
9900 REM Ukladací program
.... ................
9950 REM Príkazy DATA
.... ..............
9990 GO TO 1
```

Tento program by sa mal uložiť na pásku ako SAVE "program" LINE 9900.

2. Nastavenie strojového kódu najskôr a jeho uloženie na pásku s použitím SAVE CODE pre nasledujúce spätné LOAD do programu. Je to však časovo náročné.

3. Uložiť všetko hotové do programu v príkaze REM. Bude to potom uložené podľa všetkých pravidiel na pásku ako bežný riadok vášho programu, ktorý tieto zvuky využíva.

Nasledujúci program vám takto umožní uložiť všetkých 12 zvukov, o ktorých bola reč. Zapamätajte si, že každá rutina je 36 bajtov dlhá a bolo ich 12, takže v príkaze REM potrebujeme najmenej 12*36 alebo 432 znakov po slove REM v jednom riadku.

Riadky 10 až 60 obsahujú ukladací program. Zostatok su príkazy DATA so strojovým kódom rôznych zvukov. Normálne môžete pri každom zvuku vynechať príkaz REM. Je tam preto, aby bolo vidieť, kde každý zvuk začína. Všimnite si, že je tu len jedna premenná END, a to až na konci. Je to veľa ťukania a dôležite je to, že nesmiete urobiť chybu, inak to nebude fungovať. Akonáhle ste všetko zapísali, urobte si SAVE, pretože pri prípadnej chybe stratíte po RUN všetko!

```
      1 REM ......................
.....................................
.....................................
.....................................
.....sem zapíšte 432.....
.....................................
.............medzier.............
.....................................
.....t.j. okolo 15 riadkov.....
.....................................
...................
```

```
 10 LET address=PEEK 23635+256*PEEK 23636+5: LET end=1000
 20 READ byte
 30 IF byte>255 THEN STOP
 40 POKE address,byte
 50 LET address=address+1
 60 GO TO 20
100 REM BOMB FALLING
110 DATA 6,1,197,33,0,0,17,1
120 DATA 0,229,205,181,3,1,20,0
130 DATA 17,0,12,225,198,0,237
140 DATA 74,229,198,0,237,82
150 DATA 225,56,230,193,16,223
160 DATA 201
170 REM PHASOR FIRE
180 REM SOUNDS DATA LIST
190 DATA 6,1,197,33,0,0,17,1
200 DATA 0,229,205,181,3,1,1,0
210 DATA 17,100,1,225,198,0,237
220 DATA 74,229,198,0,237,82
230 DATA 225,56,230,193,16,223
240 DATA 201
250 REM REPEATED PHASOR FIRE
260 REM SOUNDS DATA LIST
270 DATA 6,10,197,33,0,0,17,1
280 DATA 0,229,205,181,3,1,1,0
290 DATA 17,100,1,225,198,0,237
300 DATA 74,229,198,0,237,82
310 DATA 225,56,230,193,16,223
320 DATA 201
330 REM RASPBERRY
340 REM SOUNDS DATA LIST
350 DATA 6,1,197,33,0,10,17,1
360 DATA 0,229,205,181,3,1,100,0
370 DATA 17,0,30,225,198,0,237
380 DATA 74,229,198,0,237,82
```

```
390 DATA 225,56,230,193,16,223
400 DATA 201
410 REM ALIEN MACHINERY OR UFO
420 REM SOUNDS DATA LIST
430 DATA 6,20,197,33,0,4,17,1
440 DATA 0,229,205,181,3,1,50,0
450 DATA 17,0,6,225,198,0,237
460 DATA 74,229,198,0,237,82
470 DATA 225,56,230,193,16,223
480 DATA 201
490 REN MYSTERIOUS SOUNDS
500 REM SOUNDS DATA LIST
510 DATA 6,5,197,33,0,10,17,10
520 DATA 0,229,205,181,3,1,0,2
530 DATA 17,0,19,225,198,0,237
540 DATA 74,229,198,0,237,82
550 DATA 225,56,230,193,16,223
560 DATA 201
570 REM ALARM
580 REM SOUNDS DATA LIST
590 DATA 6,10,197,33,0,0,17,100
600 DATA 0,229,205,181,3,1,0,1
610 DATA 17,0,3,225,198,0,237
620 DATA 74,229,198,0,237,82
630 DATA 225,56,230,193,16,223
640 DATA 201
650 REM LASER BEAM
660 REM SOUNDS DATA LIST
670 DATA 6,25,197,33,0,0,17,6
680 DATA 0,229,205,181,3,1,50,0
690 DATA 17,0,1,225,198,0,237
700 DATA 74,229,198,0,237,82
710 DATA 225,56,230,193,16,223
720 DATA 201
730 REM DRUM SYNTH-TYPE `PEEOW
740 REM SOUNDS DATA LIST
750 DATA 6,1,197,33,0,0,17,5
760 DATA 0,229,205,181,3,1,1,0
770 DATA 17,0,1,225,198,0,237
780 DATA 74,229,198,0,237,82
790 DATA 225,56,230,193,16,223
800 DATA 201
810 REM BIRD (OR SEEING STARS !!)
820 REM SOUNDS DATA LIST
830 DATA 6,14,197,33,0,0,17,40
840 DATA 0,229,205,181,3,1,25,0
850 DATA 17,240,0,225,198,0,237
860 DATA 74,229,198,0,237,82
870 DATA 225,56,230,193,16,223
880 DATA 201
890 REM ASCENDING TONE
900 DATA 6,1,197,33,232,3,17,1
910 DATA 0,229,205,181,3,225,1,
920 DATA 1,0,198,0,237,66,48
930 DATA 239,193,16,232,201
```



```
940 DATA 0,0,0,0,0,0,0,0,0
950 REM FASTER ASCENDING TONE
960 DATA 6,1,197,33,232,2,17,1
970 DATA 0,229,205,181,3,225,1,
980 DATA 10,0,198,0,237,66,48
990 DATA 239,193,16,232,201
999 DATA end
```

Po SAVE urobte RUN, aby sa nastavil strojový kód. To potrvá niekoľko sekúnd. Potom sa program zastaví so správou 9 STOP statenment,30:2. Aká rýchla kontrola, či je všetko OK, zadajte príkaz PRINT ADDRESS, čo má dať 24183. Tento príkaz (a jeho výsledok) platí len vtedy, ak nie je k počítaču nič pripojené (ako rozšírenie pamäte a pod.), inak totiž štartuje na inej adrese. Všeobecne, jediný úplný test je spustenie strojového kódu, čo je dôvod, prečo sme vám radili urobiť si najprv SAVE.

Než vymažete akákoľvek riadok, dopíšte tento testovací program:

```
10 LET address=PEEK 23635+256*PEEK 23636+5
11 FOR a=address TO address+423 STEP 36
12 RANDOMIZE USR a
13 PAUSE 20
14 NEXT a: STOP
```

ak všetko dobre prebehlo, potom budete počuť všetkých 12 zvukov s malými prestávkami medzi nimi a program sa zastaví normálne na riadku 14. Ak bolo všetko poriadku, potom vymažte všetky riadky, okrem riadku 1 REM a uchovajte pomocou SAVE riadok 1 REM na pásku. Toto je riadok, ktorý potom môžete pripojiť MERGEom k vašim budúcim programom.

Teraz ako vyvolať ten ktorý zvuk, keď ho potrebujete. Pokiaľ nemáte pripojené rozšírenie pamäti, potom stačí len zavolať odpovedajúcu adresu (však REM musí byť v pamäti).

Nasledujúca tabuľka vám povie, ako treba volať:

číslo zvuku	USR	zvuk
1	23760	Bomb falling
2	23796	Phasor fire
3	23832	Repeated phasor fire
4	23868	Raspberry
5	23904	Alien machinery/UFO
6	23940	Mysterious sounds
7	23976	Alarm
8	24012	Laser beam
9	24048	Drum-synthesiser "pesow" sounds
10	24084	Bird (or seeing stars!!)
11	24120	Ascending tone
12	24156	Faster ascending tone

Volanie týchto adries pravdepodobne nebude fungovať tým užívateľom, ktorí majú v pamäti pred priestorom pre program inicializovanú microdrive

mapu. Potom doporučujem volať príslušnú rutinu pomocou PEEK 23635/6 a k získanej hodnote pripočítať príslušný počet bajtov. Je to veľmi nepohodlné a tak bude lepšie vložiť FN pre výpočet adresy:

```
DEF FN u(n)=PEEK 23635+256*PEEK 23636+5+36*(n-1)
```

Vložte do riadku číslo 2 a odložte na pásku príkazom SAVE spolu s REM. Ak budete potrebovať kedykoľvek niektorý zo zvukov (napr. 6), napíšte do vlastného programu:

```
LET u=USR FN u(6)
```

Pretože zvuky budú používané pri hrách, je vhodnejšie označiť vrátenie z volania USR ako premennú, než používať RANDIMIZE USR, čo môže ovplyvniť náhodnosť náhodných čísiel.

Tu je príkad využitia USR FN:

```
 2 DEF FN u(n)=PEEK 23635+256*PEEK 23636+5+36*(n-1)
 3 REM Zvuk číslo n s USR FN u(n)
10 FOR a=1 TO 12
20 LET u=USR FN u(a)
30 NEXT a
```

Teraz nasleduje príklad na grafickú hru, ktorá tieto zvuky využíva. Asi zaznie zvuk poplachu, keď sa UFO objaví nad dvoma raketovými základňami. Pri jeho prvom prelete zničí jednu základňu bombami a pri druhom prelete bude UFO zostrelené. Je to chytené za vlasy, ale na príklad to stačí.

Bude to behať stále dokola, pokiaľ nestlačíme BREAK. Verím, že vám to dodá odvahu k vytváraniu vlastných zvukov.

```
 2 DEF FN u(n)=PEEK 23635+256*PEEK 23636+5+36*(n-1)
 3 REM Zvuk číslo n s USR FN u(n)
100 PRINT AT 15,12;"--T--";AT 15,20;"--T--": REM Základňa
105 PRINT AT 0,8; FLASH 1;"UFO prichádza"
110 LET r=USR FN u(7): REM Alarm
115 PRINT AT 0,8;"<15 medzier>"

120 FOR a=0 TO 30
130 PRINT AT 5,a;" >": REM Pred 'UFOm' je medzera!
140 IF a<5 THEN GO TO 180: REM Ďalšie
150 IF a=5 THEN LET r=USR FN u(12):  REM Bomba
160 IF a>5 AND a<14 THEN PRINT AT a,a-1;" ";AT a+1,a;"*"
170 IF a=14 THEN PRINT AT a,a-1;" ";AT 15,12; FLASH 1;
    OVER 1;" 5 medzier ": LET r=USR FN u(1): PRINT AT 15,12;
    "<5 medzier>"
180 NEXT a
185 PRINT AT 5,31;" "
186 PRINT AT 0,8; FLASH 1;"UFO prichádza"
187 LET r=USR FN u(7): REM Alarm
188 PRINT AT 0,8;" 15 medzier "
190 FOR a=0 TO 21
200 PRINT AT 5,a;" >": REM Pred UFOm je medzera!
210 NEXT a
```



```
220 PLOT 179,56: DRAW 0,75
225 LET r=USR FN u(8)
230 PLOT OVER 1;179,56:DRAW OVER 1;0,75
240 PRINT AT 5,22; FLASH 1;">"
250 LET r=USR FN u(4)
255 PAUSE 50
260 CLS: GO TO 100
```

PODROBNÝ POHĽAD DO VÁŠHO POČÍTAČA ZX SPECTRUM

Vydal: ULTRASOFT, spol. s r.o.
poštový priečinok, pošta 29
826 07 BRATISLAVA

V roku 1991 ako svoju 4. publikáciu
3. vydanie v roku 1993, náklad 1000 kusov

ISBN 80-85440-03-2