DIDAKTIK
KOMPAKT

Príručka užívateľa

DIDAKTIK KOMPAKT

Príručka užívateľa

© 1992
DIDAKTIK
Pod Kalváriou 22, 909 01 SKALICA

OBSAH

ČASŤ PRVÁ : BASIC

1. Popis jazyka BASIC 11
1.1. Prvé pokusy 11
1.2. Prvý program 16
1.3. Zavedenie nových pojmov 23
1.3.1. Číselné konštanty 23
1.3.2. Číselné premenné 25
1.3.3. Reťazcové konštanty 28
1.3.4. Reťazcové premenné 29
1.3.5. Indexované premenné (polia) 32
1.4. Použitie pri výpočtoch 35
1.5. Práca s funkciami 39
1.5.1. Základné funkcie 40
1.5.2. Matematické funkcie 47
1.6. Náhodné čísla 50
1.7. Rozhodovanie v programe 51
1.7.1. Logické operátory 55
1.8. Použitie cyklu v programe 59
1.9. Zadávanie dát počítaču 64
1.10. Doplnenie príkazu PRINT a INPUT 67
1.11. Práca s farbami 72
1.12. Počítač vie kresliť 80
1.13. Funkcie definované užívateľom 84
1.14. Čo je to podprogram 87
1.15. Práca s disketovou jednotkou 92
1.15.1. Príkaz FORMAT 94
1.15.2. Príkazy LOAD*, SAVE*, MERGE* 97
1.15.3. Príkaz CAT 104
1.15.4. Príkaz RUN 109
1.15.5. Príkaz ERASE 110
1.15.6. Príkaz LET FN 113
1.15.7. Príkaz MOVE 115
1.15.8. Príkazy RESTORE* a READ* 119
1.15.9. Príkaz LIST* 122
1.15.10. Príkaz LET ATTR 124
1.15.11. Príkazy OPEN# a CLOSE# 125

1.15.12. Príkaz PRINT* 129
1.15.13 Príkaz POKE# 130
1.15.14. Kláves SNAP 132
1.16. Práca s magnetofónom 134
1.17. Počítač meria čas 142
1.18. Počítač ako hudobník 145
1.19. Použitie tlačiarne 148
1.20. Vstupno/výstupné miesta počítača 149
1.21. Práca so strojovým kódom 151

ČASŤ DRUHÁ : OPERAČNÝ SYSTÉM

2. Operačný systém 157
2.1. Číselné sústavy 157
2.1.1. Desiatková sústava 158
2.1.2. Dvojková sústava 159
2.1.3. Šestnástková sústava 161
2.2. Vnútorná forma čísla 163
2.3. Mapa pamäti 165
2.3.1. Pamäť video RAM 166
2.3.1.1. Pamäť kresby 166
2.3.1.2. Pamäť atribútov 170
2.3.2. Pamäť programu a dát 173
2.3.2.1. Vyrovnávacia pamäť tlačiarne 175
2.3.2.2. Systémové premenné 175
2.3.2.3. Mapy mikrodrivov 175
2.3.2.4. Kanálové informácie 175
2.3.2.5. Program v BASICu 177
2.3.2.6. Premenné BASICu 179
2.3.2.7. Pracovné priestory BASICu 182
2.3.2.8. Editačná oblasť 183
2.3.2.9. Pracovný priestor, zásobník kalkulátora 184
2.3.2.10. Zásobník procesora Z80 185
2.3.2.11. Voľná pamäť 188
2.3.2.12. Umiestnenie užívateľskej grafiky 188
2.4. Znakový generátor 189
2.5. Užívateľská grafika (udg) 191
2.6. Funkcia USR, rutiny z ROM 192
2.7. Práca s V/V kanálmi 198

2.8. Práca s V/V miestami 202
2.8.1. Zákaznícky obvod ULA1 203
2.9. Interface 205

ČASŤ TRETIA : TECHNICKÝ POPIS

3. Technický popis 209
3.1. Centrálna procesorová jednotka (CPU) 209
3.2. Zákaznícky obvod ULA1 (U-106-47) 210
3.3. Pamäť ROM 211
3.4. Pamäť RAM 211
3.5. Radič floppy disku 212
3.6. Dekodér stavu zberníc 213
3.7. Klávesnica 213
3.8. TV , VIDEO a RGB 214
3.9. Magnetofón 214
3.10. Konektory pre pripojenie joystickov 215
3.11. Systémový konektor 216
3.12. Paralelný interface 217

Vážení zákazníci !

Otvorili ste príručku užívateľa mikropočítača Didaktik Kompakt, v ktorej sa podrobne zoznámite s Vaším počítačom. Príručka je rozdelená na tri časti.

V prvej časti je uvedený popis jazyka BASIC spolu s príkladmi použitia jednotlivých príkazov. Táto časť je určená hlavne pre začiatočníkov.

V druhej časti sú vysvetlené niektoré pojmy (číselné sústavy) a ďalšie podrobnosti, týkajúce sa práce operačného systému počítača i disketovej jednotky (mapa pamäti, systémové premenné,...).

V tretej časti je uvedený technický popis počítača.

Ako budeme označovať niektoré časti textu

V tejto príručke budeme používať nasledujúci dohovor pre označenie dôležitých častí textu:

tučné písmo	typ písma označujúceho, po prvý raz použité príkazy jazyka BASIC a dôležité nové pojmy
kurzíva	písmo použité pre označenie tej časti príkazu, ktorú môžete meniť
VEĽKÉ PÍSMO	označenie príkladov v texte
<Kláves>	označenie pre kláves

ČASŤ PRVÁ: BASIC

V tejto časti sú uvedené

- popis jazyka BASIC s príkladmi
- základy práce s disketovou jednotkou
- základy práce s magnetofónom
- základy práce s tlačiarňou
- základy práce v strojovom kóde

1. Popis jazyka BASIC

V tejto časti bude podrobne popísaný jazyk BASIC pre počítač Didaktik Kompakt. Tento popis môžete však využiť i pre všetky počítače kompatibilné s počítačom Sinclair ZX Spectrum (okrem príkazov pre disketu).

1.1. Prvé pokusy

Skôr než začnete pracovať, povedzme si niečo o zobrazovaní na obrazovke. Časť obrazovky, na ktorú je počítač schopný vypísať text, sa skladá z **24 riadkov**, každý riadok má **32 znakov**. Prvých 22 riadkov zhora sa používa pre výpis programu a pre štandardný výstup. Dolné 2 riadky sa používajú pre zadávanie príkazov, programových riadkov, **editovanie** (úpravu) programových riadkov, vstup dát a pre výpis hlásení. Táto dolná časť sa nazýva aj **dialógový riadok**. V prípade potreby sa dialógový riadok zväčšuje smerom hore. Po resetovaní počítača sa Vami zadávané údaje zobrazujú do dialógového riadku.

Čo musíte urobiť, aby ste napísali na hornú časť obrazovky Vaše meno? Počítaču musíte nejakým spôsobom oznámiť, čo od neho požadujete.

Pre výpis na obrazovku použijete kľúčové slovo **PRINT**. Ak zadáte cez klávesnicu:

```
PRINT "JAN NOVAK"
```

a stlačíte kláves <ENTER>, na hornú časť obrazovky sa vypíše text:

```
JAN NOVAK
```

Na dolnú časť obrazovky sa vypíše hlásenie:

```
0 OK, 0:1
```

čo znamená, že všetko je v najlepšom poriadku.

Ak zadáte počítaču:

```
PRINT "POD KALVARIOU 10"
```

a stlačíte kláves <ENTER>, počítač vypíše pod prvý výpis text:

POD KALVARIOU 10

Ďalej zadajte príkaz:

PRINT "SKALICA"

a stlačte kláves <ENTER>. Počítač pripíše k predchádzajúcim výpisom ďalší výpis:

SKALICA

Výpis v hornej časti obrazovky má tvar:

JAN NOVAK
POD KALVARIOU 10
SKALICA

Na týchto príkladoch sme ukázali, ako sa dá najjednoduchšie komunikovať s počítačom. Predchádzajúcu činnosť si podrobnejšie popíšeme.

Ak chcete na obrazovku vypísať ľubovoľný text, musíte pred text napísať slovo PRINT. Toto slovo znamená v angličtine vytlač, píš. Patrí do skupiny slov jazyka BASIC, ktoré sa nazývajú **príkazy**. Každý príkaz jazyka BASIC je tvorený anglickým názvom (alebo skratkou), ktorý určuje, akú činnosť má počítač vykonať. Príkaz PRINT je príkaz pre výpis na obrazovku. Text, ktorý chcete pomocou neho vypísať, napíšete za príkazové slovo PRINT a uzavriete ho úvodzovkami. Text sa vypíše na obrazovku od ľavého okraja časti obrazovky pre výpis. Všimnite si, že na obrazovke sa zobrazí iba text uzavretý v úvodzovkách; úvodzovky ani slovo PRINT sa nezobrazia.

Ak použijete ďalší príkaz PRINT, výpis sa zobrazí na ďalší riadok.

Doteraz ste počítaču zadávali iba jeden príkaz na spracovanie. Zadávaný riadok sa nazýva **príkazový riadok**. Po stlačení klávesu <ENTER> sa vykoná príkaz, ktorý príkazový riadok obsahuje. Príkaz sa však nevykoná, ak sa pri zápise príkazu dopustíte chyby.

Ak zadáte počítaču:

PRINT "PRACUJ

a stlačíte kláves <ENTER>, za písmenom J začne blikať **otáznik**, ktorý Vás upozorňuje na miesto v príkazovom riadku, kde počítač predpokladá chybu (chýbajú koncové úvodzovky). Po stlačení klávesu <ENTER> počítač vykoná analýzu príkazového riadku a v prípade chyby Vás "nepustí" ďalej. Chybu musíte opraviť a až keď je príkazový riadok napísaný bez chyby, je počítač ochotný vykonať príkaz v príkazovom riadku.

Príkazový riadok môže obsahovať aj viac príkazov. Príkazy sa v takom prípade oddeľujú od seba dvojbodkou. Hovoríme, že dvojbodka má funkciu **oddeľovača príkazov**.

Vytvorte príkazový riadok, ktorého výsledkom bude výpis mena a adresy. Príkazový riadok bude mať tvar:

```
PRINT "JAN NOVAK":PRINT "POD KALVARIOU 10":PRINT "SKALICA"
```

Po odoslaní príkazového riadku sa na obrazovke vypíše text:

```
JAN NOVAK
POD KALVARIOU 10
SKALICA
```

Vidíte, že výsledok je rovnaký, ako keď ste každý príkaz odoslali osobitne.

Aby sme mohli vysvetliť ďalšie vlastnosti príkazu PRINT, vysvetlíme si, ako pracuje pri výpise na obrazovku počítač. Ten musí v každom okamžiku "vedieť", na ktoré miesto obrazovky sa bude vypisovať ďalšie písmeno. Na to je určený takzvaný ukazovateľ výpisu pre príkaz PRINT. Pred prvým použitím príkazu PRINT je ukazovateľ nastavený na ľavý okraj prvého riadku obrazovky a nasledujúci výpis sa vykoná z tohto miesta. Miesto výpisu môžeme ovládať nastavením ukazovateľa výpisu. V predchádzajúcom príklade v príkazovom riadku nasledovala za textom dvojbodka. Pre počítač to značí, že na riadku obrazovky, na ktorom sa vypisoval text, nechcete už nič iné vypísať. Ukazovateľ sa nastaví na ľavý okraj nasledujúceho riadku obrazovky.

Vytvorme nasledujúci príkazový riadok:

```
PRINT "JANO MA ";:PRINT "OSEM ";:PRINT "ROKOV"
```

Počítač vypíše na obrazovku:

```
JANO MA OSEM ROKOV
```

Ak je za textom pre výpis bodkočiarka, ukazovateľ pre výpis sa nastaví za predchádzajúci text a nasledujúci text sa začne vypisovať od tohto miesta. Rovnaký výsledok dostanete aj v prípade, ak zadáte nasledujúci príkazový riadok:

```
PRINT "JANO MA ";"OSEM ";"ROKOV"
```

Vidíte, že v jednom príkaze PRINT môže byť viacej "textov", ktoré sú od seba oddelené bodkočiarkou. Hovoríme, že bodkočiarka je oddeľovač pre príkaz PRINT. Jej význam je ten, že nasledujúci text sa vypíše hneď za predchádzajúci text. Rovnaký výsledok dostaneme samozrejme aj príkazom:

```
PRINT "JANO MA OSEM ROKOV"
```

Zložitejší spôsob výpisu sme uviedli preto, aby sme ukázali význam bodkočiarky ako oddeľovača v príkaze PRINT.

Kvôli lepšej prehľadnosti je niekedy vhodné začínať výpis na čistú obrazovku. Na zmazanie obrazovky použijete príkaz **CLS**. Po vykonaní tohto príkazu sa obrazovka zmaže a nasledujúci výpis začína na prvom riadku obrazovky. Ak máte na obrazovke nejaký výpis a chcete na prvý riadok obrazovky vypísať text TABULKA, použijete príkazový riadok:

```
CLS:PRINT "TABULKA"
```

Ako ďalší oddeľovač môžete použiť čiarku. Vytvorte nasledujúci príkazový riadok:

```
PRINT "JANO","EMIL","FERO","EVA"
```

Po vykonaní vidíte, že medzi menami sú väčšie medzery. Obrazovka je rozdelená na tzv. **zóny**, každá zóna má 16 znakov (čiže v jednom riadku sú 2 zóny). Ak použijete čiarku ako oddeľovač, ukazovateľ sa nastaví na začiatok nasledujúcej voľnej zóny a nasledujúci text sa začne vypisovať z tohto miesta.
V prípade potreby sa začne výpis na nasledujúcom riadku, napr.:

```
PRINT "JANO, EMIL A FERO","EVA"
```

Prvý text sa začne vypisovať od začiatku riadku (prvá zóna), pretože však presahuje do druhej zóny, nasledujúci text sa vypíše do nasledujúcej voľnej zóny, t.j. do nasledujúceho riadku.

Príkazom PRINT môžete zobraziť aj čísla, ktoré nemusia byť uzavreté v úvodzovkách. Príkazom:

```
PRINT "VYSLEDOK JE ";5;" METROV"
```

sa na obrazovke zobrazí:

```
VYSLEDOK JE 5 METROV
```

V jednom príkaze PRINT môžete použiť viac druhov oddeľovačov, pričom môžu byť oddeľovače aj vedľa seba, napr.:

```
PRINT "JANO";8,,"JOZO"
```

Prvá čiarka za číslom 8 nastaví ukazovateľ na druhú zónu v riadku, ďalšia čiarka na ďalšiu zónu, t.j. na nasledujúci riadok.
Odošlite počítaču nasledujúci príkazový riadok:

```
PRINT "VYNECHAJ":PRINT:PRINT "RIADOK"
```

Počítač vypíše:

```
VYNECHAJ

RIADOK
```

t.j. medzi VYNECHAJ a RIADOK je jeden riadok voľný. Čo sa stalo?
Prvý príkaz PRINT vypíše VYNECHAJ a nastaví ukazovateľ na nasledujúci riadok.
Druhý príkaz PRINT nevypíše nič a pretože nie je ukončený čiarkou ani bodkočiarkou, nastaví ukazovateľ na nasledujúci riadok, čím vlastne jeden riadok vynechá.
Tretí príkaz PRINT vypíše RIADOK a nastaví ukazovateľ na nasledujúci riadok.

Ak chcete vynechať väčší počet riadkov, príkaz PRINT zadáte viackrát za sebou. V takom prípade je vhodnejšie využiť tvar príkazu PRINT s **apostrofom**. Apostrof v príkaze PRINT nastaví ukazovateľ na ďalší riadok. Príkaz:

```
PRINT ""
```

vynechá 4 riadky.

To je zatiaľ všetko o príkaze PRINT. Ďalšie možnosti ovládania výpisu na obrazovku sa dozviete v ďalších kapitolách.

1.2. Prvý program

Doteraz ste sa naučili, že po odoslaní príkazového riadku klávesom <ENTER> sa príkazy tohto riadku ihneď vykonali. Takýto režim práce sa označuje ako **priamy režim práce**. Výhodou tohto režimu je, že výsledky vidíte okamžite na obrazovke. Nevýhodou je, že počítač si nepamätá nič z toho, čo vykonal predtým.

Teraz sa naučíte vkladať údaje tak, aby si ich počítač zapamätal. Počítač si vkladané údaje uloží do pamäti a v prípade potreby vie uložené údaje použiť. Tým sa dostávame k **programovému režimu práce**, naučíte sa, čo je to program a ako sa zostavuje.

Zhruba môžeme povedať, že **program** v jazyku BASIC je postupnosť príkazov, ktoré sú usporiadané podľa určitých pravidiel (syntax jazyka). Tieto pravidlá sa musíme naučiť, lebo počítač ich pozná a riadi sa presne podľa nich. Každú odchýlku od syntaxe počítač spozná a obyčajne ohlási chybu výpisom správy do dialógového riadku.

Program sa skladá z riadkov programu (programových riadkov). **Programový riadok** je taký príkazový riadok, ktorý začína číslicou. Vložte tieto 2 riadky programu do počítača:

```
20 PRINT "tabulku"
10 PRINT "Hlavicka ";
```

a obrazovka vyzerá takto:

```
10 PRINT "Hlavicka ";
20 PRINT "tabulku"
```

Tieto riadky začali číslicami a preto neboli hneď vykonané, ale uchované ako riadky programu. Všimnite si, že čísla riadkov v programe sú zoradené. Je to dôležité pre jeho beh a tiež sa to prejaví vo výpise riadkov, ktoré teraz môžete vidieť na obrazovke. Teraz odošlite riadok:

```
15 PRINT "pre ";
```

Ak by boli riadky číslované s prírastkom jedna, nebolo by možné medzi dva napísané riadky už vložiť ďalší riadok. Preto je výhodné číslovať riadky namiesto 1, 2 , ... radšej 10, 20, ... avšak maximálne do 9999. Dodatočné vkladanie riadkov je dôvod, prečo číslovať riadky s krokom 10.
Teraz chceme zmeniť riadok 20 na:

```
20 PRINT "tabulku vysledkov"
```

Môžete to urobiť tak, že riadok 20 napíšete znova. Ak odošlete nový riadok s číslom 20, nový riadok prepíše starý.
Výhodnejšie je však využiť možnosť režimu <EDIT> (teda súčasné stlačenie klávesov <CAPS SHIFT> + <1>). Šípka > pri riadku 15 sa nazýva **programový kurzor** a riadok, na ktorý ukazuje, sa nazýva **bežný riadok**. Pri vkladaní programu je to posledný riadok, ktorý bol vložený, ale na nastavenie programového kurzora môžete použiť šípku dole alebo hore. Skúste si pohyb programového kurzora a nechajte ho nastavený na riadok 20. Keď stlačíte <EDIT>, objaví sa kópia bežného riadku v dialógovom riadku. Vo Vašom prípade je to kópia riadku 20. Pomocou šípky doprava presuňte kurzor L na koniec riadku pred úvodzovku a potom vložte text:

```
vysledkov
```

Riadok by mal teraz mať tvar:

```
20 PRINT "tabulku vysledkov"
```

Po stlačení <ENTER> sa nahradí starý riadok 20 a obrazovka vyzerá takto:

```
10 PRINT "Hlavicka ";
15 PRINT "pre ";
20 > PRINT "tabulku vysledkov"
```

Program máte uložený v pamäti. Na odštartovanie programu slúži príkaz **RUN** (kláves <R>). Zadajte RUN a stlačte <ENTER>.
Na obrazovku sa vypíše text:

Hlavicka pre tabulku vysledkov

Príkaz RUN zmaže obrazovku a odštartuje program od začiatku. Ale program môžete odštartovať od ľubovoľného riadku programu príkazom:

RUN *číslo riadku*

Tak po odoslaní príkazu:

RUN 20

dostanete na obrazovku výpis:

tabulku vysledkov

Ešte sa vrátime k režimu <EDIT>. Môžete ho použiť na vymazanie dialógového riadku. Niečo napíšte (bez <ENTER>) a potom sa rozhodnete, že to nechcete. Jedna z možností, ako to vykonať, je držať kláves <DELETE>, dokiaľ nie je riadok zrušený. Existuje však aj iná možnosť. Pokiaľ stlačíte <EDIT>, bude text v dialógovom riadku nahradený kópiou bežného riadku. Pokiaľ teraz stlačíte <ENTER>, bežný riadok sa uloží do programu bez zmeny a spodná časť obrazovky ostane voľná. Pokiaľ odošlete riadok s chybou, napr.:

12 PRINT "zly riadok ";

uloží sa tento riadok do programu a až potom si uvedomíte chybu. Pre vynechanie tohto riadku odošlite:

12

Teraz si s údivom všimnete, že programový kurzor zmizol. Môžete si predstaviť, že je niekde medzi 10. a 15. riadkom. Pokiaľ stlačíte šípku hore, bude sa nachádzať na riadku 10, ak stlačíte šípku dole, bude na riadku 15.

Odošlite:

12

Programový kurzor bude opäť medzi riadkami 10 a 15. Teraz stlačte <EDIT> a v dialógovom riadku bude kópia riadku 15. Keď je programový kurzor medzi riadkami , potom <EDIT> vyberie riadok s vyšším číslom. Stlačte <ENTER> na zmazanie dialógového riadku.

Potom odošlite:

30

Teraz je programový kurzor za koncom programu. Pokiaľ stlačíte <EDIT>, bude riadok 20 vypísaný v dialógovom riadku.
Doteraz vždy po stlačení <ENTER> sa vypísal program na obrazovku. Tento výpis nazvime automatický listing (listing je výpis programu). Existuje však špeciálny príkaz pre výpis programu na obrazovku a to príkaz

LIST

ktorý vypíše program od prvého riadku.
Ak chcete vypísať program od určitého riadku, zadáte číslo požadovaného riadku za slovo LIST v tvare:

LIST *číslo riadku*

Zadajte napr. príkaz:

LIST 15

a na obrazovke vidíte :

15 > PRINT "pre ";
20 PRINT "tabulku vysledkov"

Riadok 10 nie je zobrazený, ale je stále vo Vašom programe. Môžete si to overiť stlačením <ENTER>. Výsledkom príkazu LIST 15 je, že sa vykoná výpis programu začínajúci riadkom 15 a ďalej je na tomto riadku nastavený programový kurzor. Pokiaľ máte veľmi dlhý program, potom príkazom LIST nastavíte programový kurzor oveľa rýchlejšie ako šípkami.

Vložte do programu nasledujúci riadok:

```
5 REM hlavicka tabulky
```

a odštartujte program príkazom RUN.

Určite ste si všimli, že ste dostali ten istý výsledok, ako keď v programe tento riadok nebol. Príkaz **REM** je totiž nevýkonný. Ak sa dostane vykonávanie programu na príkaz REM, text za príkazom REM sa preskočí a pokračuje sa na ďalšom riadku. Preto môže byť za REM akýkoľvek text. Príkaz REM sa používa ako poznámka v programe, ktorou môžete označiť časti programu pre Vašu informáciu, na vykonávanie programu nemá žiadny vplyv.
Program môžete odštartovať aj príkazom **GO TO**, ktorý má tvar:

GO TO *číslo riadku*

kde *číslo riadku* má ten istý význam ako v príkaze RUN. Tým odštartujete program od zadaného čísla riadku. Napr.:

```
GO TO 20
```

Zdá sa Vám, že medzi príkazmi RUN a GO TO nie je žiadny rozdiel. V uvedenom programe je to pravda. Všeobecne to však neplatí. Príkaz GO TO vykonáva program od zadaného miesta, pričom berie do úvahy stav, v ktorom sa počítač nachádza. Príkaz RUN je nebezpečnejší, lebo pri jeho odoslaní sa vždy nastaví časť pamäti do počiatočného stavu , čím stratíte výsledky, ktoré má počítač uložené v pamäti, zmaže sa obrazovka atď. (podrobne sa to dozviete neskôr).
Ak v príkaze RUN a GO TO zadané číslo riadku v programe neexistuje, pokračuje vykonávanie programu na najbližšom vyššom riadku.

To, že príkaz GO TO ponecháva doterajšie výsledky v pamäti, je veľmi výhodné. Príkaz GO TO môžete totiž použiť priamo v programe, čím sa dostávame k pojmu programový cyklus. Je to časť programu, ktorá sa vykonáva opakovane. Doplňme program o nasledujúci riadok:

```
30 GO TO 10
```

a program odštartujte. Na celú obrazovku sa vypíše opakovane text:

hlavicka pre tabulku vysledkov

a v dolnom riadku sa objaví výpis:

scroll?

Čo sa vlastne stalo? Po prvom výpise riadku sa dostaneme na riadok 30, ktorý nás opäť vráti na riadok 10, nasleduje výpis a zase riadok 30 atď. Keď je na obrazovke zobrazených 22 riadkov, výpis sa zastaví a v dolnom riadku sa vypíše **scroll ?.**
Výpis sa zastavil preto, aby ste si mohli prezrieť výpis na obrazovke a výpisom scroll ? sa počítač pýta, či má pokračovať ďalej vo výpise, alebo má prerušiť program.
Ak stlačíte kláves <N>, kláves <SPACE> alebo kláves <BREAK> (<CAPS SHIFT> + <SPACE>), vykonávanie programu sa preruší a v dolnom riadku sa vypíše správa:

D BREAK - CONT repeats, 10:1

Ak stlačíte iný kláves, na obrazovku sa vypisujú ďalšie riadky tým spôsobom, že predchádzajúce riadky sa posúvajú smerom nahor a nasledujúci riadok sa vypisuje vždy na uvoľnené miesto dole.
Správa pri prerušení oznamuje, že program bol prerušený na riadku 10 v príkaze 1 (10:1) a v prerušenom programe je možné pokračovať príkazom

CONTINUE.

Zaujímavé je, že počas prerušenia programu môžete s počítačom vykonávať inú činnosť, napr. odoslať iné príkazy (PRINT), prezerať program príkazom LIST, ba dokonca je možné opravovať program v režime <EDIT>. Počítač si "pamätá", na ktorom mieste bol program prerušený a v programe môžete pokračovať príkazom CONTINUE. Skúste to !
V programe urobíte niekedy chybu a program beží v cykle bez výpisu na obrazovku, napr. program obsahuje tento riadok:

200 GO TO 200

Ak sa dostane vykonávanie programu na tento riadok (môžete ho odštartovať príkazom RUN 200), dostali ste sa do programového cyklu, z ktorého zdanlivo niet úniku. Na ukončenie činnosti stačí stlačiť kláves <BREAK>

(<CAPS SHIFT> + <SPACE>). Vykonávanie programu sa zastaví a v dolnom riadku sa zobrazí:

L BREAK into program, 200:1

čo znamená prerušenie programu klávesom <BREAK>. Pokračovať môžete opäť príkazom CONTINUE od toho miesta programu, kde bol prerušený. Kláves <BREAK> možno použiť kedykoľvek počas vykonávania programu. Jeho nevýhodou (a zároveň výhodou) je však to, že prerušenie môže nastať na ľubovoľnom mieste programu. Pri testovaní programu je vhodné zastaviť program na presne určenom mieste programu. Na to slúži príkaz:

STOP

Doplňte predchádzajúci program o riadok:

25 STOP

a odštartujte program. Po výpise textu sa objaví hlásenie:

9 STOP statement, 25:1

čo znamená, že program bol prerušený príkazom STOP. Ďalšia činnosť je podobná ako pri <BREAK>.

Ak ukončíte prácu s vytvoreným programom a chcete vytvoriť iný, potrebujete starý program zmazať. Mohli by ste to robiť rušením riadkov, ako ste sa to už naučili. V tomto prípade by to ešte išlo, ale u dlhých programov by ste sa "narobili". Preto je k dispozícii príkaz:

NEW

ktorý zmaže starý program. **Používajte ho však veľmi opatrne, lebo po jeho odoslaní je program beznádejne stratený !**

Videli ste použitie príkazov **PRINT, RUN, LIST, GO TO, CONTINUE, NEW** a **REM.** Všetky mohli byť použité v programe alebo v priamom režime. To platí takmer o všetkých príkazoch jazyka BASIC. Príkazy RUN, LIST, CONTINUE a NEW nie sú často používané v programoch, ale môžu byť použité.

1.3. Zavedenie nových pojmov

Zatiaľ ste využili počítač len na výpis textu na obrazovku. Počítač však dokáže pracovať s číslami, dokáže ich sčítať, násobiť, deliť a to veľmi rýchlo. Aby ste mohli tieto výhody využiť, musíte poznať niektoré pojmy, ako **číslo, reťazec, premenná, výraz** atď.

1.3.1. Číselné konštanty

Číselné konštanty sú dáta vyjadrené číslami. Môžu to byť čísla kladné, záporné, celé, desatinné. Vyjadrené môžu byť v bežnom alebo v takzvanom semilogaritmickom tvare. **Dôležité je, že u desatinných čísiel musíme písať namiesto desatinnej čiarky desatinnú bodku.** Ak zapisujete číselnú konštantu, ktorá nemá celú časť, nulu pred desatinnou bodkou nemusíte písať. Takisto nemusíte písať pred kladnou konštantou znamienko +.

Každé číslo je v počítači uložené na ohraničenej časti pamäti, preto nemôžete používať ľubovoľne veľké alebo malé čísla. Číslo môžete zadať na maximálne 8 platných číslic. Ak zadáte číslo, ktoré má viac ako 8 platných číslic, počítač toto číslo zaokrúhli na 8 platných číslic. Každé číslo, ktorého absolútna hodnota je v rozsahu od 0.00001 až 99999999 vypíše počítač v takzvanom pevnom formáte bez exponentu.

Všetky ostatné čísla sa vypisujú v **semilogaritmickom formáte**, ktorý má tvar:

SM.MMMMMMMESXX

kde jednotlivé písmená majú nasledovný význam:

S - znamienko mantisy alebo exponenta (vypisuje sa iba -)
M - číslica mantisy (0 až 9)
E - označenie exponenta písmenom e, E
X - číslica exponenta

Tento zápis znamená číslo:

SM.MMMMMMM krát 10 na SXX, t.j. mantisa krát 10 na exponent.

Uveďme si na ozrejmenie tohto pojmu niekoľko príkladov:

31 = 3.1x10 = 3.1E+1
123400000 = 1.234E+8
-0.0000089 = -8.9E-6

Rozsah čísiel, ktoré počítač umožňuje spracovať, je asi:

1.7×10^{38} až 1.7×10^{-38}

Nasledujúce príklady Vám ukážu rozdiel medzi správne a nesprávne zadaným číslom:

Správny zápis čísla	Nesprávny zápis čísla
5.3	5,3
-3.78	-3.7.8
0.01	E 02
36E02	5.31E53

V nasledujúcej časti si vysvetlíme pojem **premenná.** Počítač má k dispozícii určitý počet pamäťových miest, na ktoré môže uložiť hodnotu konštánt. Každé takéto miesto si môžete predstaviť ako nejakú schránku, do ktorej je možné vhodiť, alebo z nej možno vybrať lístok s ľubovoľnou konštantou. S obsahom týchto schránok môžete manipulovať za týchto podmienok:

a) každá schránka môže obsahovať iba jeden lístok
b) ak do schránky vložíme nový lístok, starý lístok tým zničíme (pokiaľ sme ho nepremiestnili do novej schránky)
c) obsah schránky môžeme prečítať bez toho, že by sme tento obsah zničili

Každú schránku môžeme označiť, dať jej meno. Podobne sa dá symbolicky označiť každé pamäťové miesto v počítači, do ktorého chcete nejaký obsah uložiť, ktorého obsah chcete zmeniť, resp. ktorého obsah chcete prečítať. S týmto označením môžete potom ďalej pracovať a budeme ho nazývať **premenná.**

Premenné je možné rozdeliť na skupiny podľa rôznych kritérií.

Podľa **obsahu** ich delíme na:
a) číselné (numerické)
b) reťazcové (alfanumerické)

Podľa **štruktúry** ich delíme na:
a) jednoduché
b) indexované (polia)

1.3.2. Číselné premenné

Sú to premenné, ktorých hodnotou je číslo. **Meno číselnej premennej môže byť zložené z písmen a číslic, prvé musí byť písmeno**. V názve premennej nie je rozdiel v použití malých a veľkých písmen. Dokonca kvôli prehľadnosti môžete použiť i medzery, ktoré sa však vnútorne nepoužívajú. Uveďme niekoľko prípustných mien:

x
a318
meno s medzerou
male a VELKE
maleavelke (rovnaké ako predchádzajúce)

Neprípustné sú napríklad nasledujúce mená:

100 (začína číslicou)
3 sudy (začína číslicou)
a*b88 (* nie je písmeno ani číslica)

Keď viete, ako sa označujú premenné, musíte sa naučiť priradiť premennej hodnotu. Na to je určený príkaz **LET**.
Ak chcete, aby číselná premenná *c* mala hodnotu 5, priradíte jej túto hodnotu príkazom

```
LET c=5
```

Za slovom LET je označenie premennej, ktorej chcete priradiť hodnotu, nasleduje znak = a za týmto znakom je číselná konštanta, ktorá sa priradí

premennej. Znak = v tomto prípade nie je symbolom označujúcim rovnosť, ale je **symbolom priradenia**. Preto je nesprávna interpretácia príkazu:

```
LET c=5
```

ako *c* rovná sa 5 (aj keď sa často používa), ale premennej *c* sa priradí hodnota 5. Podobne premennej *ad* priradíte hodnotu 32.45 príkazom:

```
LET ad=32.45
```

Pomocou známych príkazov môžete vytvoriť krátky program:

```
10 LET aa=82
20 LET b=2.73
30 LET c4=10E2
40 PRINT aa: PRINT b: PRINT c4
```

Po odštartovaní programu príkazom RUN vypíše počítač na obrazovku:

```
82
2.73
1000
```

Činnosť programu si pozrime bližšie.
Na riadku 10 sa premennej *aa* priradí hodnota 82.
Na riadkoch 20 a 30 sa premennej *b* priradí hodnota 2.73 a premennej *c4* hodnota 10E2 = 1000.
Na riadku 40 je použitý príkaz PRINT v tvare, v akom ho ešte nepoznáte. Ak v príkaze PRINT použijete označenie premennej, počítač vypíše hodnotu premennej, nie jej meno. To znamená, že počítač sa "pozrie" do pamäti na miesto, ktoré je označené ako *aa,* z tohto miesta zoberie číslo, ktoré je tam uložené (my sme tam uložili na riadku 10 hodnotu 82) a vypíše toto číslo. Takisto je to s premennými *b, c4.*
Všimnite si, že hodnotu premennej *c4* nevypísal počítač v tom tvare, ako sme ju zadali, ale v pevnom formáte.

Zadajte teraz príkaz:

```
GO TO 40
```

Počítač opäť vypíše na obrazovku hodnoty premenných.
Pre zmenu odštartujte program príkazom:

```
RUN 40
```

Počítač vypíše správu:

```
2 Variable not found, 40:1
```

lebo na riadku 40 hľadal počítač v pamäti premennú *aa*, ale nenašiel ju. Je to preto, lebo príkaz RUN najprv **zruší všetky premenné** a ak chcete potom s nejakou premennou pracovať, musíte jej najprv priradiť hodnotu (tým sa vytvorí v pamäti pre ňu zároveň miesto).
To isté sa stane, ak zrušíte riadok 10 a program odštartujete príkazom RUN.

Preto si zapamätajte, že každá premenná musí byť najprv definovaná (nemusí to byť len príkazom LET) a až potom s ňou môžete pracovať ! Toto platí pre všetky premenné, nielen pre číselné.

Ak chcete počas behu programu zrušiť všetky premenné, použijete príkaz **CLEAR**. Týmto príkazom sa zároveň zmaže obrazovka.
Hodnotu premennej môžete priradiť aj priamo z klávesnice, ak v programe použijete príkaz **INPUT**. Úplný popis tohto príkazu uvedieme neskôr, zatiaľ si ukážeme jeho použitie na príklade:

```
10 INPUT "Zadaj cislo :";b
20 PRINT b
30 GO TO 10
```

Príkaz INPUT vypíše do dialógového riadku správu:

```
Zadaj cislo :
```

a čaká, až zadáte číslo cez klávesnicu a odošlete ho stlačením klávesu <ENTER>. Počítač zadané číslo priradí premennej *b* (tým ju zároveň definuje). Na riadku 20 sa vypíše hodnota *b* a celý postup sa opakuje. Uvedený program môžete využiť pri zisťovaní, v akom tvare sa vypisujú čísla na obrazovku. Možný je aj tvar bez správy:

```
INPUT b
```

Analogicky sa dá priradiť hodnota reťazcovej premennej.

1.3.3. Reťazcové konštanty

Počítač si dokáže zapamätať nielen číslo, ale i iné postupnosti znakov, ktoré nemusia byť čísla. To ste videli pri vysvetľovaní činnosti príkazu PRINT, kde sme vypisovali text uzavretý v úvodzovkách. To nás privádza k pojmu **reťazcová koštanta**.
Je to ľubovoľná postupnosť znakov, ktorá sa chápe ako jeden celok. Táto postupnosť znakov je uzavretá v úvodzovkách. **Maximálna dĺžka** reťazcovej konštanty (počet jej znakov) je **255 znakov**. Uveďme si príklady :

```
"DIDAKTIK KOMPAKT"
" computer"
"56p : 45.s "
"5.78"
```

Počítač berie do úvahy všetky znaky medzi úvodzovkami, aj medzery. Poslednú konštantu môže chápať niekto ako číslo. Nie však počítač. Neskôr uvidíte, že počítač vie pracovať s číslami aj s reťazcami, ale s číslami vie robiť úplne iné operácie ako s reťazcami a naopak.
Podobný význam ako číslo 0 pre čísla má pre reťazce **prázdný reťazec**, ktorý sa označuje ako "" (t.j. dve úvodzovky vedľa seba).
Pretože reťazcové konštanty sa ohraničujú do úvodzoviek, je problém v označení konštanty, ktorá má úvodzovky obsahovať. Preto platí dohoda, že v takom prípade sa zadajú dve úvodzovky vedľa seba. Teda:

```
"a""b"
```

je konštanta obsahujúca znaky a"b.

Analogicky:

```
""""
```

je konštanta obsahujúca jednu úvodzovku.

1.3.4. Reťazcové premenné

Pre **reťazcové premenné** platí to isté, čo sme uviedli o číselných premenných - označujú nejaké miesto v pamäti počítača. V prípade reťazcovej premennej je na tomto mieste uložená reťazcová konštanta (t.j. konkrétna postupnosť znakov). **Meno reťazcovej premennej môže byť len jedno písmeno, za ktorým je znak $ (dolár)**. Tento znak rozlišuje meno reťazcovej premennej od mena číselnej premennej, musí byť vždy na konci mena reťazcovej premennej.

Správne mená reťazcových premenných sú:

a$, e$, k$

Názvy uvedených reťazcových premenných sa čítajú ako a-string, e-string, k-string.
Nesprávne mená reťazcových premenných sú

aa$, h3$, abeceda, x

Pri označovaní premenných môžete použiť pre číselné aj reťazcové premenné rovnaké meno (okrem znaku $ pre reťazcovú premennú),napr.:

a, a$, c, c$,

počítač ich chápe ako rozdielne premenné.

Hodnotu reťazcovej premennej priradíte príkazom LET:

```
LET a$ = "ulica"
LET b$ = "meno a priezvisko"
```

Na ľavej strane znaku = je meno reťazcovej premennej, na pravej strane je reťazcová konštanta, ktorej hodnota sa premennej priradí. Ak chcete vypísať hodnotu reťazcovej premennej, v príkaze PRINT zadáte meno tejto premennej:

```
10 LET a = 5:LET a$ = "Vysledna hodnota je "
20 PRINT a$;a
```

Po odštartovaní programu počítač vypíše:

```
Vysledna hodnota je 5
```

Pod pojmom **reťazec** budeme zatiaľ rozumieť reťazcovú konštantu alebo reťazcovú premennú, neskoršie tento pojem rozšírime.

Pokiaľ je daný reťazec, **podreťazec** je jeho súvislá časť. Tak sú "auto","mobil" podreťazce z "automobil", ale "util" nie je podreťazec z "automobil" (nie je to súvislá časť).

Ak chcete označiť podreťazec, použite nasledovný tvar:

reťazec (*začiatok* **TO** *koniec*) ,

kde *začiatok* a *koniec* označujú pozíciu znaku v *reťazci*.

Tak napr.:

```
"abcdef"(2 TO 5) = "bcde"
```

Keď vynecháte *začiatok*, očakáva sa, že je rovný 1, keď vynecháte *koniec*, potom je reťazec ponechaný do konca. Preto:

```
"abcdef"(TO 5) = "abcdef"(1 TO 5) = "abcde"
"abcdef"(2 TO) = "abcdef"(2 TO 6) = "bcdef"
"abcdef"(TO) = "abcdef"(1 TO 6) = "abcdef"
```

Posledný príklad môžete písať tiež ako "abcdef"( ).

Podobne platí:

```
"abcdef"(3) = "abcdef"(3 TO 3) = "c"
```

V uvedených príkladoch znak = značí rovnosť reťazcov.

Musí platiť, že začiatok aj koniec musí zodpovedať existujúcej časti reťazca. Výnimkou je, ak je začiatok väčší než koniec, výsledkom je prázdny reťazec. Napr.:

```
"abcdef"(5 TO 7)
```

dáva chybu:

```
3 Subscript wrong
```

ale:

```
"abcdef"(8 TO 7) = ""
"abcdef"(1 TO 0) = ""
```

Začiatok a *koniec* nesmú byť záporné čísla, inak dostaneme toto hlásenie o chybe:

```
B Integer out of range.
```

Výber podreťazcov si vyskúšajte na príkladoch nasledujúceho typu:

```
10 LET a$ = "abcdefgh"
20 PRINT a$(2 TO 4):PRINT a$(3)
```

Pre reťazcové premenné nemusíte len vyberať podreťazce, ale tiež ich meniť. Napr. zadajte:

```
LET a$ = "V praci nespi!"
LET a$(3 TO 7) = "skolenie"
PRINT a$
```

Všimnite si, že podreťazec *a$*(3 TO 7) je dlhý len 5 znakov, preto bolo použitých len prvých 5 znakov z reťazca "skolenie". To je charakteristické pre zmenu podreťazcov, podreťazec musí mať presne rovnakú dĺžku ako mal predtým. Z dlhšieho reťazca sa zoberie len potrebný počet znakov, kratší sa doplní sprava medzerami.
Presvedčte sa, že je to tak !
Z reťazcov môžeme nielen vyberať podreťazce, ale reťazce môžeme aj **spájať** pomocou znaku + . Platí:

```
"abc" + "def" = "abcdef"
```

Na komplikované reťazcové vyjadrenie budete potrebovať zátvorky, pretože z nich môže byť ďalej vyberané. Platí:

```
"abc" + "def"(1 TO 2) = "abcde"
("abc" + "def")(1 TO 2) = "ab"
```

1.3.5. Indexované premenné (polia)

Dôvodom pre zavedenie **indexovaných premenných** je snaha uľahčiť prácu s dátami. Vysvetlime si, čo to vlastne je.
Chceme označiť každý deň v roku. Môžeme to urobiť takto:

d(1), *d*(2), *d*(3),...,*d*(365)

Použili sme označenie písmenom *d*, za ktorým je v okrúhlych zátvorkách uvedené poradové číslo dňa v roku. Číslu v zátvorkách budeme hovoriť **index**. Jednotlivé označenia dní sa líšia od seba len hodnotou indexov, písmeno *d* je spoločné.
Podobne by sme mohli označiť dni v roku tak, že v zátvorke budú dve čísla oddelené čiarkou. Prvé číslo označuje mesiac, druhé deň v mesiaci. Napr. *d*(5,20) označuje 20-ty deň v piatom mesiaci.
Hovoríme, že indexované premenné, ktoré majú to isté označenie, tvoria **pole**. Jednotlivé indexované premenné z tohto poľa nazývame **prvkami** tohto poľa. Každý prvok poľa je určený menom poľa a svojou pozíciou v poli, pozícia je určená hodnotou indexov prvku.
Meno poľa môže byť len jedno písmeno, za meno sa do okrúhlych zátvoriek udávajú rozmery jednotlivých dimenzií (t.j. maximálne hodnoty indexov, ktoré môžu byť pri označení prvkov tohto poľa použité). Indexy môžu byť len celé čísla väčšie ako 0 (1,2,3,..), oddeľujú sa od seba čiarkou.
Správne označené indexované premenné sú napríklad:

b(3,3), y(10)

Nesprávne označené indexované premenné sú napríklad:

q(4;2), c(1,0), d(-1,-1).

Ak chcete použiť pole dát, musíte počítaču oznámiť, ako sa bude pole volať a aké bude mať rozmery. Na to sa používa príkaz **DIM**.
Tento príkaz musíte použiť pred prvým použitím prvku tohto poľa (nikdy nepracujeme s poľom ako s celkom, ale vždy iba s prvkom poľa). Príkazom DIM sa rezervuje v pamäti počítača miesto pre jednotlivé prvky:

```
10 DIM a(10)
20 DIM b(4,5)
```

Príkaz DIM na riadku 10 rezervuje v pamäti miesto pre číselné pole, ktoré má 10 prvkov *a*(1), *a*(2),...,*a*(10) a hodnoty prvkov sú nastavené na 0. Zároveň zruší akékoľvek pole *a*, ktoré už bolo definované.
Príkaz DIM na riadku 20 rezervuje v pamäti miesto pre číselné pole, ktoré má 4x5 prvkov:

b(1,1), b(1,2),...,b(1,5)
b(2,1), b(2,2),...,b(2,5)
.
.
.
b(4,1), b(4,2),...,b(4,5)

Počet indexov ani veľkosť jednotlivých indexov nie sú obmedzené. Obmedzenie je však v tom, že pole sa musí do voľnej pamäti zmestiť. Po zadaní príkazu:

DIM c(100,100)

ohlási počítač chybu:

4 Out of memory

čo značí, že pre pole *c* nie je v pamäti dosť miesta.

Upozornenie: V jednom príkaze DIM môže byť uvedené len jedno pole.

Existujú tiež **reťazcové polia**. Reťazec a reťazcové pole sa od seba líšia tým, že pole má konštantnú dĺžku a priradenie je vykonané tak, že zvyšok dĺžky, ktorý nie je obsadený, je nahradený medzerami. Môžeme mať tiež pole jednotlivých znakov (ale s jedným ďalším rozmerom). **Meno reťazcového poľa je jedno písmeno a znak $.** Reťazec a reťazcové pole nesmú mať navzájom zhodné meno (na rozdiel od polí číselných). Nemôžeme použiť preto súčasne premenné:

a$, a$(1,1)

Predpokladajte, že chcete pole *a$* s piatimi reťazcami. Musíte sa rozhodnúť, ako dlhé budú tieto reťazce. Predpokladajte, že 10 znakov postačí. Pole definujte príkazom:

DIM a$ (5,10)

Tak je definované pole 5x10 znakov, ale môžete tiež uvažovať o rade znakov jednotlivých:

```
a$(1) = a$(1,1)a$(1,2) ... a$(1,10)
a$(2) = a$(2,1)a$(2,2) ... a$(2,10)
. . . .
. . . .
a$(5) = a$(5,1)a$(5,2) ... a$(5,10)
```

Teda, keď zadáte 2 indexy, potom dostanete jeden znak, ale keď nenapíšete druhý index, dostanete reťazec definovanej dĺžky, kde dĺžka reťazca je daná druhou položkou v príkaze DIM. Tak napr. *a$(2,7)* je siedmy znak reťazca *a$(2)*. Mohli by sme vlastne tiež písať *a$(2)(7)*. Skúste:

```
LET a$(2) = "1234567890"
PRINT a$(2),a$(2,7)
```

a dostanete:

```
1234567890      7
```

Pre posledný index (ten, ktorý môžeme vynechať) možno použiť techniku pre výber časti reťazca:

```
a$(2,4 TO 8) = a$(2)(4 TO 8) = "45678"
```

Pamätajte: V reťazcovom poli majú reťazce rovnakú dĺžku. Posledný index príkazu DIM určuje dĺžku každého reťazca.

Po zadaní:

```
DIM a$(10)
```

uvidíte, že *a$* sa chová rovnako ako reťazcová premenná s výnimkou, že má dĺžku 10 a po priradení hodnoty je zvyšok doplnený medzerami.

1.4. Použitie pri výpočtoch

Dostávame sa konečne k tomu, prečo sa počítač volá počítačom. Dokáže totiž veľmi rýchlo počítať, presnejšie povedané dokáže vykonávať **matematické operácie**. Uveďme si príklad:

```
10 LET a=3: LET b=4
20 LET c=a+b
30 PRINT c
```

Čo bude výsledkom programu?
Na riadku 10 sme priradili hodnotu premenným *a*, *b*.
Na riadku 20 počítač sčíta hodnoty premenných *a*, *b* a výslednú hodnotu priradí premennej *c* (t.j. $c=3+4=7$).
Na riadku 30 počítač vypíše hodnotu premennej *c*, t.j. číslo 7.

Uveďme si ďalší príklad:

```
10 LET x=4
20 LET x=x+1
30 PRINT x
```

Niekomu sa môže zdať, že na riadku 20 je napísaný nezmysel, lebo nemôže platiť rovnosť $x=x+1$. To je pravda. Ide o to, že znamienko = nevyjadruje rovnosť, ale **priradenie hodnoty premennej**.

Počítač pracuje takto:

Na riadku 10 sa premennej *x* priradí hodnota 4.
Vypočíta , čomu sa rovná $x+1$. Pretože *x* bolo 4, bude $x+1=5$.
Výslednú hodnotu 5 priradí počítač premennej *x*.
Vypíše sa hodnota premennej *x*, t.j. číslo 5.

Pri počítaní môžete čísla nielen sčítať, ale i odčítať, násobiť, deliť a umocňovať.
Ak chcete sčítať dve čísla, dáte medzi ne znamienko +, ak ich chcete odčítať, dáte medzi ne znamienko −.
V matematike je zvykom dávať pri násobení medzi čísla znak x alebo bodku, pri delení dvojbodku.

Pri práci s počítačom je to inak. Odteraz budete pri počítaní používať tieto znaky:

\+ pre sčítanie
– pre odčítanie
* pre násobenie
/ pre delenie
^ pre umocňovanie

Značky +, –, *, /, ^ budeme nazývať **aritmetické operátory**. Operátor sa musí písať vždy, nemožno ho vynechať. Keby sme napr. namiesto *b=a*8* napísali *b=a8*, pre počítač to znamená, že premennej *b* má priradiť hodnotu premennej *a8* (vynechaním * sme dostali meno premennej).
Spomenuli sme operáciu **umocňovania**. Pre tých, ktorí nevedia, čo to je, nasleduje malé vysvetlenie.
Umocňovať číslo na mocninu znamená násobiť číslo samo sebou toľkokrát, koľko určuje mocnina. V matematike sa mocnina píše nad prvé číslo; pre počítač je to obťažné a tak používame symbol ^ (<SYMBOL SHIFT> + <H>):

2 ^ 1 =2
2 ^ 2 = 2*2 = 4
2 ^ 3 = 2*2*2 = 8
2 ^ 4 = 2*2*2*2 = 16

Teda *x* ^ *y* znamená násobiť výraz *x* *y*-krát. Obvykle to má zmysel, keď *y* je celé kladné číslo. Uvedieme však takú definíciu, že to pracuje pre ďalšie hodnoty b. Uvažujme pravidlo:

a ^ (b+c) = (a ^ b) * (a ^ c)

Nemusíme uvažovať, že *b* a *c* sú celé kladné čísla, nasledujúce rovnosti platia obecne:

a ^ 0 = 1
a ^ (–b) = 1/(a ^ b)
a ^ (1/b) = b-ta odmocnina z a , je to číslo, ktoré musíme násobiť b-krát, aby vzniklo a.
a ^ (b * c) = (a ^ b) ^ c

Pokiaľ ste nič z tohoto skôr nevideli, nemusíte si to pamätať. Stačí, ak budete zatiaľ vedieť:

a ^ (−1) = 1/a

V počítačovej terminológii je základným pojmom pojem **aritmetický výraz**. Čo to je ?
Aritmetický výraz je kombinácia konštánt, premenných a funkcií spojených prípustnými aritmetickými operátormi (s funkciami sa zoznámime v ďalšej kapitole tejto príručky).
Aritmetický výraz môže obsahovať viac operátorov, ktoré spájajú viac konštánt, premenných a funkcií (**operandov**).
Príklady aritmetických výrazov:

a+b−c
(a+b)*d
(e+f)^4−e^2

Ak obsahuje aritmetický výraz viac operátorov aj operandov, hodnota výrazu sa vyhodnocuje postupne podľa **priority** (prednosti) jednotlivých operátorov.

Priorita aritmetických operátorov je táto:

1. ^ umocnenie
2. *, / násobenie a delenie
3. +, − sčítanie a odčítanie

Najvyššiu prioritu má umocňovanie, najnižšiu sčítanie a odčítanie. Pri rovnakej priorite sa operácie vykonávajú zľava doprava. Ak chcete poradie vykonávania zmeniť, použijete v aritmetickom výraze okrúhle zátvorky. Zátvorkové páry môžete vkladať do seba. Vtedy sa vyhodnotí najprv výraz v najvnútornejších zátvorkách, potom v druhých najvnútornejších atď. Zátvorky majú najvyššiu prioritu.
Ukážme si, ako sa vyhodnotí výraz:

aa+(b−c)*d^2

Počítač spracuje výraz v týchto krokoch:

1. $b-c$
2. $d \wedge 2$
3. výsledok z 1. je vynásobený výsledkom z 2.
4. výsledok z 3 je pripočítaný k *aa*

Ak neviete presne, v akom poradí vyhodnotí počítač aritmetický výraz, použite zátvorky! Tým sa stane výraz aj prehľadnejší.

Uveďme si príklady aritmetických výrazov a ich zápis v jazyku BASIC:

Aritmetický výraz	Zápis v jazyku BASIC
$\frac{a+b}{c}-d$	(a+b)/c−d
$\frac{a+b}{c-d}$	(a+b)/(c−d)
$a+\frac{b}{c*d}$	a+b/(c*d), a+b/c/d
$\frac{a*c}{b}+d$	a/b*c+d, a*c/b+d, (a*c)/b+d
$a+b*c^{d}$	a+b*c ^ d

1.5 Práca s funkciami

Uvažujte stroj na výrobu klobás. Na jednom konci vložíte kus mäsa, pohnete pákou a na druhom konci vypadne klobása. Z bravčového sú bravčové, z rýb rybie, z hovädzieho hovädzie. Od stroja na klobásy sa počítač líši tým, že pracuje s číslami a reťazcami namiesto mäsa. Vložíte jednu hodnotu (nazvanú **argument**), urobíte nejakú **operáciu** a dostanete ďalšiu hodnotu **(výsledok)**:

MÄSO ..... STROJ NA KLOBÁSY ..... KLOBÁSY

Podobne pracujú **funkcie**:

ARGUMENTY ..... FUNKCIE ..... VÝSLEDOK

Rôzne argumenty dávajú rôzne výsledky a pokiaľ je argument úplne nevhodný, funkcia sa nevykoná a počítač hlási chybu.

Rovnako ako existujú rôzne stroje, jeden na klobásy, druhý na oblečenie, tretí na konzervy, budú rôzne funkcie rôzne počítať. Každá bude mať vlastné meno, ktoré ju odlíši od ostatných. Funkciu použijete tak, že napíšete meno funkcie, za ktorým nasleduje argument funkcie.

Pri riešení matematických problémov sa používajú často všeobecne známe matematické funkcie. Počítač vie s týmito funkciami tiež pracovať, vie vypočítať ich hodnoty.

V tejto kapitole sa budeme zaoberať **štandardnými** (vstavanými) funkciami, ktoré môžete použiť v jazyku BASIC, aj keď niektoré funkcie ešte nepoznáte. **Každá štandardná funkcia je označená menom, za ktorým sa uvádza argument, ktorý môže (ale nemusí) byť uzavretý v okrúhlych zátvorkách.**

Dohodneme sa teraz, že ak to nebude v konkrétnom prípade povedané inak, na mieste, kde v popise funkcie alebo príkazu môže byť číslo, môže byť tiež aritmetický výraz. Ak sa požaduje celé číslo, z hodnoty výrazu sa automaticky berie celá časť.

Analogicky na mieste reťazca môže byť výraz, ktorého hodnota je reťazec.

1.5.1. Základné funkcie

Ako prvú uvedieme funkciu **LEN**, ktorá určuje **dĺžku reťazca**. Jej argumentom je reťazec, ktorého dĺžku chcete zistiť.
Napr. zadajte:

```
PRINT LEN "Skalica"
```

a počítač napíše 7, čo je počet písmen reťazca (na získanie LEN, ako u väčšiny funkčných mien, musíte použiť rozšírený mód klávesnice a potom stlačte kláves). Pokiaľ spojíte funkcie pomocou operátorov do jedného výrazu, potom budú funkcie spracované pred operáciami. Toto pravidlo môžete samozrejme meniť použitím zátvoriek. Napr. sú tu dva výrazy, ktoré sa rozlišujú len v zátvorkách a operácie sa v každom prípade vykonávajú v inom poradí (hoci sú výsledky rovnaké):

```
LEN "Jano" + LEN "Sevcik"          LEN ("Jano" + "Sevcik")
4 + LEN "Sevcik"                   LEN ("JanoSevcik")
4 + 6                              LEN "JanoSevcik"
10                                 10
```

Ďalšou funkciou je funcia **STR$**, ktorá **konvertuje (prevádza) čísla na reťazce**; argumentom je číslo a výsledkom je reťazec.
Skúste napísať:

```
LET a$ = STR$ 1e2
```

čo má ten istý význam ako:

```
LET a$ = "100"
```

Ak zadáte:

```
PRINT LEN STR$ 100.0000
```

a dostanete odpoveď 3, lebo STR$ 100.0000 = "100"
Funkcia **VAL** má opačný význam ako STR$, mení **reťazec na číslo**. Napr.:

```
VAL "3.5" = 3.5
```

Funkcia VAL je **obrátená** k funkcii STR$, pretože ak máte nejaké číslo, aplikujete naň STR$ a potom ešte VAL, výsledkom je pôvodné číslo. Pokiaľ sa však vezme reťazec a v ňom sa aplikuje VAL a potom STR$, nie vždy dostaneme pôvodný reťazec. VAL je veľmi mocná funkcia, pretože reťazec, ktorý je argumentom, nie je obmedzený len na číslo, ale môže to byť i aritmetický výraz. Napr.:

```
VAL "2*3" = 6 alebo dokonca
VAL ("2" + "*3") = 6
```

Tu sú spojené dve dôležité veci.
Po prvé argument VAL je spracovaný ako reťazec. Reťazcové vyjadrenie "2" + "*3" je najprv spojené do reťazca "2*3". Potom je z reťazca zostavený výraz, ako keby nebolo úvodzoviek, je vynásobené 2*3 s výsledkom 6.
Teraz sa Vás pokúsime zmiasť:

```
PRINT VAL "VAL""VAL""""2""""""""
```

(Pamätajte, že vnútorné úvodzovky v reťazci musia byť písané dvojito). Čo bude výsledkom tohto príkazu?

Existuje podobná funkcia ako VAL, ale asi menej používaná, nazýva sa **VAL$**. Jej argumentom je stále reťazec, ale výsledok je takisto reťazec. Táto funkcia funguje ako VAL v dvoch krokoch: najprv je argument chápaný ako reťazec, potom sú odstránené úvodzovky a všetko je chápané ako číslo. VAL$ funguje rovnako, ale výsledok je považovaný opäť za reťazec. Preto:

```
VAL$ """"dva""""  = "dva"
```

Teraz zadajte:

```
LET a$ = "99"
```

a vypíšte hodnoty nasledovných výrazov:

```
VAL a$ , VAL "a$", VAL """"a$"""",
VAL$ a$ , VAL$"a$", VAL$""""a$""""
```

Niektoré z týchto príkladov budú pracovať, niektoré nie. Skúste nájsť odpoveď, prečo.

Funkcia **INKEY$** (nemá argument) **číta klávesnicu**. Pokiaľ stlačíte jeden kláves (alebo <SHIFT> + jeden kláves), potom výsledkom je znak, ktorý dostanete stlačením tohto klávesu v L-móde. Ak nie je stlačený žiadny kláves, výsledkom je prázdny reťazec.
Nasledujúci program pracuje ako písací stroj:

```
10 IF INKEY$< > ""THEN GO TO 10
20 IF INKEY$ = "" THEN GO TO 20
30 PRINT INKEY$ ;
40 GO TO 10
```

Na riadku 10 sa čaká, až dáte prst preč z klávesnice a na riadku 20 sa čaká, až stlačíte nový kláves. Pamätajte, že na rozdiel od príkazu INPUT funkcia INKEY$ nevyžaduje potvrdenie klávesom <ENTER>.
Porozmýšľajte čo sa stane, ak vypustíte riadok 10 v poslednom programe!

Ďalší spôsob použitia INKEY$ je využitie s **PAUSE:**

```
10 PAUSE 0
20 PRINT INKEY$ ;
30 GO TO 10
```

A do tretice:

```
10 IF INKEY$ ="" THEN GO TO 10
20 PRINT AT 11,14;"jaj !"
30 IF INKEY$< > "" THEN GO TO 30
40 PRINT AT 11,14;" "
50 GO TO 10
```

Písmená, číslice a ďalšie symboly, ktoré sme používali v reťazcoch, sa nazývajú znaky a tvoria abecedu alebo **systém znakov**, ktoré používa počítač. Väčšina týchto znakov sú jednoduché symboly, ale tiež symboly, ktoré reprezentujú kľúčové slová jazyka BASIC (PRINT, STOP,...). Existuje celkom 256 znakov, ktoré majú **kód** od 0 do 255 (kód je číslo, pod ktorým je znak vnútorne uložený v počítači). Ich úplný zoznam je uvedený v prílohe.

Prevod medzi znakmi a ich kódmi umožňujú funkcie **CODE** a **CHR$.** Majú tvar:

CODE *reťazec*
CHR$ *číslo*

Funkcia CODE je aplikovaná na *reťazec,* jej výsledkom je **kód prvého znaku** v *reťazci* alebo 0, pokiaľ ide o prázdny reťazec.

Funkcia CHR$ je aplikovaná na *číslo* a pre *čísla* 32 až 255 je jej výsledkom znak, ktorého kód je daný argumentom funkcie:

```
CODE "ABC" = 65 , lebo kód "A" je 65
CODE "B" = 66 , lebo kód "B" je 66
CHR$ 65 = "A"
CHR$(3*22) = "B"
```

Nasledujúci program vypisuje na obrazovku všetky znaky s kódom 32 až 255:

```
10 FOR a = 32 TO 255: PRINT CHR$ a;: NEXT a
```

Výpis môžno rozdeliť na dve časti. Prvá časť sú znaky s kódom po hodnotu 127. Všetky tieto znaky (okrem dvoch) sú z kódu známeho ako **ASCII** (tento kód sa používa u väčšiny mikropočítačov).

Zvyšok znakov nie je časťou kódu ASCII a je použitý v počítačoch typu ZX Spectrum. Prvý medzi nimi je medzera a 15 čiernobielych vzorov. Sú to grafické symboly a používajú sa na kreslenie obrázkov. Môžete ich vkladať z klávesnice použitím **grafického módu**. Keď stlačíte GRAPHICS (< CAPS SHIFT + < 9 >), kurzor sa zmení na **G.** Teraz stlačte číslice < 1 > až < 8 > a budú sa objavovať grafické symboly, ktoré sú zobrazené na klávesoch, so stlačením < SHIFT > dávajú invertovaný symbol, tj. čierne sa stáva bielym a naopak. V grafickom móde, bez ohľadu na iné klávesy, kláves < 9 > nastavuje späť L-mód a kláves 0 funguje ako < DELETE >.

Po grafických symboloch potom uvidíte kópie abecedy od A do U. Tieto znaky môžete modifikovať sami. Keď je počítač prvý raz zapnutý, sú k dispozícii len znaky základné. Sú to takzvané znaky **užívateľsky definovanej grafiky** (udg), ktoré sa vkladajú z klávesnice v grafickom móde G použitím písmen od A do U. Postup definovania užívateľských znakov bude uvedený neskôr.

Po užívateľských znakoch nasledujú kľúčové slová jazyka BASIC.

Mohli ste si všimnúť, že sme netlačili prvých 32 znakov s kódmi 0 až 31. Sú to **riadiace znaky**. Nevytvárajú nič, čo sa dá tlačiť, ale majú vplyv pri výpise na obrazovku alebo sú využité na riadenie iných vecí. Obrazovka používa 3 znaky s kódmi 6, 8 a 13.

CHR$ 6 : nastavuje ukazovateľ pre výpis na začiatok ďalšej zóny, má ten istý význam ako čiarka vo význame oddeľovača v príkaze PRINT.

Porovnajte výsledok týchto dvoch príkazov:

```
PRINT 1; CHR$ 6;2
PRINT 1,2
```

Výhodné je využitie, ak tento znak vložíte priamo do reťazca:

```
LET a$ = "1" + CHR$ 6 + "2"
PRINT a$
```

CHR$ 8 : posúva tlačovú pozíciu o jeden znak späť (je to vlastne <DELETE>). Skúste napísať:

```
PRINT "1234"; CHR$ 8;"5"
```

Výsledkom je výpis 1235, lebo najprv sa vypíše 1234, potom sa ukazovateľ pre výpis nastaví späť a znak 4 sa prepíše znakom 5.

CHR$ 13 : posúva ukazovateľ výpisu na začiatok ďalšieho riadku.
Využitie ďalších riadiacich znakov bude uvedené neskôr.

Použitím kódov a funkcie CHR$ môžete vytvoriť reťazce s ľubovoľnými znakmi, nielen s písmenami. Počet všetkých znakov je 256.
Tu sa dostávame k pojmom **porovnanie reťazcov, usporiadanie reťazcov**. Nasledujúce reťazce sú usporiadané vzostupne:

```
CHR$ 3 + "ZOOLOGICKA ZAHRADA"
CHR$ 8 + "ABECEDA"
" AAAARGH!"
"(okruhle zatvorky)"
"100"
"129.95"
```

"AAAARGH!"
"PRINT"
"Zoo"
"aaaargh!"
"zoo"
"zoologia"

Všimnite si, že veľké písmená sú pred malými.

Reťazce sú usporiadané podľa nasledujúcich pravidiel:

Znaky sa berú postupne od začiatku a porovnávajú sa ich kódy. Naprv sa porovnajú prvé znaky. Pokiaľ sú rozdielne, potom jeden má kód menší než druhý a reťazec, ktorého znak má nižší kód je menší. Keď sú prvé znaky rovnaké, porovnávajú sa rovnakým spôsobom znaky ďalšie a reťazce sú zhodné, keď sú si všetky znaky rovné.

Relácie = , < , > , < = , > = , < > sú použiteľné pre reťazce i pre čísla. Pre reťazce < znamená "je pred" a > znamená "je za".
Nasledujúce výrazy sú pravdivé:

"AA man" < "AAAA"
" AAAA" > "AA man"

Relácie < = , > = pracujú podobne ako s číslami:

"Rovnaky retazec" < = "Rovnaky retazec" je pravdivé
"Rovnaky retazec" > "Rovnaky retazec" je nepravdivé

Odštartujte tento program:

```
10 INPUT a
20 PRINT CHR$ a;
30 GO TO 10
```

Pokiaľ budete experimentovať zistíte, že ak nie je zadané číslo v rozsahu 0 až 255, program sa zastaví s hlásením:

B Integer out of range

Niekomu je čudné, prečo je rozsah práve 0 až 255. Tieto hodnoty sú dané technickými prostriedkami počítača. Podrobnejší popis bude uvedený v druhej časti príručky. Zatiaľ si povieme to, že pamäť počítača sa skladá z "buniek", táto bunka sa nazýva **byte** (čítaj bajt) a zmestí sa do nej celé číslo od 0 do 255. Každý znak je uložený v jednom byte, preto kód znaku je od 0 do 255.
Každý byte má svoju **adresu,** čo môže byť celé číslo od 0 do 65535. Každý byte sa skladá z 8 menších častí, táto časť sa nazýva **bit** (bit je skratka z anglického binary digit a znamená dvojková číslica). Jeden bit môže mať hodnotu 0 alebo 1.

Kto sa vyzná v dvojkovej sústave, využije možnosť zadávať čísla v tejto sústave pomocou funkcie **BIN**, ktorá má tvar:

BIN *dčíslo*

kde *dčíslo* je **dvojkové číslo** (teda číslo v dvojkovej sústave) s maximálnou dĺžkou **16 bitov,** takže hodnota tejto funkcie bude od
0 do 65535. Presvedčte sa, že:

```
BIN 101 = 5
BIN 11111111 = 255
BIN 11111111 11111111 = 65535
```

Z toho by malo byť jasné, že adresa je daná hodnotou 16 bitov.
Keďže vieme adresovať každý byte pamäti a vieme, čo môže obsahovať, popíšeme si teraz, ako môžete s pamäťou pracovať.

Hodnotu jedného bytu v pamäti je možné nastaviť príkazom **POKE**, ktorý má tvar:

POKE *adresa,hodnota*

kde *adresa* a *hodnota* majú známy význam.
Pre načítanie hodnoty bytu z pamäti sa používa funkcia **PEEK** v tvare:

PEEK *adresa*

ktorá udáva hodnotu bytu uloženého na zadanej *adrese.*

Príkazom:

POKE 30000,5 alebo POKE 30000,BIN 101

nastavíme hodnotu bytu s adresou 30000 na hodnotu 5. Či je to skutočne pravda zistíme príkazom:

PRINT PEEK 30000

Príkaz POKE umožňuje meniť takzvané systémové premenné, čo podstatne ovplyvňuje "chovanie" počítača. Podrobnejšie sa o nich dozviete v druhej časti a v prílohe.

1.5.2. Matematické funkcie

Táto kapitola sa zaoberá ďalšími štandardnými funkciami. Je celkom možné, že niektoré funkcie hneď nepoužijete, lebo ste sa s nimi doteraz nestretli. Zahŕňa funkcie **SGN, ABS, INT, SQR, EXP, LN,** trigonometrické funkcie **SIN, COS, TAN** a ich inverzné funkcie **ASN, ACS, ATN.**

Funkcia **SGN** *x* je funkcia signum (znamienková funkcia). Jej argument i výsledok sú čísla. Výsledok je +1, keď je argument kladný, 0 keď je argument 0 a −1, keď je argument záporný.

Funkcia **ABS** *x* je ďalšia funkcia, ktorej argument i výsledok je číslo. Mení argument na nezáporné číslo. Tak napr.:

ABS −3.2 = ABS 3.2 = 3.2
ABS 1000 = 1000

Funkcia **INT** *x* znamená "celá časť". Celá časť z čísla *x* je najväčšie celé číslo, ktoré je menšie alebo rovné ako zadané číslo. Platí:

INT(x) < = x < = INT(x) + 1

Uveďme si konkrétny príklad:

INT 3.9 = 3

Pozor na záporné čísla:

```
INT – 3.9 = -4
```

Funkcia **SQR** *x* vypočítava druhú odmocninu čísla *x*.

```
SQR 4 = 2 ,pretože 2*2=4
SQR 0.25 = 0.5 ,pretože 0.5*0.5=0.25
SQR 2 = 1.414211136
```

Pokiaľ umocňujete číslo (i záporné), výsledok je vždy kladný. To znamená, že záporné čísla nie je možné odmocňovať a pokiaľ tak urobíte, dostanete správu:

```
A Invalid argument
```

Exponenciálna funkcia **EXP** *x* vypočíta hodnotu e^x (e na *x*), kde e je základ prirodzených logaritmov (e = 2.7182818...):

```
EXP x = e ^ x
```

Ľahko zistíte hodnotu čísla e. Zadajte:

```
PRINT EXP 1
```

a dostanete hodnotu čísla **e,** pretože EXP 1 = e ^ 1 = e. Je to len približné vyjadrenie, hodnota **e** sa nedá nikdy presne vyčísliť.

Funkcia **LN** *x* je prirodzený logaritmus *x*. Obecne platí, že logaritmus čísla *x* so základom *z* je výraz, na ktorý musíme umocniť základ *z*, aby sme získali číslo *x* a píše sa $\log_z x$. Preto platí:

$$z \wedge \log_z x = x \,,\; \log_z (z \wedge x) = x$$

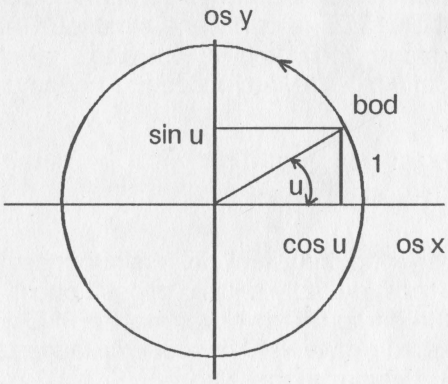
V škole ste možno poznali logaritmy so základom 10, to sú bežné logaritmy. Počítač má funkciu **LN,** ktorá počíta logaritmy so základom **e**. Na výpočet logaritmu s iným základom použite vzťah:

$$\log_z x = \text{LN } x / \text{LN } z.$$

Argument *x* funkcie **LN** musí byť kladné číslo.

Keď je daná kružnica, môžete spočítať jej obvod násobením priemeru číslom **PI**. Číslo PI má v počítači hodnotu 3.1415927. Získate ho v rozšírenom móde stlačením <M>. PI funguje ako každá iná číselná premenná a má dôležité použitie hlavne s funkciami SIN, COS, TAN, ASN, ACS, ATN.
Pre tých, ktorí nevedia o týchto funkciách nič, sa pokúsime teraz vysvetliť ich význam. Nakreslite si dve osi kolmé na seba (jednu vodorovnú, jednu zvislú), ich spoločný bod nech je stredom kružnice, ktorá má polomer 1. Po kružnici sa začne pohybovať bod z pravej strany vodorovnej osi v smere proti pohybu hodinových ručičiek. Súradnice bodu x, y v tejto súradnicovej sústave (poloha bodu vzhľadom na os x a y) sa nazývajú kosínus a sínus uhla u, uhol určuje otočenie bodu od začiatočného bodu (napravo od stredu na osi x). Uhol u budeme zadávať v **radiánoch** (360 stupňov = 2*PI radiánov). Pre prevod medzi stupňami a radiánmi platia vzorce:

rad = stupne * PI/180
stupne = rad * 180/PI



Napr. platí:

SIN 0 = 0
COS 0 = 1
SIN PI/2 = 1
COS PI = −1

Všimnite si, že keď sa bod dostáva vľavo od osi **y**, kosínus je záporný a keď sa bod dostáva pod os **x**, stáva sa i sínus záporným. Ďalšia vlastnosť je, že keď sa bod dostane späť do východzej pozície, majú sínus a kosínus opäť rovnaké hodnoty ako na začiatku pohybu. Preto platí:

```
SIN (a+2*PI) = SIN a
COS (a+2*PI) = COS a
```

Funkcia tangens je definovaná ako podiel sínus/kosínus. Zodpovedajúca funkcia na počítači je **TAN.**
Občas potrebujeme opačný postup, kedy poznáme hodnotu jednej z týchto funkcií a potrebujeme určiť odpovedajúci uhol. Vtedy použijeme funkcie arcsin (na počítači **ASN**), arccos (**ACS**) a arctg (**ATN**).

1.6. Náhodné čísla

Táto kapitola pojednáva o funkcii **RND** a kľúčovom slove **RANDOMIZE**, pomocou ktorých pracujeme s **náhodnými číslami**. Obidve sú na rovnakom klávese <T>. RANDOMIZE je označené skratkou RAND. Funkcia RND nemá žiaden argument. Vždy, keď ju použijete, je jej výsledkom nové náhodné číslo medzi 0 a 1. Niekedy môže dať hodnotu 0, ale nikdy nie 1. Skúste:

```
10 PRINT RND
20 GO TO 10
```

a vidíte, ako sa menia hodnoty funkcie. V skutočnosti RND nie je úplne náhodné číslo, pretože ide vždy o číslo z postupnosti 65536 čísiel (táto postupnosť je pevne daná). Preto hovoríme, že RND je pseudonáhodné číslo. Ako sme už uviedli, dáva RND náhodné čísla medzi 0 a 1, ale môžete ľahko získať i čísla v inom rozsahu. Napr.:

```
5 * RND je číslo medzi 0 a 5,
1.3+0.7*RND je medzi 1.3 a 2.
```

Na získanie celých čísiel použite funkciu **INT** (pamätajte, že INT zaokrúhľuje vždy dole), ako 1 + INT(RND*6), čo simuluje hádzanie kockou. RND*6 je v rozsahu 0 až 6, ale v skutočnosti nikdy nedostanete 6, preto + 1.
Príkaz RANDOMIZE sa používa na to, aby odštartoval RND v určitom mieste postupnosti čísiel, ako ukazuje nasledujúci program:

```
10 RANDOMIZE 1
20 PRINT RND
30 GO TO 20
```

Po každom vykonaní RANDOMIZE 1 začne postupnosť RND číslom 0.0022735596. Môžete samozrejme použiť i iné čísla od 1 po 65535 v príkaze RANDOMIZE, potom je začiatok postupnosti náhodných čísiel v iných miestach. Samotné RANDOMIZE (RANDOMIZE 0 má rovnaký účinok) je rozdielne, pretože skutočne tvorí RND, ako ukazuje ďalší program:

```
10 RANDOMIZE
20 PRINT RND : GO TO 10
```

Postupnosť, ktorú dostanete, nie je náhodná, pretože RANDOMIZE využíva čas od zapnutia počítača. Od tej doby je každé RANDOMIZE vykonané rovnako a preto i RND sú rovnaké. Na získanie náhodných čísiel by bolo lepšie nahradiť príkaz GO TO 10 príkazom GO TO 20.

Poznámka: Väčšina verzií jazyka BASIC používa RND a RANDOMIZE na tvorbu náhodných čísiel, ale nie všetky ich získavajú rovnakým spôsobom.

1.7. Rozhodovanie v programe

Všetky programy, ktoré ste doteraz videli, mali jednoduchú štruktúru. Program sa vykonával postupne od začiatku do konca, prípadne sa vracal na určenú časť.
V praxi je výhodné, ak počítač sám môže rozhodovať na základe nejakej informácie o ďalšom postupe v činnosti. Na to sa používa príkaz **IF - THEN**, ktorý sa nazýva tiež **príkaz podmieneného skoku** (podmieneného preto, lebo ďalšie vykonávanie programu je závislé od splnenia určitej podmienky).

Príkaz má tvar:

IF *podmienka* **THEN** *príkazy*

Ak nie je splnená *podmienka* uvedená medzi slovami IF a THEN, vykonávanie programu pokračuje na ďalšom riadku.
Ak je podmienka splnená, vykonávanie programu pokračuje vykonávaním *príkazov* za slovom THEN.

Použitím NEW vymažte predošlý program, vložte a odštartujte nasledujúci program (je to hra na hádanie celého čísla od 0 do 9, toto číslo si "vymyslí" počítač):

```
10 REM hadanie čísla
20 PRINT "Hadaj cislo od 0 do 9"
30 LET x = INT(10* RND)
40 INPUT "hadaj cislo", y
50 IF x=y THEN GO TO 80
60 PRINT "neuhadol si":PRINT
70 GO TO 40
80 PRINT "uhadol si":PRINT "KONIEC"
```

Program vytvorí na riadku 30 náhodné číslo od 0 do 9 a priradí ho premennej *x*. Po zadaní čísla z klávesnice a jeho uložení do premennej *y* sa dostávame na riadok 50. Ak je splnená podmienka *x=y* (číslo sme uhádli), vykonávanie programu pokračuje za časťou **THEN**, teda vykoná odskok na riadok 80. Ak nie je splnená podmienka, pokračuje sa na riadku 60. Po výpise "neuhadol si" sa z riadku 70 vracia program na riadok 40.
Program môžeme skrátiť zmenou riadku 50 na:

```
50 IF x=y THEN PRINT "uhadol si":PRINT "koniec":STOP
```

a riadok 80 môžeme zmazať.
Podmienka *x=y* v predchádzajúcom príklade sa nazýva **jednoduchá podmienka**. Jednoduché podmienky porovnávajú dve čísla alebo dva reťazce a môžu testovať, či sú si čísla rovné, či je jedno väčšie než druhé a môžu testovať, či dva reťazce si sú rovné alebo sa alfanumericky predchádzajú. Používajú sa značky:

```
=  ,  <  ,  >  ,  <=  ,  >=  ,  <>
```

ktoré sa nazývajú relačné operátory. Ich význam je nasledovný:

= znamená rovná sa; i keď symbol je rovnaký ako v príkaze LET, má tu celkom iný význam.
< (<SYMBOL SHIFT> + <R>) znamená je menší než
1<2, −2 <1, −3 <1 sú pravdivé podmienky,
1<0, 0 < −2 sú nepravdivé podmienky
> (<SYMBOL SHIFT> + <T>)znamená je väčší než, t.j. opačne ako <
<= (<SYMBOL SHIFT> + <Q>) znamená menší alebo rovný
1<= 2, 2 <= 2 sú pravdivé podmienky,
3<= 2, 0 <= −1 sú nepravdivé podmienky
>= (<SYMBOL SHIFT> + <E>) znamená je väčší alebo rovný
<> (<SYMBOL SHIFT> + <W>) znamená nerovnosť a stojí ako protiklad k operátoru =

Upozorňujeme, že symboly <=, >=, <> nemôžeme zložiť z dvoch znakov, ale musíme ich napísať na jeden krát.

V matematike sa píšu výrazy tiež ako:

2<3<4,

čo znamená:

2 < 3 a zároveň 3 <4 ,

ale v jazyku BASIC tento zápis nie je možný.
V programe môžete použiť podmieňovací príkaz viackrát, čo umožňuje vetviť program na viac ako dve možnosti.
To si ukážeme na príklade hádania celého čísla od 0 do 99, pričom nám počítač trochu pomôže:

```
10 REM hadanie cisla od 0 do 99
20 PRINT "Hadaj cislo od 0 do 99"
30 LET x = INT(100* RND)
40 INPUT "hadaj cislo", y
50 IF x=y THEN GO TO 100
```

```
 60 IF y>x THEN GO TO 90
 70 IF y<x THEN GO TO 80
 80 PRINT "tvoje cislo je mensie":PRINT:GO TO 40
 90 PRINT "tvoje cislo je vacsie":PRINT:GO TO 40
100 PRINT "uhadol si":PRINT "koniec"
```

Ak číslo neuhádneme, vypíše počítač informáciu o tom, či je hádané číslo väčšie alebo menšie a program pokračuje na riadku 30. V programe sme použili trikrát podmieňovací príkaz.
Riadok 70 môžeme vynechať. Ak sa dostane riešenie programu až na tento riadok, podmienky na riadkoch 50 a 60 nie sú splnené, takže musí platiť podmienka na riadku 70.
Je zrejmé, že program sa dá opäť skrátiť. Skúste to !

Teraz si ukážeme jeden praktický príklad použitia podmieneného príkazu. Nasledujúci program usporiada 3 čísla podľa veľkosti:

```
10 REM usporiadanie cisiel
20 INPUT "zadaj tri cisla",a,b,c
30 IF a<b THEN GO TO 50
40 LET x=a:LET a=b:LET b=x
50 IF a<c THEN GO TO 70
60 LET x=a:LET a=c:LET c=x
70 IF b THEN GO TO 90
80 LET x=b:LET b=c:LET c=x
90 PRINT "usporiadane cisla ";a,b,c
```

Program pracuje tak, že hodnoty premenných *a*, *b*, *c* sa vymieňajú navzájom tak, aby na konci programu boli premenné *a*, *b*, *c* usporiadané od najmenšieho po najväčšie. Porovnávajú sa vždy dve hodnoty. Zoberme si napr. riadok 30.
Ak $a<b$, tieto čísla sú usporiadané a pokračujeme na riadku 50 ďalšou dvojicou.
Ak neplatí, že $a<b$, hodnoty premenných *a*, *b* navzájom vymeníme na riadku 40. Hodnotu *a* uložíme do pomocnej premennej *x* príkazom LET x=a, lebo nasledujúcim príkazom LET a=b by sme hodnotu premennej *a* stratili. Až potom môžeme príkazom LET b=x priradiť premennej *b* pôvodnú hodnotu premennej *a*.
Analogicky sa robí porovnanie pre ďalšie dve dvojice čísiel.

V podmienke môžeme porovnávať aj reťazce.
Analogicky ako je príklad pre usporiadanie troch čísiel, môžeme nahradením premenných *a*, *b*, *c*, *x* premennými *a$*, *b$*, *c$*, *x$* dostať program pre usporiadanie troch reťazcov.
Vyskúšajte to !

1.7.1. Logické operátory

Pri rozhodovaní môžeme z jednoduchých podmienok vytvoriť podmienku zloženú, ktorá sa nazýva **logický výraz**. Podmienky spájame pomocou **logických operátorov**. V jazyku BASIC používame tieto logické operátory:

Symbol	Príklad	Význam
NOT	NOT A	negácia (opak);ak A je pravdivá podmienka, potom NOT A je nepravdivá (a naopak)
AND	A AND B	logický súčin; A AND B je pravdivá podmienka, ak sú pravdivé podmienky A a B súčasne, inak je A AND B nepravdivá
OR	A OR B	logický súčet; A OR B je pravdivá podmienka, ak je pravdivá aspoň jedna z podmienok A a B, inak je nepravdivá

Význam logických operátorov si objasníme na konkrétnych príkladoch. Zoberme do úvahy celé čísla s nasledujúcimi podmienkami:

A: x > 3 t.j. 4, 5, 6, 7,...
B: x < 7 t.j. 6, 5, 4, 3,...

Potom

NOT A : x < =3 t.j. 3, 2, 1, ...
A AND B : x>3 AND x<7 t.j. celé čísla väčšie ako 3 a súčasne menšie ako 7, čiže 4, 5, 6.

A OR B : x>3 OR x<7 t.j. celé čísla väčšie ako 3 alebo menšie ako 7, čiže všetky celé čísla

Použitie v programe ukazuje táto časť programu:

```
10 INPUT "Zadaj cislo od 10 do 20",a
20 IF a<10 OR a>20 THEN PRINT "Este raz":GO TO 10
30 ...
```

Po chybnom zadaní čísla nás počítač nepustí ďalej, až dokiaľ nezadáme číslo z uvedeného rozsahu.
Logické výrazy môžu byť zapísané s AND, OR, NOT rovnako ako aritmetické výrazy vyjadrujeme pomocou +, −, atď. Pokiaľ je to potrebné, použite zátvorky.

Priorita logických operátorov je nasledovná:

1) NOT
2) AND
3) OR

V prípade rovnocenných operátorov sa postupuje zľava doprava.
NOT je operátor s jedným argumentom, jeho priorita je najvyššia z logických operátorov. Preto jeho argument nepotrebuje zátvorky, aj keď podmienka obsahuje AND alebo OR (alebo obidve).

Teraz si uvedieme niekoľko vlastností logických operátorov, platnosť ktorých si skúste preveriť na konkrétnych príkladoch (ako pomoc použite uvedený príklad s číslami).

NOT a=b znamená to isté ako NOT (a=b) a to isté ako a<>b. <> je negácia =, znamená to, že je pravdivá len vtedy, keď rovnosť je nepravdivá. Taktiež a<>b je to isté ako NOT a=b.

Ďalej NOT a<>b je to isté ako a=b. Presvedčte sa, že >= a <= sú negácie <, > a tak NOT môžete niekedy vynechať zmenením relácií. Ďalej platí:

NOT(*výraz1* AND *výraz2*) je to isté ako NOT *výraz1* OR NOT *výraz2*

NOT(*výraz1* OR *výraz2*) je to isté ako NOT *výraz1* AND NOT *výraz2*

Pri použití NOT v programe nie je nutné používať medzi vzťahmi zátvorky. Ďalšia časť výkladu je trochu komplikovaná a môžete ju zatiaľ preskočiť (pri hlbšom štúdiu si ju však preštudujte).
Skúste zadať:

PRINT 1 = 2,1 < >2

Asi očakávate syntaktickú chybu, ale všetko je v poriadku.
Počítač totiž prevádza logické hodnoty na čísla, s ktorými vie pracovať podľa nasledujúcich pravidiel:

a) relácie =, <, >, < =, > =, < > dávajú číselné výsledky:
1 pre pravdivú podmienku
0 pre nepravdivú podmienku;
preto vyššie uvedený príkaz PRINT tlačí hodnotu 0 pre podmienku 1 = 2, pretože ide o nepravdivú podmienku, podmienka 1 < >2 dáva hodnotu 1, lebo je pravdivá.

b) v príkaze IF *podmienka* THEN
môže byť podmienka akýkoľvek aritmetický výraz. Pokiaľ je jeho hodnota 0, potom sa chová ako keby tam bola nepravdivá podmienka. Všetky ostatné hodnoty (vrátane 1) sa berú ako pravdivá podmienka. Preto
IF *podmienka*
v skutočnosti znamená
IF *podmienka* < >0 THEN ...

c) AND, OR, NOT sú operácie tvoriace číselné hodnoty

x AND y má hodnotu x, pokiaľ y je pravdivé (nie 0)
0, pokiaľ je y nepravdivé

x OR y má hodnotu 1, keď je y pravdivé (nie 0)
x, keď je y nepravdivé

NOT x má hodnotu 0, keď x je pravdivé (nie 0)
1, keď x je nepravdivé

Nakoľko sme si vysvetlili doteraz všetky potrebné pojmy, uvedieme si prioritu vyhodnocovania výrazov:

1) vyhodnotenie zátvoriek
2) vyhodnotenie funkcie
3) umocňovanie
4) unárne mínus (záporné znamienko)
5) násobenie a delenie
6) sčítanie a odčítanie
7) relačné operátory
8) negácia
9) logický súčin
10) logický súčet

V prípade rovnakej priority sa postupuje zľava doprava. Ak chcete zmeniť poradie vyhodnocovania, použite zátvorky (majú najvyššiu prioritu). Nakoniec si ukážme, ako sa dajú použiť zložité logické výrazy v programe:

```
10 INPUT a
20 INPUT b
30 PRINT (a AND a > = b) + (b AND a < b)
40 GO TO 10
```

Výsledkom je výpis väčšieho z dvoch čísiel *a*, *b*.

Analogicky sa používajú reťazcové hodnoty v podmienkových výrazoch, ale len pri použití AND.

```
x$ AND y má hodnotu x$, keď y nie je 0
                     "",  keď y je 0
```

Prezrite si nasledujúci program, ktorý vypíše dva vstupujúce reťazce v abecednom poradí.

```
10 INPUT "vložte dva reťazce",a$, b$
20 IF a$ > b$ THEN LET c$ = a$: LET a$ = b$: LET b$ = c$
30 PRINT a$;" ";("<" AND a$ < b$) + ("=" AND a$ = b$);" ";b$
40 GO TO 10
```

1.8. Použitie cyklu v programe

Pri programovaní sa veľmi často opakujú tie isté výpočty. Pomocou príkazu GO TO môžeme vracať riešenie programu na žiadané opakované výpočty.
Máme za úlohu sčítať celé čísla od 1 do 30.

```
10 LET s=0:LET c=1
20 LET s=s+c
30 LET c=c+1
40 IF c<=30 THEN GO TO 20
50 PRINT "sucet = ";s
```

Na riadku 10 nastavíme začiatočné hodnoty *s* a *c*.
Premenná *c* predstavuje číslo, ktoré máme do celkového súčtu pričítať.
Premenná *s* predstavuje súčet čísiel. Okamžitá hodnota s predstavuje medzisúčet čísiel od 1 do *c*.
Na riadku 20 pričítame do medzisúčtu číslo *c*, kde *c* je nasledujúce číslo z požadovaných čísiel.
Na riadku 30 vytvoríme nasledujúce číslo tak, že predchádzajúce číslo zväčšíme o 1.
Na riadku 40 testujeme, či je takto vytvorené číslo < = 30. Ak je, program pokračuje na riadku 20.
Ak je číslo väčšie ako 30, v medzisúčte sú už všetky čísla od 1 do 30 sčítané, podmienka v príkaze IF - THEN nie je splnená a počítač vypíše celkový súčet (riadok 50).
Opakované využívanie časti programu nazývame **programový cyklus**.

Pri použití cyklu platia nasledujúce zásady:

a) pred vstupom do cyklu nastavíme začiatočné hodnoty premenných, ktoré v cykle využívame (v našom prípade *s*, *c*)
b) v cykle sa hodnota niektorých premenných mení o určenú hodnotu (*c* sa zvyšuje o 1)
c) v cykle sa testuje, či je splnená podmienka ukončenia cyklu; ak nie je splnená, skáče sa na začiatok cyklu

Programový cyklus, ktorý má tieto vlastnosti, môžeme jednoduchšie vytvoriť pomocou príkazu **FOR-TO-STEP-NEXT**.

Začiatok cyklu má tvar:

FOR *premenná* = *od* **TO** *po* **STEP** *krok*

Koniec cyklu sa označuje

NEXT *premenná*

Premenná je takzvaná **riadiaca premenná (parameter cyklu)**, ktorej meno môže byť len jedno písmeno.
Výrazy *od*, *po*, *krok* sú ľubovoľné aritmetické výrazy.
Časť programu medzi začiatkom cyklu a koncom cyklu sa volá **telo cyklu**.

Výraz *od* určuje začiatočnú hodnotu riadiacej premennej.
Výraz *po* určuje hornú hranicu riadiacej premennej.
Výraz *krok* určuje veľkosť kroku, o ktorý sa hodnota riadiacej premennej zmení pri každom prechode cyklom.

Telo cyklu sa vykoná opakovane takto:

Riadiacej premennej sa priradí hodnota výrazu *od* a vykoná sa telo cyklu. Na konci cyklu sa k okamžitej hodnote riadiacej premennej pričíta hodnota výrazu *krok*. Testuje sa, či je hodnota riadiacej premennej väčšia (pri kladnom kroku), alebo menšia (pri zápornom kroku), ako hodnota výrazu *po* (koncová hodnota). Ak nie je koncová hodnota prekročená, telo cyklu sa vykoná s novou hodnotou riadiacej premennej. V opačnom prípade je cyklus ukončený a vykonávanie programu pokračuje ďalším príkazom za príkazom NEXT.
Ukážme si, ako bude vyzerať program pre súčet čísiel od 1 do 30 s využitím tohto príkazu:

```
10 LET s=0
20 FOR c=1 TO 30 STEP 1
30 LET s=s+c
40 NEXT c
50 PRINT "sucet = ";c
```

Na riadku 10 je nastavený medzisúčet *s* na hodnotu 0.

Na riadku 20 sú nastavené vlastnosti cyklu; riadiaca premenná bude nadobúdať hodnoty od 1 do 30 s krokom 1, riadok 30 tvorí vlastné telo cyklu. Riadok 40 je ukončenie cyklu, za NEXT je uvedený názov riadiacej premennej cyklu.
Ak má krok hodnotu +1, časť príkazu STEP výraz 3 sa nemusí v príkaze uvádzať. Riadok 20 sme mohli preto napísať takto:

```
20 FOR c = 1 TO 30
```

Ak chcete zmeniť program na súčet iných čísiel, stačí zmeniť jediný riadok programu, riadok 20.
Ak chceme sčítať čísla od 10 do 100 s krokom 0.5, zmeníme riadok na tvar:

```
20 FOR c = 10 TO 100 STEP 0.5
```

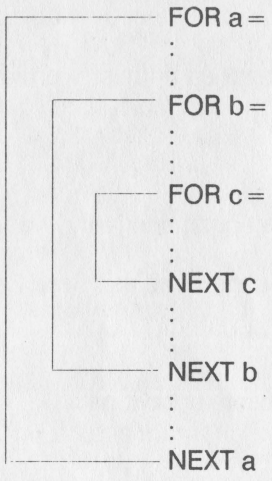
Hodnota kroku môže byť i záporná. Riadok 20 môžete upraviť na :

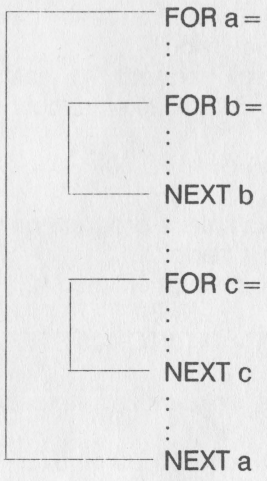
```
20 FOR c = 100 TO 10 STEP -0.5
```

Tu sme však museli zmeniť aj začiatočnú a koncovú hodnotu riadiacej premennej.

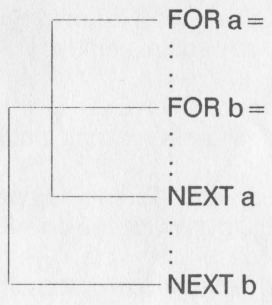
Ak je pri kladnom kroku začiatočná hodnota riadiacej premennej väčšia ako koncová hranica a pri zápornom kroku menšia ako koncová hranica, telo cyklu sa nevykoná ani raz a program pokračuje ďalším príkazom za príkazom NEXT.
Hodnota riadiacej premennej môže byť v tele cyklu zmenená. Na konci cyklu sa testuje vždy jej okamžitá hodnota. Hranice cyklu sa nedajú meniť vo vnútri cyklu.
Do cyklu môžeme vstúpiť iba cez počiatok cyklu (cez FOR-NEXT).
Skok zvonku dovnútra cyklu nie je dovolený. Je ale možné vyskočiť z cyklu pred jeho ukončením.
V jednom programe môžeme použiť niekoľko cyklov. Ak sú vzájomne nezávislé, nemali by sme urobiť chybu. Ak sa použijú cykly vložené do seba, musí sa vždy ako prvý ukončiť najvnútornejší cyklus.
Na niekoľkých príkladoch si ukážeme rôzne umiestnenie cyklov v programe.

Správne vložené cykly:

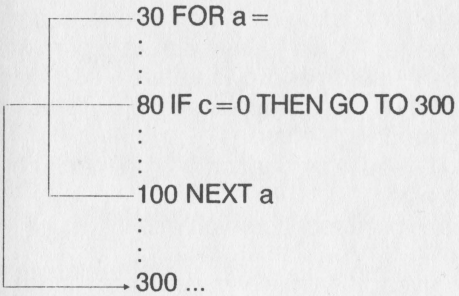




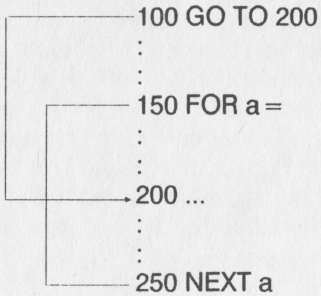
Nesprávne vložené cykly:



Povolený skok z cyklu:



Nesprávny skok do cyklu:



V nasledujúcom príklade uvidíte, aký užitočný je príkaz cyklu.
Úlohou je vytvoriť program pre načítanie mien a ich usporiadanie podľa abecedy. Meno je dlhé maximálne 10 znakov.

```
 10 PRINT "Usporiadanie mien podla abecedy"
 20 PRINT:INPUT "Zadaj pocet mien",k
 30 DIM a$(k,10)
 40 FOR i=1 TO k:PRINT "Zadaj meno cislo ";i
 50 INPUT a$(i)
 60 NEXT i
 70 FOR i=1 to k-1
 80 FOR j=i+1 TO k
 90 IF a$(i)a$(j) THEN GO TO 110
100 LET x$=a$(i):LET a$(i)=a$(j):LET a$(j)=x$
110 NEXT j
120 NEXT i
130 PRINT :PRINT"Usporiadane mena":PRINT
140 FOR i=1 to k:PRINTa$(i):NEXT i
```

Na riadku 10,20 je úvodný text.
Na riadku 30 sa zadá počet mien a rezervuje sa miesto v pamäti pre tieto premenné.
Na riadkoch 40 až 60 sa načítajú mená do reťazcových premenných a$(1)...a$(k).

Vlastné usporiadanie sa vykoná na riadkoch 70 až 120. Princíp usporiadania je ten, že sa zoberie prvé meno ($i=1$) a porovná sa so všetkými nasledujúcimi (to sú tie, ktorých indexy sú väčšie ako i, teda pre $j=i+1$ až k na riadku 80). Vlastné porovnanie je na riadku 90. Ak je splnená podmienka, nechá sa reťazec na svojom mieste. Ak nie je, vymení sa s tým, ktorý je "menší" (riadok 100). Po ukončení vnútorného cyklu bude v *a$(1)* uložené "najmenšie" meno. Hodnota i sa zvýši a postup sa opakuje. Tak sa nájde druhé "najmenšie" meno a uloží sa do *a$(2)*, atď.
Na riadku 70 je horná hranica *k*-1, lebo je nájdených *k*-1 "najmenších" mien a to ostávajúce nemôže byť "menšie".
Na riadkoch 130, 140 sa vykoná výpis usporiadaných mien.

Na tomto príklade si môžete uvedomiť, aký význam majú indexované premenné. Keby sme ich nemali, problém usporiadania by sa riešil veľmi ťažko.
Ďalšiu výhodu vidíte na riadku 140, kde sa jedným krátkym príkazovým riadkom vypíše ľubovoľný počet premenných.
Príkaz cyklu spolu s možnosťou použitia indexovaných premenných tvoria veľmi silný programovací prostriedok.
Ak v uvedenom programe nahradíte reťazcové premenné číselnými premennými, program usporiada a vypíše zadaný počet čísiel.

1.9. Zadávanie dát počítaču

Naučili ste sa, ako priradíte premenným hodnotu príkazom LET a INPUT. Tento spôsob nie je niekedy výhodný, hlavne vtedy, ak je zadávaných údajov viac. V takom prípade sa používa príkaz **READ,** s ktorým úzko súvisia príkazy **DATA** a **RESTORE.**

Príkaz **READ** má tvar:

READ *zoznam premenných*

kde *zoznam premenných* je zoznam číselných a reťazcových premenných, oddelených od seba čiarkou.

Príkaz **DATA** má tvar:

DATA *zoznam výrazov*

kde *zoznam výrazov* je zoznam aritmetických výrazov alebo reťazcov, ktoré sú oddelené od seba čiarkou.
V programe môžete použiť viac príkazov READ a DATA, pričom príkaz DATA môže byť na ľubovoľnom mieste programu. Priradenie hodnôt premenným sa vykoná takto:
Podobne, ako má počítač ukazovateľ pre výpis na obrazovku, má pre príkaz DATA ukazovateľ dát, ktorý ukazuje na výraz v príkaze DATA, ktorého hodnota sa bude priraďovať ako nasledujúca.
Na začiatku programu sa nastaví ukazovateľ na prvý výraz v prvom príkaze DATA. Príkazom READ sa priradí premennej hodnota výrazu, na ktorý je nastavený ukazovateľ. Po priradení sa ukazovateľ nastaví na ďalší výraz v príkaze DATA. Ak sa položky zo zoznamu DATA vyčerpali, ukazovateľ sa nastaví na prvú položku v nasledujúcom príkaze DATA.
V príkaze READ môžu byť v zozname vedľa seba číselné aj reťazcové premenné. Číselnej premennej sa môže priradiť len číselná hodnota, t.j. odpovedajúci výraz v príkaze DATA musí byť aritmetický výraz. Reťazcovej premennej sa môže priradiť len reťazec:

```
10 DATA 2*3,.5,"den"
20 READ a,b,c$
30 PRINT a,b,c$
```

Po vykonaní programu budú mať premenné tieto hodnoty:

a = 6, b = 0.5, c$ = "den"

Ak je položiek v príkazoch DATA menej ako premenných v príkazoch READ, počítač ohlási chybu. Zmeňte riadok 10 na:

```
10 DATA 1,2
```

Po odštartovaní programu vypíše počítač hlásenie:

```
E Out of DATA
```

Niekedy je treba použiť rovnaké hodnoty pre rôzne premenné. Aby sa nemusel zoznam dát vypisovať viackrát za sebou v programe, využijeme príkaz **RESTORE**, ktorý má tvar:

RESTORE *číslo riadku*

Tento príkaz nastaví ukazovateľ dát na prvú položku v príkaze DATA na *riadku* so zadaným *číslom.* Ak nie je číslo riadku zadané, ukazovateľ dát sa nastaví na prvú položku v prvom príkaze DATA v programe. Vyskúšajte to na príklade:

```
10 DATA 1,2,3
20 READ a,b,c,pocet
30 RESTORE
40 READ a1,a2
50 DATA 10,20
60 RESTORE 50
70 READ i,j
```

Premenným *a, b, c, pocet, a1, a2, i, j* sa príkazom READ priradia hodnoty 1, 2, 3, 10, 1, 2, 10, 20.
Často sa používa príkaz READ v cykle:

```
10 DIM a(8)
20 DATA 3,9,5,6,4,1,7,8,78,55
30 FOR i=1 TO 8
40 READ a(i)
50 NEXT i
```

Všimli ste si, že v príkaze DATA je viac ako 8 položiek, hoci indexovaných premenných je len 8. To však nevadí, niektoré položky v príkaze DATA nemusia byť nikdy prečítané.
Ak použijete v programe príkaz CLEAR na zrušenie premenných, počítajte s tým, že sa vykoná zároveň RESTORE.

1.10. Doplnenie príkazu PRINT a INPUT

Príkaz PRINT sme použili v doterajších príkladoch dosť často, takže iste už viete ako pracuje a ako sa používa. Výrazy, ktorých hodnoty sú tlačené, sa nazývajú položky príkazu PRINT a sú oddelené čiarkami alebo bodkočiarkami, ktoré sa nazývajú oddeľovače príkazu PRINT. V príkaze PRINT môžeme použiť špeciálne položky, ktoré nám uľahčujú prácu pri výpisoch.

Často chceme výpis na obrazovku umiestniť na presne stanovené miesto. Na to používame položku **AT**, ktorá ma tvar:

AT *riadok,stĺpec*

Týmto sa nastaví ukazovateľ pre výpis na špecifikované miesto obrazovky (*riadok*, *stĺpec*). Riadky sa číslujú od 0 (horný) po 21, stĺpce od 0 (vľavo) po 31. Príkazom

```
PRINT AT 12,16; "*"
```

vypíšeme znak "*" do stredu obrazovky. Pretože položka AT je chápaná ako položka v príkaze PRINT, musí byť od ostatných položiek oddelená povoleným oddeľovačom (v našom prípade bodkočiarka). Ukazovateľ pre výpis sa nastaví analogicky, ako sme uvádzali už na začiatku príručky.

Funkcia **SCREEN$** je opačná k činnosti položky AT. Zisťuje, aký znak je v určitej pozícii na obrazovke zobrazený. Používa číslo riadku a stĺpca rovnako ako položka AT, ale sú uzavreté v zátvorkách:

SCREEN$ (*riadok,stĺpec*)

Príkaz:

```
PRINT SCREEN$ (12,16)
```

znovu vypíše hviezdičku, ktorá bola tlačená v predchádzajúcom príklade. Znaky sa berú normálne ako znaky, medzery sa vracajú ako medzery. Ak sú na zadanom mieste obrazovky užívateľské znaky, grafické symboly a grafy, funkcia má hodnotu nulového (prázdneho) reťazca.

Ďalšou položkou je **TAB** v tvare:

TAB *stĺpec*

Táto položka tlačí medzery na presun do špecifikovaného *stĺpca* riadku, riadok sa nemení. Pamätajte, že počítač redukuje číslo stĺpca modulo 32 (delí 32 a berie zvyšok), tak napr. TAB 33 znamená to isté ako TAB 1.

Skúste odštartovať program:

```
10 FOR n=0 TO 20
20 PRINT TAB 8*n;n;
30 NEXT n
```

Program ukazuje, že čísla pre TAB sú redukované modulo 32 (do úvahy sa berú zvyšky po delení číslom 32). Aby bol príklad elegantnejší, zmeňte 8 v riadku 20 na 6.

Niektoré dôležité poznámky:

1) Nastavenie pomocou AT je najlepšie ukončovať bodkočiarkou. Ak nie je na konci bodkočiarka, pozícia sa znovu zmení a potom je ju treba opäť nastaviť.
2) Nemôžete tlačiť na spodné dva riadky (22 a 23), pretože sú rezervované pre príkazy, vstupné dáta, správy atď. Odkazy na spodný riadok obvykle znamenajú riadok 21.
3) AT môžete použiť na to, aby PRINT bol vykonaný tam, kde je už niečo vypísané, starý výpis bude prepísaný.

Niekedy je vhodné, aby sa počítač nezastavil po výpise obrazovky a nečakal na stlačenie klávesu. Dá sa to urobiť pomocou príkazu

```
POKE 23692,255
```

Po tomto príkaze sa bude scrolling vykonávať 255x bez zastavenia s otázkou:

```
scroll ?
```

Skúste tento príklad:

```
10 FOR n=0 TO 10000
20 PRINT n: POKE 23692,255
30 NEXT n
```

a pozorujte, čo sa bude diať na obrazovke.
Zoznam v príkaze PRINT môže obsahovať rôzne kombinácie čiarok, bodkočiarok, AT, TAB. Ich spracovanie prebieha zľava doprava a pre každú položku platia pravidlá, ktoré sme doteraz uviedli.

Ďalším príkazom, ktorý sa týka vstupu a výstupu dát, je príkaz INPUT, ktorý dokáže viac, než sme doteraz uviedli. Zatiaľ sme tento príkaz používali asi nasledovne:

```
INPUT "Kolko mas rokov "; vek
```

v ktorom tlačí počítač uvedenú otázku v dolnej časti obrazovky a Vy potom zadáte svoj vek.
V skutočnosti je príkaz INPUT zostavený z položiek rovnako ako PRINT a tak text "Kolko mas rokov" a premenná vek sú obe položky príkazu INPUT. Všeobecne sú položky v príkazoch INPUT a PRINT rovnaké, ale sú tu veľmi dôležité rozdiely.
Obvyklý vstup INPUT je premenná, ktorej hodnota musí byť vložená, v našom príklade *vek*. Ak položka v INPUT začína znakom, musí to byť premenná, ktorej hodnotu potom vložíme. Ak sa očakáva reťazec, kurzor vo vstupnom riadku je zobrazený v úvodzovkách.
Ak chcete tlačiť hodnoty premenných ako časť záhlavia, urobíte to tak, že okolo premennej dáte zátvorky. Dokonca môžete do zátvoriek uzavrieť ľubovoľný výraz, ktorého hodnotu chcete tlačiť v príkaze INPUT.

Tu je názorný príklad ilustrujúci tento postup:

```
LET mojvek = INT(RND*100):INPUT("Ja mam ";moj vek;" rokov,");
             " ty mas kolko rokov ?",tvojvek
```

Premenná *mojvek* je v zátvorke preto, lebo sa jej hodnota tlačí, *tvojvek* nie je v zátvorke, preto musíte vložiť jej hodnotu. Všetko, čo je v príkaze INPUT, sa píše v dolnej časti obrazovky a chová sa nezávisle na hornej časti obrazovky.
Vo všetkých prípadoch sa tieto riadky vzťahujú k hornému riadku spodnej časti, i keď vkladáte veľa údajov v príkaze INPUT.

Aby ste poznali, ako pracuje AT v príkaze INPUT, skúste zadať:

```
10 INPUT "To je riadok 1." ,a$; AT 0,0;"To je riadok
 0.",a$;AT 2,0; "To je riadok 2.",a$; AT 1,0;"To je stale
 riadok 1.",a$
```

Teraz skúste:

```
10 FOR n=0 TO 19: PRINT AT n,0;n:NEXT n
20 INPUT AT 0,0;a$; AT 1,0;a$;AT 2,0;a$; AT 3,0;a$;
   AT 4,0;a$; AT 5,0;a$
```

Dolná časť obrazovky sa posúva smerom nahor, horná časť je neporušená. Keď má byť písané na ten istý riadok ako v PRINT, začne sa horná časť obrazovky rolovať.

Často sa stáva, že chcete prerušiť program počas zadávania dát v príkaze INPUT. Stlačenie klávesu <BREAK> tu nepomôže. Ak sa očakáva vstup číselnej premennej, zadáte príkaz <STOP> (<SYMBOL SHIFT + <A>) a stlačíte <ENTER>. Ak sa očakáva reťazec, presuniete kurzor pred úvodzovky, zadáte <STOP> a stlačíte <ENTER>. Ešte doplňme, že v prípade zadávania reťazca sa môže zadať namiesto reťazcovej konštanty uzavretej v úvodzovkách i zložitejší reťazcový výraz (úvodzovky môžeme prípadne zmazať).

Ďalšou možnosťou v príkaze INPUT, ktorú ste doteraz nepoznali, je typ **LINE** vstup, kde ide o iný spôsob vkladania hodnôt reťazcovým premenným. Pokiaľ napíšete LINE pred meno reťazcovej premennej (pri vstupe) napr.:

INPUT LINE *a$*

potom počítač nevypíše reťazcové úvodzovky, ako je to pri vstupe reťazca obvyklé, ale očakáva vstup reťazca, ako keby tam úvodzovky boli. Tak keď napíšete "ano" ako vstup dát, dostane hodnotu ano. Pretože reťazcové úvodzovky nie sú vypísané, nemôžete ich zrušiť a vkladať tak iný druh reťazcového vyjadrenia pre vstup dát. Pamätajte, že nemôžete použiť LINE vstup pre číselné premenné.

Riadiace znaky CHR$ 22 a CHR$ 23 majú účinok ako AT a TAB. Pretože ide o riadiace znaky, musia byť nasledované dvoma ďalšími znakmi, s ktorými sa počíta (presnejšie s ich kódmi) ako s číslami na špecifikáciu stĺpca a riadku (pre AT) a pozície (pre TAB). Takmer vždy je ľahšie použiť AT a TAB v ich obvyklej forme, než použiť riadiace znaky, ale i tie sú niekedy užitočné. V riadiacom znaku po CHR$ 22 (pre AT) prvý znak špecifikuje číslo riadku a druhý číslo stĺpca. Preto:

```
PRINT CHR$ 22 + CHR$ 1 + CHR$ c;
```

má presne rovnaký účinok ako:

```
PRINT AT 1,c;
```

Riadiaci znak pre TAB je CHR$ 23 a dva znaky za tým určujú číslo od 0 do 65535, špecifikujúce číslo v položke TAB.

```
PRINT CHR$ 23 + CHR$ a + CHR$ b;
```

má rovnaký účinok ako:

```
PRINT TAB (a + 256*b);
```

Vyskúšajte si nasledujúci program pre deti, ktorý testuje, ako vedia násobiť:

```
 10 LET m$ = ""
 20 LET a = INT (RND*12) + 1:LET b = INT (RND*12) + 1
 30 INPUT (m$)"kolko je ";(a);"*";(b);"?";c
100 IF c = a*b THEN LET m$ = "Spravne.":GO TO 20
110 LET m$ = "Zle. Zadaj znova.":GO TO 30
```

Pokiaľ sú deti vnímavé, mohli by zistiť, že vlastne nemusia vôbec počítať. Napr. keď sa ich počítač spýta, koľko je 2*3, môžu namiesto správnej odpovede napísať 2*3. Cesta, ako tomu zabrániť, je čítať vstupný reťazec namiesto čísla. Nahraďte *c* v riadku 30 premennou *c$*, v riadku 100 výrazom VAL c$ a vložte riadok:

```
40 IF c$ < > STR$ VAL c$ THEN LET m$ = "píšte správne ako
   číslo.":GO TO 30.
```

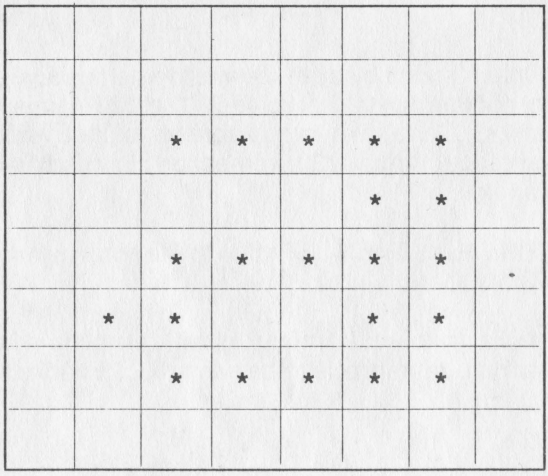
To ich bude hnevať. Za niekoľko dní môžu objaviť, že možno vkladať reťazec STR$ (2*3). Aby sa tomu zabránilo, je treba nahradiť *c$* v riadku 30 príkazom LINE c$.

1.11. Práca s farbami

Počítač je schopný zobraziť na farebnom televízore (alebo monitore) 8 farieb. Pokiaľ je Váš televízor čiernobiely, uvidíte len odtiene šedej farby. Jednotlivé farby sú označené číslom:

0 - čierna
1 - modrá
2 - červená
3 - purpurová (fialová)
4 - zelená
5 - svetlo modrá
6 - žltá
7 - biela

Aby ste používali farby správne, musíte pochopiť, ako je obraz zostavený. Obraz je rozdelený na 768 pozícií (24 riadkov a 32 stĺpcov), kde môžu byť znaky vytlačené.
Každý znak je tlačený ako 8x8 bodov obrazovky, ako ukazuje obrázok malého a :



Každý bod z 8x8 bodov štvorca pre výpis jedného znaku môže mať jednu z dvoch farieb: farbu znaku (farba tmavých bodov v našom štvorci) a farba pozadia (je použitá pre biele body).

Na začiatku práce s počítačom majú všetky zobrazené body čiernu farbu a pozadie (podklad) farbu bielu, a tak sa znaky javia ako čiernobiele. Každý znak má tiež jas (normálny alebo zvýšený) a môže blikať (blikanie je vlastne výmena farby pozadia a zobrazených bodov). Všetko je kódované v číslach a počítač potrebuje pri výpise jedného znaku o znakovej pozícii vedieť nasledujúce informácie:

1. 8x8 bodov, kde 0 znamená body podkladu a 1 znamená zobrazené body, definujúce tvar znaku
2. farby zobrazených bodov a podkladu, každá z nich má kód 0 až 7.
3. jas - 0 normálny, 1 zvýšený
4. blikanie - 0 bez blikania, 1 bliká

Z toho vyplýva, že jedna znaková pozícia (8x8 bodov) môže mať najviac dve farby, všetky body majú normálny alebo zvýšený jas a všetky body blikajú alebo neblikajú.
Hodnoty dvoch farieb, jasu a blikania sa nazývajú **atribúty**. Atribúty sa dajú programovo meniť a príkazy na ich zmenu si teraz popíšeme.

Programovo môžeme ovládať nielen farbu vypisovaných znakov, ale tiež okraj obrazovky, t.j. časť, do ktorej sa nedá nič vypísať príkazom PRINT. Farba okraja obrazovky sa mení príkazom **BORDER** v tvare:

BORDER *farba*

kde *farba* je celé číslo od 0 do 7. Ihneď po zadaní tohto príkazu sa okraj obrazovky zafarbí zadanou farbou.

Farbu podkladu pracovnej časti obrazovky meníme príkazom **PAPER** v tvare:

PAPER *farba*

kde *farba* je celé číslo od 0 do 9. V prípade, že je v rozmedzí od 0 do 7, určuje farbu podkladu. Ak má hodnotu 8, farba podkladu sa nemení, t.j.

ostáva nastavená tak, ako to bolo pred týmto príkazom. Ak má hodnotu 9, farba podkladu bude kontrastná oproti farbe výpisu znakov, t.j. bude:

-biela ako kontrast pre tmavú (čiernu, modrú, červenú, purpurovú)
-čierna ako kontrast pre svetlú (zelená, svetlo modrá,žltá,biela)

Farbu výpisu znakov zadáme príkazom **INK** v tvare:

INK *farba*

kde pre *farbu* platí to isté, ako v príkaze PAPER.

Pre určenie jasu použijeme príkaz **BRIGHT** v tvare:

BRIGHT *číslo*

kde *číslo* môže mať tieto hodnoty:

0 - jas je normálny
1 - jas je zvýšený
8 - jas sa nemení, ostáva nastavený podľa predchádzajúceho príkazu BRIGHT

Blikanie zadáme príkazom **FLASH** v tvare:

FLASH *číslo*

kde *číslo* môže mať tieto hodnoty

0 - bez blikania
1 - blikanie
8 - bez zmeny podľa predošlého nastavenia FLASH

Použitie uvedených príkazov môžete odskúšať na nasledujúcom príklade:

```
10 FOR m=0 TO 1:BRIGHT m
20 FOR n=1 TO 10
30 FOR c=0 TO 7
40 PAPER c: PRINT "       ";:REM 4 farebne medzery
```

```
 50 NEXT c: NEXT n: NEXT m
 60 FOR m=0 TO 1:BRIGHT m:PAPER 7
 70 FOR c=0 TO 3
 80 INK c:PRINT c;" ";
 90 NEXT c: PAPER 0
100 FOR c=4 TO 7
110 INK c: PRINT c;" ";
120 NEXT c: NEXT m
130 PAPER 8: INK 7: BRIGHT 0
```

Program zobrazuje 8 farieb (vrátane čiernej a bielej) a dve úrovne jasu, ako sa zobrazujú na farebnom televízore.
Tlačí sa vo farbách podľa zadaných príkazov (medzera je znak, kde farba INK a PAPER je zhodná).
Kontrastnú farbu vyskúšate nasledovným príkazovým riadkom:

```
INK 9: FOR c=0 TO 7: PAPER c: PRINT c: NEXT c
```

Ďalšie dva príkazy **INVERSE** a **OVER** neriadia atribúty, ale spôsob akým sú znaky vypisované na obrazovke. Tieto príkazy majú tvar:

INVERSE *číslo*
OVER *číslo*

kde *číslo* môže mať len hodnotu 0 a 1.

Ak zadáte INVERSE 1, potom výpis znaku bude **invertovaný** (opačný) vzhľadom k normálnemu výpisu (farby určené príkazmi BORDER a INK sa navzájom vymenia). Pokiaľ máte čierny INK a biely BORDER (ako po prvom zapnutí), tak po inverzii budú písmená biele na čiernom podklade.
Zadaním INVERSE 0 sa prejde do pôvodného stavu.

Príkaz OVER 1 zapína zvláštny druh výpisu - **prepisovanie**. Normálne, keď je niečo napísané do znakovej pozície, je celkom jedno, čo tam bolo skôr, ale po OVER 1 nový znak nezmaže starý znak, ale znaky sa prelínajú cez seba. Môže to byť zvlášť užitočné pre písanie kompozičných znakov, podobne ako znakov, na ktoré je kladený dôraz (napr. v nemčine prehlasované o):

```
10 OVER 1
20 FOR n=1 TO 32
30 PRINT "o";CHR$ 8;"";
40 NEXT n
```

Veľkou výhodou príkazov pre farbu je to,že tieto príkazy môžu byť položkami v príkaze PRINT a ich účinok platí v tom prípade iba v tom príkaze PRINT, v ktorom sú uvedené.
Napr. po odoslaní:

```
PRINT PAPER 6;"x";:PRINT "y"
```

bude iba x na žltom pozadí.

Tieto príkazy nemajú účinok na farby v dolnej časti obrazovky, kde sú vkladané príkazy INPUT. Dolná časť obrazovky používa farbu pre podklad podľa príkazu BORDER a kód 9 (ako kontrast) pre farbu INK, bez blikania s normálnym jasom.

Keď vkladáte údaje pomocou INPUT, farbu môžete zmeniť použitím INK a PAPER ako v príkaze PRINT. Účinok trvá do konca príkazu alebo do ďalšieho príkazu INPUT. Skúste:

```
INPUT FLASH 1; INK 1; "Zadaj svoje meno";n$
```

Ďalší spôsob zmeny farieb je použitím riadiacich znakov, kde:

```
CHR$ 16 zodpovedá INK
CHR$ 17 zodpovedá PAPER
CHR$ 18 zodpovedá FLASH
CHR$ 19 zodpovedá BRIGHT
CHR$ 20 zodpovedá INVERSE
CHR$ 21 zodpovedá OVER
```

Za nimi nasleduje jeden znak, ktorého kód udáva farbu, napr.:

```
PRINT CHR$ 16+CHR$ 9; ...          má rovnaký účinok ako
PRINT INK 9; ...
```

Tieto riadiace znaky nemusíte používať, používajte radšej príkazy. Niekedy je však užitočné pracovať s nimi pri vkladaní programu. Výpis programu bude mať v rôznych častiach rôznu farbu, aby sa časti odlíšili od seba, alebo pekne vyzerali.

Aby ste ich dostali do programu, musíte ich zadať priamo z klávesnice použitím rozšíreného módu. Zmenu farby zadáme vždy za číslom riadku. Keď ste v E-móde a stlačíte kláves <6> (pre žltú farbu), potom sa vložia dva znaky; najprv CHR$ 17 ako PAPER a CHR$ 6 ako kód pre žltú farbu. Keď stlačíte <CAPS SHIFT> + <6>, dostávate kód CHR$ 16 ako INK a CHR$ 6 pre farbu. Pretože sa vložili 2 znaky, vymazanie musíte vykonať tak, že dvakrát stlačíte <DELETE>. Po prvom stlačení sa objaví znak "?". Nebojte sa a stlačte <DELETE> znovu.

Taktiež ostatné klávesy horného radu klávesnice majú v rozšírenom móde E nejaký význam:

<8> dáva CHR$ 19 a CHR$ 0 pre normálny jas
<9> dáva CHR$ 19 a CHR$ 1 pre zvýšený jas
<CAPS SHIFT> + <8> dáva CHR$ 18 a CHR$ 0 pre neblikanie
<CAPS SHIFT> + <9> dáva CHR$ 18 a CHR$ 1 pre blikanie

V obyčajnom L móde platí:

<CAPS SHIFT> + <3> dáva CHR$ 20 a CHR$ 0 pre normálne znaky
<CAPS SHIFT> + <4> dáva CHR$ 20 a CHR$ 1 pre inverzné znaky

V nasledujúcej tabuľke je podrobný popis významu horného radu klávesnice v rôznych módoch:

Mód	SHIFT	1	2	3	Význam 4	5	6	7	8	9	0
E	SYMBOL	DEF FN	FN	LINE	OPEN	CLOSE	MOVE	ERASE	POINT	CAT	FORMAT
	CAPS	INK modrá	INK červená	INK purpur	INK zelená	INK sv.modrá	INK žltá	INK biela	FLASH nie	FLASH áno	INK čierna
	ŽIADEN	PAPER modrá	PAPER červená	PAPER purpur	PAPER zelená	PAPER sv.modrá	PAPER žltá	PAPER biela	BRIGHT norm.	BRIGHT zvýšený	PAPER čierna
G	OBA									koniec GRAPHICS	DELETE
	ŽIADEN									koniec	DELETE
K L C	CAPS	EDIT	CAPS LOCK	TRUE VIDEO	INVERS VIDEO					GRAPHICS MOD	DELETE
	SYMBOL	!	@	#	$	%	&	'	(	)	_
	ŽIADEN	1	2	3	4	5	6	7	8	9	0

Zadávať príkazy pre farebné zobrazenie znaku sme sa už naučili.
Niekedy je vhodné vedieť, ako je zobrazený znak na určitej pozícii na obrazovke, presnejšie, aké atribúty má určitá pozícia na obrazovke. Na to použijeme funkciu **ATTR**, ktorá má tvar:

ATTR (*riadok*,*stĺpec*)

kde *riadok* a *stĺpec* majú ten istý význam, ako v príkaze AT. Výsledkom je číslo udávajúce atribúty zodpovedajúceho znaku na obrazovke. Toto môžete využiť samostatne alebo ako časť inej funkcie. Číslo, ktoré je výsledkom tejto funkcie, je súčet 4 ďalších čísiel:

1) 128 pokiaľ pozícia bliká, 0 keď nebliká
2) 64 ak je na pozícii zvýšený jas, 0 keď je normálny
3) 8*n , kde n je kód farby podkladu pre PAPER
4) kód farby pre INK

Napr. nech pozícia bliká s normálnym jasom , žltým podkladom a farba znaku je modrá. Potom čísla, ktoré sa sčítajú, sú 128,0, 8*6=48 a 1 , výsledok je 177.
Skúste príkazový riadok:

```
PRINT AT 0,0; FLASH 1; PAPER 6; INK 1;" "; ATTR (0,0)
```

Teraz skúste:

```
PRINT "B"; CHR$ 8; OVER 1;"/";
```

Dostanete znak B prepísaný znakom / .

Je zaujímavé, že pokiaľ vypíšete to isté 2 x, dostanete späť to, čo tam bolo. Skúste teraz napísať:

```
PRINT CHR$ 8; OVER 1;"/" (dostanete B - prečo?)
```

Zadajte:

```
PAPER 0: INK 0:PRINT "NIC"
```

Čo ste dostali ? Čo by sa stalo, keby to platilo aj v dialógovom riadku?

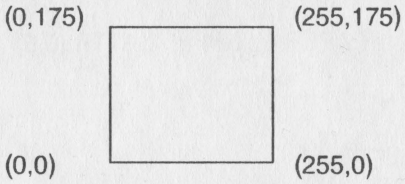
Nakoniec nechajte bežať tento program:

```
10 POKE 22527+RND*704,RND*127
20 GO TO 10
```

Program pracuje tak, že mení farbu pozícií pre výpis znakov, funkcia RND zaisťuje náhodnosť. Diagonálne prvky, ktoré momentálne vidíte, sú dokladom toho, že RND dáva pseudonáhodné číslo.

1.12. Počítač vie kresliť

V tejto kapitole uvidíte, ako sa dajú na počítači kresliť obrázky. Časť obrazovky, ktorú môžete použiť, má 22 riadkov a 32 stĺpcov, tj. 22*32 = 704 znakov. Ako si pamätáte z predchádzajúcej kapitoly, každý z týchto znakov je vytvorený z 8x8 bodov. Každý bod je špecifikovaný dvoma číslami - **súradnicami**. Prvá súradnica je **x** a druhá je **y**. Tieto súradnice sa píšu ako pár v zátvorke. Rohové body kresliaceho poľa sú body so súradnicami.



To zároveň znamená, že pri kreslení môže byť súradnica x len celé číslo od 0 do 255 a súradnica y od 0 do 175. Ak nie je hodnota x a y celé číslo, berie sa do úvahy len celá časť.
V tomto poli máme k dispozícii nasledujúce grafické príkazy:

PLOT *x,y*

nakreslí bod daný súradnicami *x, y*. Súradnice *x, y* sa chápu ako absolútne súradnice (t.j. vzhľadom na ľavý dolný roh).

Nasledujúci program:

```
10 PLOT INT(RND*256),INT (RND*176): INPUT a$: GO TO 10
```

nakreslí bod s náhodnými súradnicami po každom stlačení <ENTER>.

Máme tu však oveľa zaujímavejší program. Kreslí grafy funkcie SIN (sínusovka) pre hodnoty 0 až 2*PI:

```
10 FOR n=0 TO 255
20 PLOT n,88+80*SIN (n/128*PI)
30 NEXT n
```

Ďalší program kreslí graf časti paraboly (funkcia SQR) medzi hodnotami 0 a 4:

```
10 FOR n=0 TO 255
20 PLOT n,80*SQR (n/64)
30 NEXT n
```

Všimnite si, že súradnice bodu pre kreslenie sa líšia od súradnice v AT pre nastavenie ukazovateľa výpisu znaku.
Počítač však dokáže vykresliť nielen bod, ale i úsečku, dokonca aj kružnicu alebo jej časť. Úsečku nakreslíme pomocou príkazu:

DRAW *x,y*

Začiatočným miestom úsečky je bod, v ktorom skončil predchádzajúci grafický príkaz svoju činnosť (po zapnutí počítača a po príkazoch RUN, CLEAR, CLS a NEW je to spodný ľavý roh (0, 0)) a končí bodom s udanými súradnicami *x* a *y*. Súradnice koncového bodu sa berú vzhľadom na začiatočný bod úsečky (relatívne súradnice, na rozdiel od príkazu PLOT). Preto môžu byť hodnoty *x* a *y* v príkaze DRAW aj záporné.

Skúšajte príkazy PLOT a DRAW napr.:

```
PLOT 0,100: DRAW 80, -35
PLOT 90,150: DRAW 80, -35
```

Môžete samozrejme kresliť vo farbách, pamätajte však, že farby platia pre celú pozíciu znaku (8x8 bodov) a nemôžu byť špecifikované pre jednotlivé body. Bod je tlačený farbou INK a farba všetkých bodov v jednej znakovej pozícii je daná tou istou farbou, ako ukazuje tento program:

```
10 BORDER 0: PAPER 0: INK 7: CLS
20 LET x1=0: LET y1=0: REM zaciatok priamky
30 LET c=1: REM farba pre INK - zacina sa modrou
40 LET x2=INT (RND*256):LET y2=INT (RND*176): REM dokoncenie
50 DRAW INK c;x2-x1, y2-y1
60 LET x1=x2: LET y1=y2: REM dalsia ciara zacina tam, kde minula koncila
70 LET c=c+1: IF c=8 THEN LET c=1: REM nova farba
80 GO TO 40
```

Ako program postupuje, čiary sa zdajú širšie a to preto, že menia farbu. Pamätajte, že môžete použiť PAPER, INK, FLASH, BRIGHT, INVERSE, OVER v príkaze PLOT a DRAW, rovnako ako v PRINT a INPUT. Ďalšou možnosťou využitia DRAW je možnosť kresliť časť kružnice namiesto priamok pridaním ďalšieho čísla k príkazu, ktorý špecifikuje uhol pre kružnicový oblúk:

DRAW *x,y,u*

x a *y* udávajú polohu koncového bodu (relatívne) a *u* je uhol v radiánoch udávajúci otáčanie; pokiaľ je kladné, otáča sa doľava, pokiaľ je záporné, tak doprava. Na *u* sa môžeme pozerať tak, že je to časť úplnej kružnice. Celá kružnica je 2*PI radiánov a potom pre *u* = PI bude nakreslená polkružnica, pokiaľ *u* = 0,5 * PI bude to štvrťkruh atď. Predpokladajme, že *u* = PI. Potom nasledujúci program:

```
10 PLOT 100,100: DRAW 50,0,PI
```

nakreslí kruhový oblúk (polkružnicu).
Odštartujte program niekoľkokrát a PI vždy nahraďte iným číslom:

```
-PI,PI/2, 3* PI/2, PI/4, 1,0
```

Posledný príkaz tejto kapitoly je príkaz **CIRCLE**, ktorý kreslí celé kružnice. Má tvar:

CIRCLE *x, y, r*

kde *x* a *y* sú súradnice stredu kružnice v absolútnych súradniciach a *r* je polomer kružnice.
Podobne ako v PLOT a v DRAW môžete použiť rôzne farby.
Funkcia **POINT** udáva, či má bod farbu podľa INK alebo PAPER.
Má tvar:

POINT (*x,y*)

kde *x, y* sú súradnice bodu (musia byť v zátvorkách). Výsledok je 0, keď bod má farbu podľa PAPER a je 1, ak má farbu podľa INK.

Skúste, ako pracuje INVERSE, OVER, PLOT. Majú vplyv len na vlastný bod a na zvyšok pozície znaku. Ak nie sú OVER a INVERSE zadané, je to to isté ako použitie s parametrom 0. Zmeňte ju na 1 a pozrime sa, čo sa stane:

```
PLOT INVERSE 1; - kreslí bod vo farbe papiera
PLOT OVER 1; - zmení bod z čohokoľvek, vymení sa farba INK a PAPER
PLOT INVERSE1; OVER 1; necháva bod ako bol pôvodne
```

Zadajte:

```
PLOT 0,0:DRAW OVER 1;250,175
```

a skúste zmazať nakreslenú úsečku pomocou:

```
DRAW OVER 1; -250, -175
```

Tento spôsob nepracuje presne, pretože body tvoriace druhú úsečku nie sú rovnaké ako body tvoriace prvú úsečku. Čiaru musíte vymazať v tom smere, v akom ste ju kreslili.
Doporučujeme precvičiť položky príkazov PAPER, INK, BRIGHT v príkaze PLOT. Tie platia pre celú znakovú pozíciu, obsahujúcu bod. Používajú sa v tvare:

```
PLOT PAPER a; FLASH b; BRIGHT c;x,y
```

Pokúste sa nakresliť kruhy s použitím funkcií SIN, COS a tiež pomocou príkazu CIRCLE:

```
10 FOR n=0 TO 2*PI STEP PI/180
20 PLOT 100+80*COS n,87+80*SIN n
30 NEXT n
40 CIRCLE 150,87,80
```

Ako vidíte, príkaz CIRCLE je oveľa rýchlejší.
Zadajte:

```
CIRCLE 100,87,80: DRAW 50,50
```

Všimnite si, že po príkaze CIRCLE je nastavený kurzor pre grafické príkazy uprostred pravého okraja kružnice. Preto je vhodné na ďalšie kreslenie použiť PLOT na definovanie počiatočnej pozície.
Nakoniec ukážeme program, ktorý kreslí grafy takmer akejkoľvek funkcie. Na riadku 30 sa zadáva rozsah argumentu a funkcia pre graf. Funkcia sa vkladá ako reťazec, mal by to byť výraz, používajúci *x* ako argument funkcie:

```
10 PLOT 0,87: DRAW 255,0
20 PLOT 127,0: DRAW 0,175
30 INPUT n,e$
35 LET t=0
```

```
 40 FOR f=0 TO 255
 50 LET x=(f-128)*n/128: LET y=VAL e$
 60 IF ABS y>87 THEN LET t=0: GO TO 100
 70 IF NOT t THEN PLOT f,y+88: LET t=1: GO TO 100
 80 DRAW 1,y-stare y
100 LET stare y=INT (y+.5)
110 NEXT f
```

Nechajte program bežať a ako príklad vložte číslo 10 pre *n* a 10*TAN x pre funkciu. Výsledkom bude graf funkcie tangens v rozsahu −10 až +10.

1.13. Funkcie definované užívateľom

Pri programovaní budeme potrebovať často vypočítať hodnotu z neštandardných funkcií, ktoré sa v programe opakujú. Aby sme nemuseli vždy tieto zložité výrazy celé vypisovať, definujeme si vlastnú funkciu. Ak potrebujeme vypočítať hodnotu tejto funkcie, vypíšeme len meno funkcie (hovoríme, že voláme funkciu) s príslušným argumentom. Ak je funkcia raz definovaná, pracujeme s ňou ako so štandardnou funkciou. Pred volaním funkcie ju musíme definovať príkazom **DEF**, ktorý má tvar:

DEF FN *názov (zoznam formálnych argumentov)* = *výraz*

kde *názov* je názov nami definovanej funkcie. Názov funkcie je jedno písmeno (pokiaľ je výsledok číslo) alebo jedno písmeno a znak $ (pokiaľ je výsledkom reťazec). *Zoznam formálnych argumentov* je zoznam premenných oddelených čiarkou, kde mená sú jednopísmenné. *Zoznam argumentov* je nutné písať do uzatvorených zátvoriek. Funkciu definujete tým, že napíšete príkaz DEF niekde do programu. Definícia funkcie môže byť len na jednom programovom riadku.
Definujme funkciu, ktorej výsledkom je druhá mocnina argumentu:

```
10 DEF FN s(x)=x * x :REM druha mocnina x
```

DEF získame v rozšírenom móde použitím <SYMBOL SHIFT> + <1>. Pri vkladaní počítač pridá FN automaticky (pretože v príkaze DEF vždy ihneď nasleduje FN). Rovnítko potom znamená skutočnú definíciu funkcie.

Funkcia môže byť definovaná akýmkoľvek výrazom a môžete sa na ňu odvolávať ako na obyčajnú premennú.
Volanie funkcie sa vykoná takto:

FN *názov (zoznam skutočných argumentov)*

kde *názov* je názov definovanej funkcie, v *zozname* sú *argumenty* oddelené čiarkou. Skutočným argumentom môže byť ľubovoľný odpovedajúci výraz (aritmetický alebo reťazec podľa definície funkcie). Pri volaní funkcie sa formálny argument v definícii nahradí skutočným argumentom (jeho hodnotou), ostatné premenné použité v definícii sa nahradia svojimi hodnotami v okamihu volania funkcie.

Príklad:
Našou úlohou je vypočítať obsah plochy medzikružia, polomer menšej kružnice je 10 cm, väčšej 15 cm. Obsah plochy medzikružia dostaneme tak, že od obsahu väčšieho kruhu odčítame plochu menšieho kruhu. Pretože obsah kruhu budeme počítať dvakrát, použijeme na jeho výpočet funkciu definovanú užívateľom.

```
10 DEF FN o(r) = PI*r*r
20 LET p1 = FN o(10)
30 LET p2 = FN o(15)
40 LET p = p2 - p1
50 PRINT "Obsah medzikruzia je ";p
```

Na riadku 10 definujeme funkciu *o(r)*, ktorá počíta obsah kruhu s polomerom *r*.
Na riadku 20 voláme funkciu *o* so skutočným parametrom 10. Do výrazu na riadku 10 sa dosadí za argument *r* skutočný argument 10 a vypočíta sa hodnota výrazu PI*10*10. Hodnota funkcie sa priradí premennej *p1*.
Na riadku 30 analogicky voláme funkciu *o* so skutočným argumentom 15, vypočíta sa hodnota PI*15*15 a priradí premennej *p2*.
Na riadku 40 sa vypočíta rozdiel obsahov a výsledok sa vypíše (riadok 50).
Teraz si ukážeme definíciu užitočnej funkcie pre zaokrúhlenie. Funkcia INT vždy "zaokrúhľuje" dole. Aby zaohrúhlenie bolo "normálne" (do .5 dole, nad .5 hore), je treba použiť konštantu 0.5. Funkcia bude mať tvar:

```
20 DEF FN r(x) = INT (x + 0.5) :REM zaokruhlenie
```

Potom dostanete:

FN r(2.9) = 3 FN r(2.4) = 2
FN r(−2.9) = −3 FN r(−2.4) = −2

Porovnajte tieto výsledky s výsledkami, ktoré dostanete pri použití funkcie INT *x* namiesto FN r(*x*).

Máme tu ďalší program pre použitie funkcií:

```
10 LET x = 0: LET y = 0: LET a = 10
20 DEF FN p(x,y) = a + x*y
30 DEF FN q( ) = a + x*y
40 PRINT FN p(2,3),FN q( )
```

V tomto programe je viac zaujímavých bodov.
Po prvé, funkcia nie je obmedzená na 1 argument, môže mať viac alebo dokonca žiadny argument, ale stále musíte písať zátvorky. Po druhé, nezáleží na umiestnení DEF v programe, definícia funkcie môže byť na ľubovoľnom mieste programu. Funkcia nemôže byť však definovaná v priamom režime. V našom príklade sa po vykonaní riadku 10 preskočí na riadok 40, kde sa vytlačia hodnoty funkcií.
Po tretie, *x* a *y* sú mená premenných v programe a zároveň mená formálnych argumentov funkcie (čo nemá na hodnotu premenných žiadny vplyv). V definícii funkcie nie je však žiaden argument nazvaný *a*, ale iba premenná *a*. Tak pri vykonaní FN p(2,3) má premenná *a* stále hodnotu 10, pretože ide o premennú, argument *x* má hodnotu 2, pretože je to prvý argument, argument *y* má hodnotu 3, pretože je to druhý argument. Výsledkom je 10 + 2*3 = 16. Keď sa vykoná FN q( ) (funkcia nemá žiadne argumenty), *a*, *x*, *y* sú premenné a majú hodnoty 10, 0, 0. Odpoveď je v tomto prípade 10 + 0*0 = 10. Teraz zmeňte riadok 20 na:

```
20 DEF FN p(x,y) = FN q( )
```

teraz FN p(2,3) bude mať hodnotu 10, pretože FN q sa bude vracať k hodnotám *x* a *y* a argument FN p nie je použitý.

Niektoré verzie jazyka BASIC majú funkcie LEFT$, RIGHT$, MID$ a TL$, kde:

LEFT$ (a$,n) dáva podreťazec z *a$* majúci prvých *n* znakov.
RIGHT$(a$,n) dáva podreťazec z *a$* majúci posledných *n* znakov
MID$(a$,n1,n2) dáva podreťazec z a$, ktorý má n2 znakov a začína na pozícii n1
TL$(a$) dáva podreťazec z a$, ktorý sa skladá zo všetkých znakov mimo prvého

Kto je na tieto funkcie zvyknutý, môže napísať užívateľsky definované funkcie, ktoré robia to isté:

```
10 DEF FN T$(a$) = a$(2 TO): REM TL$
20 DEF FN L$(a$,n) = a$(TO n):REM LEFT$
```

Skontrolujte, či tieto funkcie pracujú s reťazcami s dĺžkou 0 alebo 1. Všimnite si, že naša funkcia *L$* má dva argumenty, jeden je číslo a druhý je reťazec. Funkcia môže mať až 26 číselných argumentov (prečo 26 ?) a zároveň až 26 reťazcových argumentov.
Na pravej strane definície funkcie môžeme použiť tiež ľubovoľnú funkciu (štandardnú i užívateľskú). Preto môžeme zadať napr.:

```
DEF FN s(x) = SIN x
```

1.14. Čo je to podprogram ?

Pri programovaní potrebujeme často vykonávať nejakú činnosť viackrát. Ak je treba túto činnosť vykonávať nepravidelne, nemôžeme použiť príkaz cyklu. Aj s takýmto problémom si vieme poradiť, ak použijeme príkaz pre volanie **podprogramu**. Podprogram voláme príkazom **GOSUB**, ktorý má tvar:

GOSUB *číslo riadku*

Príkaz pre návrat z podprogramu má tvar:

RETURN

Podprogram je relatívne samostatná časť programu.

Pri odštartovaní programu s podprogramom pracuje počítač takto:

Hlavný program sa vykonáva postupne podľa pravidiel doteraz uvedených až po príkaz GOSUB. Na tomto mieste pokračuje vykonávanie programu na *riadku* uvedenom v príkaze GOSUB. Začne sa vykonávať podprogram (hovoríme, že voláme podprogram). Ak vo vykonávaní podprogramu príde počítač po príkaz RETURN (ukončenie podprogramu), vykonávanie programu sa vráti do hlavného programu za príkaz GOSUB, ktorým sme podprogram volali. Po ukončení podprogramu pokračuje vykonávanie programu teda prvým príkazom za príkazom GOSUB. Počítač si vždy "pamätá", z ktorého miesta programu je podprogram volaný a po ukončení podprogramu sa na toto miesto vráti.
Použitie príkazu GOSUB si ukážeme na príklade:

```
 10 CLS
 20 PRINT:PRINT"Ucast v zaujmovych kruzkoch":PRINT
 30 GOSUB 200
 40 PRINT"Modelarsky";TAB(20);12
 50 GOSUB 200
 60 PRINT"Tanecny";TAB(20);21
 70 GOSUB 200
 80 PRINT"Vytvarny";TAB(20);15
 90 GOSUB 200
100 STOP
200 PRINT:PRINT"------------------------"
210 PRINT
220 RETURN
```

Tento program vypíše na obrazovku účasť v záujmových krúžkoch v škole. Činnosť programu je nasledovná:

Riadok 10-20 - mazanie obrazovky a výpis textu
Riadok 30 - volanie podprogramu na riadku 200; začne sa vykonávať program od riadku 200 až po príkaz RETURN na riadku 220 (výpis čiary "---...---" na oddelenie textu). Príkaz RETURN "vráti" program na riadok 40.
Riadok 40 - výpis textu
Riadok 50 - volanie podprogramu, po jeho ukončení návrat za príkaz GOSUB
Riadok 60 - výpis textu
Riadok 70 - volanie podprogramu
Riadok 80 - výpis textu

Riadok 90 - volanie podprogramu
Riadok 100 - koniec hlavného programu
Riadok 200-220 - podprogram, jeho činnosť už bola popísaná

Podprogram sme využili celkom 4 krát na oddelenie výpisu jednotlivých krúžkov od seba. Keby sme nemohli použiť volanie podprogramu, museli by sme namiesto príkazu GOSUB 200 vždy uvádzať celú postupnosť príkazov na riadkoch 200-210.
Na riadku 100 je použitý príkaz STOP na zastavenie programu. Keby sme riadok 100 vynechali, hlavný program by pokračoval vykonávaním podprogramu bez použitia príkazu GOSUB. Ak by sa mal potom vykonať príkaz RETURN, počítač ohlási chybu:

```
7 RETURN without GOSUB, 220:1
```

lebo počítač sa snaží vrátiť na miesto volania podprogramu, ale nevie kam, lebo nebolo použité volanie podprogramu príkazom GOSUB.
Ďalšie vlastnosti podprogramu vysvetlíme po nasledujúcom príklade:

```
 10 LET x=6:LET y=4:LET s=0
 20 GOSUB 100
 30 LET x=7:LET y=3
 40 GOSUB 100
 50 LET x=12:LET y=2
 60 GOSUB 100
 70 GO TO 140
100 LET n=x*y
110 PRINT x;"*";y;"=";n
120 LET s=s+1
130 RETURN
140 PRINT "Podprogram bol volany ";s;" krat."
```

Na riadkoch 10, 30, 50 sa priradia hodnoty premenným *x*, *y*. Podprogram na riadkoch 100 až 130 vypočíta súčin zadaných hodnôt a vypíše výsledok.

Podprogram môže pracovať s tými premennými, s ktorými pracuje hlavný program, dokonca môže tieto hodnoty zmeniť (to je niekedy nevýhoda, ale nedá sa s tým nič robiť). V našom príklade sa mení v podprograme hodnota premennej *s*. Ak sa skončí podprogram, hlavný program môže pracovať s

touto premennou, pričom hodnota premennej zostala nastavená z podprogramu. Na toto musíme dávať pozor. U nás premenná *s* predstavuje počítadlo, koľkokrát bol podprogram volaný.

Na riadku 70 treba použiť príkaz GO TO, aby sa nezačal vykonávať podprogram bez príkazu GOSUB.

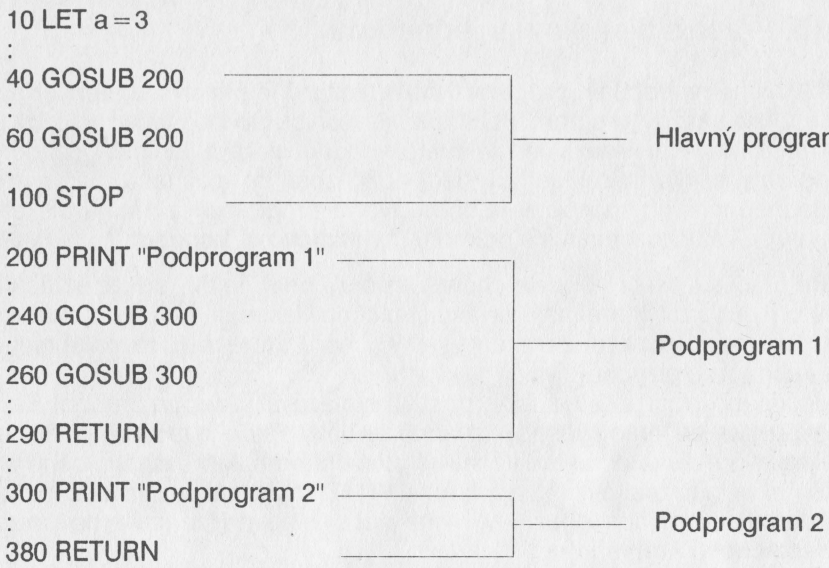
Na uvedených príkladoch je vidieť, že podprogram môžeme volať viackrát z rôznych miest programu.

Každý podprogram by sa mal končiť príkazom RETURN. Ak má podprogram viac logických ukončení (vetvenie), doporučuje sa každú vetvu podprogramu zakončiť skokom na spoločný riadok s príkazom RETURN (kvôli prehľadnosti).

Vo vnútri podprogramu môžeme použiť ďalší príkaz GOSUB pre volanie ďalšieho podprogramu. Takýto postup voláme vloženie podprogramov do seba. Vo vloženom podprograme môžeme použiť opäť príkaz GOSUB atď.

Počet vložených podprogramov je obmedzený len veľkosťou voľnej pamäti pre zásobník.

Správne volanie podprogramov je ukázané na nasledujúcom obrázku:



Na riadkoch 40 a 60 sa volá podprogram 1.
V podprograme 1 sa volá na riadkoch 240 a 260 ďalší (vložený) podprogram 2.
Ak použijete viac podprogramov vložených do seba, príkazy GOSUB a RETURN sa môžu vyskytnúť v poradí GOSUB, GOSUB a RETURN, ale nie v postupnosti GOSUB, RETURN, RETURN.
Kvôli prehľadnosti sa umiestňujú podprogramy na koniec hlavného programu, i keď môžu byť umiestnené na ľubovoľnom mieste programu.
Na koniec si povedzme, že číslo riadku v príkaze GO TO a GOSUB môže byť zadané aritmetickým výrazom. Preto môžeme výstižným menom premennej označiť začiatok podprogramu a na toto meno sa potom odvolávať. Napr:

```
10 LET obsah = 100: LET hlavicka = 200: LET este = 30
:
80 GOSUB hlavicka: GOSUB obsah :GO TO este
```

1.15. Práca s disketovou jednotkou

Ak vypneme počítač, program v ňom uložený je pre nás stratený. Keby sme museli každý program vložiť cez klávesnicu do počítača, bolo by to veľmi zdĺhavé a nezáživné. Je preto vhodné niekam "odložiť" užitočné programy a v prípade potreby ich vložiť späť do počítača. Jedným z najjednoduchších spôsobov je uchovávanie programov a dát na diskete. Najprv sa však zoznámime s pojmom "**hviezdičková konvencia**".

Pri práci s disketou budete chcieť často vykonať nejaký príkaz s väčším počtom súborov, ktoré majú nejakú spoločnú vlastnosť, patria do rovnakej skupiny (napr. vymazať z diskety všetky súbory, ktoré majú rovnaké meno, ale ľubovoľnú príponu).

Jedna možnosť je tieto súbory poctivo vymazávať jeden po druhom. Sami však uznáte, že tento spôsob je značne zdĺhavý. Preto je možné vo väčšine príkazov pre disketu nahradiť znaky v mene súboru, ktoré sú spoločné pre danú skupinu, znakom "*" a znakom "?". Týmto znakom sa tiež hovorí "wildcards" a pri ich použití v mene súboru hovoríme, že sme použili **hviezdičkovú konvenciu**.

Znak "*" slúži k nahradeniu **niekoľkých alebo žiadneho písmena** v mene súboru a hovorí: "na tejto časti mena alebo prípone nezáleží". Uvedieme si príklady:

Príklad 1:

hra*.B budeme pracovať so všetkými súbormi na diskete, ktorých meno začína slovom "hra", za ktorým môžu nasledovať ľubovoľné znaky. Súbory majú príponu "B" (napr. hra1.B, hra2.B, hraNova.B, hrajet.B atď.).

Príklad 2:

a*.* budeme pracovať so všetkými súbormi, ktorých meno začína znakom "a", za ktorým môžu nasledovať ľubovoľné znaky a súbory môžu mať ľubovoľnú príponu (napr. a.B, a.N, auto.p, archiv1.S, a.P atď.).

Príklad 3:

***.N** budeme pracovať so všetkými súbormi na diskete, ktoré majú príponu "N" (napr. data1.N, datasklad.N, pokus.N, teplota.N, znamky.N atď.).

Príklad 4:

. budeme pracovať so všetkými súbormi na diskete s ľubovoľnou príponou. To sú vlastne **všetky** súbory uložené na diskete.

Znak "**?**" slúži k nahradeniu **najviac jedného písmena** v mene súboru a hovorí: "miesto znaku ? môže byť čokoľvek". Uvedieme si príklady:

Príklad 1:

pokus?.S budeme pracovať so všetkými súbormi na diskete, ktoré začínajú slovom "pokus" za ktorým môže nasledovať najviac **jeden** ľubovoľný znak. Súbory majú príponu "S" (napr. pokus.S, pokusB.S, pokus1.S, pokusx.S atď.).

Príklad 2:

pok??.b budeme pracovať so všetkými súbormi na diskete, ktoré začínajú slovom "pok", za ktorým môžu nasledovať najviac **dva** ľubovoľné znaky. Súbory majú príponu "b" (napr. pok.b, pokA.b, pokAB.b, pok00.b atď.).

Príklad 3:

?hra??.P budeme pracovať so všetkými súbormi na diskete, ktoré začínajú ľubovoľným znakom (v tomto prípade tam znak musí byť), za ktorým nasleduje slovo "hra" a za týmto slovom môžu nasledovať najviac **dva** ľubovoľné znaky. Súbory majú príponu "P" (napr. 1hra.P, AhraA.P, UhraLL.P, [hra]1.P, hra!!.P atď.).

Príklad 4:

????dat.N budeme pracovať so všetkými súbormi na diskete, ktoré musia mať na začiatku ľubovoľné **štyri** znaky za ktorými nasleduje slovo "dat". Súbory majú príponu "N" (napr. 1234dat.N, minidat.N, NOVYdat.N atď.).

Príklad 5:

editor.? budeme pracovať so všetkými súbormi na diskete, ktoré majú meno "editor". Súbory majú ľubovoľnú príponu (napr. editor.P, editor.B, editor.C atď.).

Príklad 6:

????????.? budeme pracovať so všetkými súbormi na diskete, ktorých meno má dĺžku najviac **osem** znakov. Súbory majú ľubovoľnú príponu (napr. 1.Q, 12.p, abc.n, xxxxx.C, 12345678.B, d-writer.P atď.)

Príklad 7:

SNAPSHOT??.S budeme pracovať so všetkými subormi na diskete, ktoré majú meno SNAPSHOT, za ktorým nasledujú najviac **dva** ľubovoľné znaky. Súbory majú príponu "S" (napr. SNAPSHOT.S, SNAPSHOT10.S, SNAPSHOT99.S atď).

1.15.1. Príkaz FORMAT

Pred prvým použitím novej diskety (teda diskety, ktorá ešte nebola použitá) je treba ju **naformátovať a dať jej meno**. K tomu slúži príkaz **FORMAT**. Predpokladajme, že chcete naformátovať disketu a dať jej meno "prva". Toto meno budeme ďalej používať v príkladoch.

To však neznamená, že Vy si disketu nemôžete pri formátovaní nazvať iným menom. Pokiaľ si ju nazvete napríklad "disk1", stačí, ak všade v príkladoch nahradíte meno diskety "prva" Vaším menom diskety "disk1".

Je dobré, keď rôznym disketám dáte rôzne meno, napríklad podľa toho, čo je na diskete uložené. Pod týmto menom sa potom musíte v niektorých príkazoch na disketu odvolávať. Pri formátovaní diskety postupujte nasledovne:

1) do mechaniky v počítači vložte disketu. Disketa nesmie byť chránená proti zápisu.
2) zadajte príkaz FORMAT "a:prva" a stlačte <ENTER>.Znak "a:" označuje mechaniku A, ktorá je vstavaná v počítači, meno diskety je "prva".
3) po odoslaní príkazu sa do spodného riadku na obrazovke vypíše hlásenie:

All data will be discarded !
(Proceed = P)

Týmto hlásením Vás systém upozorňuje na to, že formátovaním diskety sa zrušia všetky dáta na diskete !!!

4) po stlačení klávesu <P> alebo <R> sa disketa naformátuje. Formátovanie trvá asi dve minúty. Po stlačení ľubovoľného iného klávesu, (disketu nechcete naformátovať), sa príkaz FORMAT nevykoná.

Poznámka: Klávesy <P> a <R> sú pri potvrdení ľubovoľnej otázky, ktorú Vám položí počítač, rovnocenné.

Po vykonaní príkazu FORMAT sa na obrazovke zobrazí nasledujúca informácia:

Format complete.
Formatted *xxx* good blocks
and *yyy* bad blocks.
Total capacity is 730112 bytes.

0 OK, 0:1

Pre Vás sú dôležité dve čísla. Číslo *xxx* uvedené pred výpisom "good blocks" označuje, koľko blokov (sektorov) je na diskete **voľných** pre ukladanie dát. Číslo *yyy* uvedené pred výpisom "bad blocks" označuje, koľko **zlých** blokov (sektorov), sa na diskete nachádza.

V prípade, že disketa neobsahuje žiadné zlé bloky, číslo *xxx* má hodnotu 1426 a číslo *yyy* má hodnotu 0. Ak disketa obsahuje zlé bloky, číslo *xxx* sa zmenšuje a číslo *yyy* zväčšuje.

Zlé bloky sú tie, ktoré systém nemôže použiť na uloženie dát a môžu sa vyskytovať na disketách, ktoré už boli veľakrát formátované, alebo sú nekvalitné. V prípade, že počet zlých blokov sa po každom ďalšom formátovaní zvýši, je lepšie takúto disketu ďalej nepoužívať.

Číslo 730112 označuje **celkovú kapacitu diskety v bytoch**.

V prípade, že chcete naformátovať disketu vloženú vo vonkajšej mechanike B, je treba v príkaze FORMAT zmeniť znak označenia mechaniky A na znak B.

Poznámka: Meno diskety nemôže byť ani jeden z nasledujúcich znakov: A, B, C, D, R, S, K, P alebo a, b, c, d, r, s, k, p. Tieto znaky sú použité pre názvy mechaník a kanálov.

Po vykonaní príkazu FORMAT sú všetky dáta na diskete stratené a na diskete je vytvorená nasledujúca štruktúra:

- na každej strane diskety sa vytvorí 80 stôp
- každá stopa je rozdelená na 9 sektorov
- každý sektor má veľkosť 512 bytov.

Operačný systém dokáže pracovať iba s takto naformátovanou disketou. To znamená, že diskety naformátované pod iným operačným systémom (napr. MS DOS pre počítače IBM PC) nedokáže Váš systém priamo prečítať, ani na ne priamo zapisovať. V tomto prípade je tiež nutné disketu naformátovať.

Po vykonaní príkazu FORMAT je disketa pripravená pre používanie.

Poznámka: Načítanie a zápis súborov na diskety naformátované pod operačným systémom MS DOS však môže byť vyriešený pomocou konverzného programu.

Príklad: Tento program môže byť použitý v nasledujúcich aplikáciach:

- prevod textu z textového editora pracujúceho na počítači Didaktik Kompakt do textového editora T602 a naopak. V tomto prípade môžete texty písané doma na Didaktiku Kompakt s úspechom spracúvať v práci na počítači IBM PC a taktiež môžete pokračovať v práci doma.
- prevod údajov z databázových programov z počítača IBM PC do počítača Didaktik Kompakt a naopak. Takto si môžete doma pripraviť dáta pre ďalšie spracovanie na počítači IBM PC.
- ďalšie prenos ľubovoľných údajov medzi počítačmi IBM PC a Didaktik Kompakt.

1.15.2. Príkazy LOAD*, SAVE*, MERGE*

Tieto príkazy slúžia k ukladaniu programu a dát na disketu a na ich opätovné nahratie späť do počítača. Program z počítača na disketu uložíme príkazom **SAVE***. Predpokladajme, že v počítači je uložený tento program:

```
 1 PRINT 1
 2 PRINT 2
10 PRINT 10
20 LET x=20
```

Tento program chcete uložiť na disketu. Zadajte príkaz:

```
SAVE* "test"
```

Test je názov programu, pod ktorým bude uložený na disketu. Názov programu môže mať až 10 znakov (písmená alebo číslice). Po odoslaní príkazu klávesom <ENTER> počítač uloží program na disketu. Praktický postup pri nahrávaní programu bude:

1) vložte nafomátovanú disketu do disketovej jednotky
2) zatlačte ju na doraz
3) zadajte SAVE* "test" a stlačte <ENTER>

Po uložení programu na disketu počítač vypíše hlásenie:

```
0 OK, 0:1
```

Ak sa na obrazovku vypíše iné hlásenie, znamená to, že počítač Váš program nemohol uložiť. Zoznam všetkých hlásení o chybách a spôsob ich odstránenia je uvedený v prílohe A. Teraz rozoberieme iba niekoľko najdôležitejších:

Drive is not ready (Retry = R)

Disketová jednotka nie je pripravená k práci (v mechanike nie je disketa, popr. nie je dotlačená na doraz). Po stlačení klávesu <R> alebo <P> sa počítač pokúsi o nové uloženie programu, po stlačení iného klávesu program nie je uložený.

Rewrite old file(s) (Proceed = P)
Pri zápise programu na disketu bol na nej nájdený program s rovnakým menom. Po stlačení klávesu <P> alebo <R> je starý program prepísaný novým, po stlačení iného klávesu program nie je uložený.

r Disk is write protected
Disketa je chránená proti zápisu.

U Disk full
Na diskete už nie je miesto na uloženie ďalšieho programu. Niektoré súbory musíte zmazať, alebo do disketovej jednotky musíte vložiť inú naformátovanú disketu.
Program musíte znova uložiť príkazom SAVE* "test" a <ENTER>.

Správnosť uloženia programu nemusíte kontrolovať, počítač urobí tuto prácu automaticky za Vás.

Ak máte program nahratý na diskete, pre nahratie do počítača použijeme príkaz **LOAD*** v tvare:

LOAD* "test"

Upozorňujeme, že príkazom LOAD* sa starý program a premenné v počítači zmažú.

Po načítaní programu z diskety počítač vypíše hlásenie:

0 OK, 0:1

Ak sa na obrazovku vypíše iné hlásenie, znamená to, že pri čítaní súboru nastala chyba, alebo súbor s daným menom na diskete nie je. Opäť uvedieme niekoľko najdôležitejších hlásení:

Drive is not ready (Retry = R)
Disketová jednotka nie je pripravená k práci (v mechanike nie je disketa, popr. nie je dotlačená na doraz). Po stlačení klávesu <R> alebo <P> sa počítač pokúsi o nové

nahratie programu, po stlačení iného klávesu program nie je z diskety nahraný.

S File not found
Pokúsili ste sa nahrať program, ktorý nie je na diskete uložený.

Na zistenie toho, či sa program na diskete nachádza slúží príkaz CAT (<E-mód>, potom <SYMBOL SHIFT> + <9>). Po jeho zadaní sa na obrazovku vypíšu všetky súbory uložené na diskete. Pomocou tohto výpisu môžete zistit, či sa Váš program na diskete nachádza. Viacej sa o ňom dozviete v nasledujúcej kapitole.

Ak chcete zmazať iba časť starého programu použijete príkaz **MERGE*** v tvare:

MERGE* "test"

ktorý pracuje nasledovne:

- z diskety nahrá do voľnej pamäti nový program
- postupne sa prezerajú čísla riadkov nového programu od najmenšieho po najväčší a porovnávajú sa s číslami riadkov starého programu. Ak sú čísla rovnaké, starý riadok je zrušený a na jeho miesto je zaradený riadok z nového programu. Ak sú čísla rôzne, je nový riadok zaradený do starého programu.
- postupne sa prezerajú všetky premenné starého programu. Ak premenná s rovnakým menom existuje v novom programe, je stará premenná zrušená a nahradená premennou z nového programu. Ak nie, je premenná z nového programu priradená starému programu.

Doteraz uvedené príkazy pre prácu s disketou sú:

SAVE* - nahráva program a premenné na disketu
LOAD* - z diskety nahrá nový program a premenné. Súčasne maže starý program a premenné uložené v počítači.
MERGE* - je podobné LOAD* s výnimkou toho, že neruší starý program a premenné, pokiaľ sú riadky inak očíslované a premenné majú iné označenie.

V každom z týchto príkazov za kľúčovým slovom nasleduje meno programu. Meno môže byť ľubovoľný reťazec (môže to byť teda aj reťazcový výraz, nie iba reťazcová konštanta).

Pre príkaz SAVE* je tento reťazec menom programu, pod ktorým má byť program na disketu uložený. Ďalšie dva príkazy "hľadajú" program s daným menom na diskete. **Meno súboru nesmie byť nikdy prázdny reťazec !**

Ak nechceme nahrať premenné spolu s programom (môžu podstatne predĺžiť záznam), zmažeme ich príkazom CLEAR pred použitím príkazu SAVE*.

Druhá možnosť príkazu SAVE* je táto:

SAVE* *meno* LINE *číslo*

Program je uchovaný tak, že pri spätnom načítaní pomocou LOAD* automaticky nasleduje GOTO na riadok s udaným *číslom*, odkiaľ začína program pracovať.

Zatiaľ jediným druhom informácií, ktoré sme uchovávali na diskete, boli programy spoločne s ich premennými. Sú tu však ďalšie možnosti pre pole a časť pamäti.

S poľom sa pracuje trochu odlišne. Pole môžete uchovať pri použití DATA v príkaze SAVE*:

SAVE* *reťazec* DATA *meno poľa ( )*

Reťazec je meno uchovávaného záznamu na diskete a má ten istý význam ako pri ukladaní programu. *Meno poľa* špecifikuje pole, ktoré chcete uchovať. Môže to byť jedno písmeno (pre číselné pole) alebo písmeno a znak $ (pre reťazcové pole). Pamätajte na zátvorky za menom poľa. Možno si myslíte, že sú zbytočné, ale počítaču to uľahčuje prácu. Musí Vám byť jasný rozdiel medzi reťazcom a menom poľa. Pokiaľ napr. zadáte:

```
DIM b(100)
SAVE* "udaje" DATA b( )
```

potom príkaz SAVE berie pole b a ukladá ho na disketu pod menom "udaje". Príkaz:

```
LOAD* "udaje" DATA b( )
```

hľadá pole na diskete a potom, pokiaľ je pre neho v počítači miesto, zruší v počítači pole b (pokiaľ existuje) a nahrá nové pole b. Príkaz MERGE* nemôžete použiť pri nahrávaní polí.
Takisto môžete uložiť na disketu reťazcové pole. Zadajte:

```
DIM b$(10)
SAVE* "udaje" DATA b$( )
```

Keď nahrávate do pamäti reťazcové pole, neruší to len reťazcové pole s týmto menom, ale tiež reťazec s týmto menom.
Uložiť časť pamäti je možné bez ohľadu na jej ďalšie použitie. Mohol by to byť obsah obrazovky, užívateľom definovaná grafika, atď. Obsah obrazovky uložíme príkazom:

```
SAVE* "obrazovka" CODE 16384,6912
```

Jednotkou pamäti je **byte** (číslo medzi 0 a 255), každý byte má svoju **adresu** (číslo medzi 0 a 65535). Prvé číslo po CODE je adresa prvého bytu, ktorý sa má nahrať na disketu a druhé je počet bytov, ktoré sa nahrajú.

V našom prípade je 16384 adresa prvého bytu v pamäti obrazovky a 6912 je veľkosť pamäti vyhradenej pre obrazovku. Tak uložíme kópiu obsahu obrazovky. Skúste to! Meno "obrazovka" je meno, podobne ako u programu. Na nahranie späť do počítača použijeme:

```
LOAD* "obrazovka" CODE
```

Môžete použiť aj tvar:

LOAD* *meno* CODE *začiatok, dĺžka*

Keď počítač zistí, že dĺžka záznamu na diskete je väčšia, než *dĺžka* v príkaze LOAD*, vypíše sa správa:

```
x File too long
```

Pokiaľ sa *dĺžka* vynechá, bude počítač čítať z diskety bez ohľadu na dĺžku. *Začiatok* udáva adresu, kam sa má uložiť prvý byte (môže byť rozdielny od adresy v príkaze SAVE*). Pokiaľ sú adresy rovnaké, môže sa začiatok aj

dĺžka v príkaze LOAD* vynechať (začiatočnú adresu a dĺžku počítač načíta z diskety).

Príkaz SAVE* meno CODE 16384, 6912 pre uloženie pamäti obrazovky je tak často používaný, že môžete použiť jednoduchší príkaz:

SAVE* *meno* SCREEN$

a pre čítanie z diskety zasa:

LOAD* *meno* SCREEN$

Meno v uvedených príkazoch je akýkoľvek reťazec, pod ktorým je informácia uchovaná na diskete. Mal by sa skladať z ASCII znakov, avšak môže ich byť maximálne 10.

Zatiaľ poznáme 4 druhy informácií, ktoré môžu byť uchované na diskete: program a premenné spolu, číselné pole, reťazcové pole, časť pamäti.

SAVE* uchováva informáciu na diskete pod daným menom. Hlásenie:

F Invalid file name

sa vypíše vtedy, keď je meno prázdny reťazec alebo má viac ako 10 znakov.

Teraz uvedieme kompletný zoznam troch základných príkazov pre prácu s disketovou jednotkou.

Príkaz SAVE*

1. Program a premenné
   SAVE* meno LINE číslo riadku
   uchováva program tak, že po LOAD* automaticky nasleduje GOTO číslo riadku. Ak nie je daná časť LINE, program sa po príkaze LOAD* iba nahrá.

2. Časť pamäti
   SAVE* meno CODE začiatok, dĺžka
   uloží časť pamäti, ktorá začína na uvedenej adrese a má zadanú dĺžku.
   SAVE* meno SCREEN$

je vlastne
SAVE* meno CODE 16384,6912
a uloží obsah obrazovky.

3. Pole
SAVE* meno DATA písmeno ( ) alebo
SAVE* meno DATA písmeno$ ( )
uloží pole, ktorého názov je zadaný za DATA.

Príkaz LOAD*

1. Program a premenné
LOAD* meno
ruší starý program a premenné, nahrá nový program a premenné z diskety. Pokiaľ bol program uchovaný príkazom
SAVE* meno LINE číslo
program sa automaticky odštartuje od riadku *číslo*.
Chyba:

4 Out of memory
sa objaví, keď nie je miesto pre nový program a premenné. V tomto prípade starý program a premenné nie sú zrušené.

2. Časť pamäti
LOAD* meno CODE začiatok, dĺžka
načíta záznam do pamäti.
Pokiaľ je dĺžka v hlavičke na diskete väčšia ako je špecifikované, hlási sa chyba.

LOAD* meno CODE začiatok
nahrá časť pamäti z diskety do počítača, začína sa ukladať od adresy *začiatok*
LOAD* meno CODE
nahrá časť pamäti z diskety do počítača podľa toho, ako bol špecifikovaný v príkaze SAVE*.
LOAD* meno SCREEN$
nahrá časť pamäti z diskety do video pamäti.

3. Pole
LOAD* meno DATA písmeno ( ) alebo
LOAD* meno DATA písmeno$ ( )

ruší akékoľvek pole nazvané písmeno alebo písmeno$ a nahratím z diskety vytvára nové pole toho istého mena.
Chyba:

4 Out of memory
sa objaví, pokiaľ nie je miesto pre nové pole. Staré pole potom nie je zrušené.

Príkaz MERGE*

1. Program a premenné
   MERGE* meno
   spojuje program z diskety s programom v počítači, pripisuje riadky programu alebo premenné do starého programu.

   Chyba:
   4 Out of memory
   sa objaví, pokiaľ nie je dostatok miesta pre starý a nový program spolu.

2. Časť pamäti - nie je možné použiť

3. Pole - nie je možné použiť

Poznámka: U všetkých príkazov môže meno obsahovať i mechaniku alebo meno disku, s ktorým sa má príkaz vykonať.

Tak napr. môžete namiesto mena "test" použiť meno "a: test". V tomto prípade bude program "test" hľadaný na diskete v mechanike A. Pokiaľ použijete meno "b:test", program bude hľadaný na diskete v mechanike B.
Pokiaľ použijete meno "prva:test", potom bude program hľadaný na diskete, ktorá sa volá "prva".

1.15.3. Príkaz CAT

Zatiaľ sme na disketu s menom "prva" uložili štyri rôzne súbory:

- program s menom "test"
- číselné pole b s menom "udaje"
- reťazcové pole b$ s menom "udaje"
- obsah obrazovky s menom "obrazovka"

Tento prehľad máme preto, že súborov je málo, a tiež preto, že ich mená sú uvedené v príručke. Iste však nastane situácia, keď na diskete budete mať uložených mnoho súborov, o ktorých budete chcieť získať nejaké informácie (ako sa súbor volá, či to je program, časť pamäti, aká je jeho dĺžka atď.). K tomu slúži príkaz **CAT**, ktorý umožňuje vypísať informácie o súboroch uložených na diskete.
Zadajte príkaz CAT a stlačte <ENTER>. Na obrazovke sa vypíše nasledujúca informácia (adresár diskety) :

```
Directory of prva

N udaje          503   ----RWED
P test            62   ----RWED
C udaje           13   ----RWED
B obrazovka     6912   ----RWED

4 File(s),          721408 Bytes free.

0 OK, 0:1
```

Za slovom Directory (adresár) sa vypíše meno diskety. V našom prípade má disketa meno prva. V ďalších riadkoch sa vypisujú informácie o uložených súboroch :

Typ súboru
(1.stĺpec) : označenie typu uloženého súboru. Písmeno v tomto stĺpci je totožné s príponou súboru. Význam označenia je nasledujúci :

P - Program	(program v BASICu)
N - Number array	(číselné pole)
C - Character array	(znakové pole)
B - Bytes	(časť pamäti, obrazovka)
Q - seQuence	(sekvenčný súbor)
S - Snap	(SNAPSHOT súbor)

Meno súboru
(2.stĺpec) : meno uloženého súboru (max. 10 znakov).

Dĺžka súboru
(3.stĺpec) : dĺžka uloženého súboru. V prípade programu v BASICu je to dĺžka programu spolu s premennými, v prípade obsahu

pamäti (uloženej pomocou CODE) dĺžka ukladanej časti pamäti. Súbory SNAPSHOT majú vždy dĺžku 49280 bytov. Dĺžka ostatných typov súborov (N, C, Q) závisí od veľkosti prvkov (znakov, čísiel), ktoré sa do nich ukladajú. O veľkosti týchto prvkov sa viacej dozviete v druhej časti príručky, v kapitole venovanej uloženiu dát v počítači.

Príznaky (4.stĺpec) : každý uložený súbor má osem príznakov. Význam a spôsob nastavenia príznakov uvedieme pri popise príkazu LET ATTR. Zatiaľ iba niekoľko poznámok.
Po prvom zápise súboru na disk sú nastavené iba príznaky RWED. Teda zo súboru je možné čítať, do súboru je možné zapisovať, súbor sa môže spustiť ako program (pokiaľ je programom) a súbor je možné vymazať z diskety. Ostatné príznaky nastavené nie sú.

V poslednom riadku je vypísaný počet súborov na diskete (v našom prípade sú na diskete uložené štyri súbory) a rozsah voľného miesta na diskete (v našom prípade je to 721408 bytov).

Z uvedeného výpisu je tiež vidieť, že sme mohli pre číslené a znakové pole použiť rovnaké meno. Tieto súbory sa totiž odlišujú svojou príponou. Príkaz CAT môže mať rôzne modifikácie, podľa toho, aké informácie chceme vypísať. Pre výpis všetkých súborov uložených na diskete v mechanike A môžete použiť variant:

```
CAT "a:"
```

obdobne pre mechaniku B príkaz:

```
CAT "b:"
```

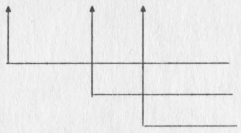
Pre výpis všetkých súborov uložených na diskete s menom "prva" môžete použiť príkaz:

```
CAT "prva:"
```

Týmto sme vypísali všetky súbory uložené na diskete s názvom "prva". Predpokladajme však, že chcete vypísať informáciu iba o niektorých súbo-

roch, napr. o všetkých súboroch, ktoré sa volajú "udaje" a majú ľubovoľnú príponu. V tomto prípade použijete príkaz:

```
CAT "udaje.*"
```



- adresár diskety
- všetky súbory s menom "udaje"
- súbory majú ľubovoľnú príponu

Na obrazovke sa vypíše:

```
Directory of prva

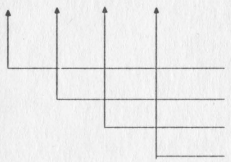
N udaje          503     ----RWED
C udaje           13     ----RWED

2 File(s),       721408 Bytes free.
```

Je zrejmé, že môžete použiť i ďalšie spôsoby zadania požadovanej informácie.

Zadajte:

```
CAT "a:udaje.*"
```



- adresár diskety
- disketa v mechanike A
- všetky súbory s menom "udaje"
- súbory majú ľubovoľnú príponu

Na obrazovke sa vypíše:

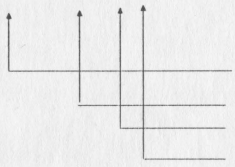
```
Directory of prva

N udaje          503     ----RWED
C udaje           13     ----RWED

2 File(s),       721408 Bytes free.
```

V príkaze CAT môžete použiť namiesto mena mechaniky ("a:","b:") aj meno diskety. Zadajte:

```
CAT "prva:*.p"
```



- adresár diskety
- disketa s menom "prva"
- všetky súbory
- súbory majú príponu "p"

Na obrazovke sa vypíše:

```
Directory of prva

P test                62        ----RWED

1 File(s),            721408 Bytes free.
```

Ďalším variantom je príkaz **CAT-**. Tento príkaz vypisuje skrátený adresár, v ktorom sú uvedené iba typ súboru a jeho meno. Zadajte:

```
CAT-
```

a na obrazovke sa vypíše:

```
Catalogue of prva

N  udaje      C     udaje
P  test       B     obrazovka


4 File(s),            721408 Bytes free.
```

Podobne ako v príkaze CAT sa dajú použiť i ďalšie varianty pre výpis súborov:

CAT- "a:"	- všetky súbory na diskete v mechanike A
CAT- "b:"	- všetky súbory na diskete v mechanike B
CAT- "prva:"	- všetky súbory z diskety s menom "prva:"
CAT- "udaje.*"	- všetky súbory s menom "udaje" a ľubovoľnou príponou
CAT- "A:udaje.*"	- všetky súbory na diskete v mechanike A, ktoré majú meno "udaje" a ľubovoľnú príponu
CAT- "prva:*.p"	- všetky súbory z diskety s menom "prva", s ľubovoľným menom a príponou "p" (teda programy)

1.15.4. Príkaz RUN

Pokiaľ nie je v počítači uložený program, potom príkaz **RUN** slúži k nahratiu programu s menom *run* z diskety. Zadajte príkaz:

```
LOAD*"test"
```

Tým ste do počítača nahrali Váš skôr vytvorený program, ktorý bol uložený na disketu pod menom test. Teraz ho uložíme na disketu pod menom run:

```
SAVE*"run"LINE 10
```

Program bol uložený na disketu pod menom run s autoštartom od riadku 10. Teraz zadajte:

```
CAT
```

a na obrazovku sa vypíše:

```
N    udaje        503     ----RWED
P    test         62      ----RWED
C    udaje        13      ----RWED
B    obrazovka    6912    ----RWED
P    run          62      ----RWED

5 File(s),                721396 Bytes free.

0 OK, 0:1
```

Vidíte, že na diskete je uložený nový program s menom *run*. Teraz zresetujte počítač. Po resete zadajte príkaz:

RUN

Systém sa najprv pozrie, či je na diskete uložený program s menom *run*. Pokiaľ je, nahrá ho do počítača (vlastne za Vás urobí príkaz LOAD*"run") a spustí ho od riadku 10 (teda od riadku autoštartu).
Pokiaľ bol program s menom *run* uložený bez čísla riadku autoštartu, potom je po nahratí do počítača odštartovaný od prvého riadku programu.
Pokiaľ na diskete súbor run nie je, nič sa nestane.

Príkazom RUN je možné nahrať program *run* z diskety iba vtedy, keď v počítači **NIE JE** uložený iný program v BASICu (preto ste pred zadaním príkazu zresetovali počítač). V opačnom prípade má príkaz RUN štandardnú funkciu, teda spustí program uložený v počítači.

1.15.5. Príkaz ERASE

V prípade, že máte na diskete uložené súbory, ktoré už nepotrebujete, môžete ich z diskety **vymazať**. K tomu slúži príkaz **ERASE**. Tento príkaz umožňuje vymazať z diskety jeden alebo viacej súborov, popr. dovoľuje vymazať z diskety všetky uložené súbory. Z toho vyplýva aj jeho opatrné používanie. **Súbory, ktoré boli z diskety vymazané, už nie je možné obnoviť.** Preto pri vymazávaní súborov postupujte veľmi uvážlivo.
Pri vymazávaní soúborov nesmie byť disketa chránená proti zápisu.
Predpokladajme, že chcete z diskety vymazať súbor "run.P". Zadajte:

ERASE "run.p"

a teraz použite príkaz:

CAT

Na obrazovku sa vypíše:

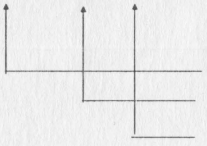
```
Directory of prva

N    udaje          503   ----RWED
P    test            62   ----RWED
C    udaje           13   ----RWED
B    obrazovka     6912   ----RWED

4 File(s),          721408 Bytes free.
```

Súbor "run.p" bol z diskety vymazaný a nie je možné ho už obnoviť v pôvodnej forme. Určite ste si všimli, že pri zadávaní mena sme použili aj príponu (teda použili sme celé meno). Je to z toho dôvodu, že meno súboru musí byť jednoznačne určené, aby pri vymazávaní nemohlo prísť k omylu. Pokiaľ chcete vymazať viacej súborov naraz, môžete použiť tento variant príkazu:

```
ERASE "udaje.*"
```



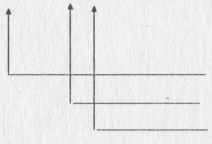
- zrušenie súborov
- všetky súbory s menom "udaje"
- súbory majú ľubovoľnú príponu

Po odoslaní príkazu sa z diskety vymažú všetky súbory ktoré majú meno "udaje" a ľubovoľnú príponu (v našom prípade to bola prípona "N" a "C"). Zadajte:

```
DIM b(100)
DIM b$(10)
SAVE* "udaje" DATA b( )
SAVE* "udaje" DATA b$( )
```

Tým sme na disketu opäť uložili číselné a znakové pole. Skúsime teraz vymazať všetky súbory, ktoré sú programy (v našom prípade je to iba jeden súbor). Zadajte príkaz:

ERASE "*.p"



- zrušenie súborov
- všetky súbory na disku
- súbory majú príponu "p"

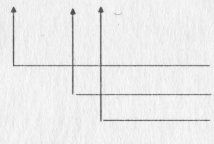
Po odoslaní príkazu Vam počítač položí nasledujúcu otázku:

Erase all files ? (Proceed = P)

Týmto Vás počítač varuje, že chcete **vymazať všetky súbory rovnakého typu**, a pýta sa Vás, či to myslíte naozaj vážne. Po stlačení klávesu <R> alebo <P> sú súbory vymazané, po stlačení iného klávesu vymazané nie sú. Súbor nevymazávajte (stlačte napr. kláves <ENTER>).

Na vymazanie všetkých súborov z diskety musíte zadať príkaz:

ERASE "*.*"



- zrušenie súborov
- všetky súbory na disku
- súbory majú ľubovoľnú príponu

Po odoslaní príkazu Vam opäť počítač položí nasledujúcu otázku:

Erase all files ? (Proceed = P)

Týmto Vás počítač varuje, že chcete vymazať všetky súbory z diskety, a pýta sa Vás, či to myslíte naozaj vážne. Po stlačení klávesu <P> alebo <R> sú všetky súbory vymazané, po stlačení iného klávesu vymazané nie sú. Súbory nevymazávajte (stlačte napr. kláves <ENTER>).

Je zrejmé, že príkaz ERASE môže mať viacej možných tvarov v závislosti na tom, z ktorej diskety alebo mechaniky chcete súbor vymazať. Niektoré z možných tvarov uvádzame:

ERASE "a:udaje.*"	- vymaž z diskety v mechanike A všetky súbory s menom "udaje" a s ľubovoľnou príponou
ERASE "b:*.p"	- vymaž z diskety v mechanike B všetky súbory typu program ("p")
ERASE "prva:obrazovka.b"	- vymaž z diskety s menom "prva" súbor "obrazovka.b"
ERASE "prva:*.*"	- vymaž z diskety s menom "prva" všetky súbory

1.15.6. Príkaz LET FN

Iste príde čas, keď zistíte, že meno niektorého súboru na diskéte Vám nevyhovuje (napr. ste ho zle zadali v príkaze SAVE*) a budete ho chcieť **premenovať**. K tomu slúži príkaz **LET FN**.

Zadajte:

```
LET FN("udaje.N") = "UDAJE.N"
```

a potom príkaz CAT. Na obrazovku sa vypíše:

```
Directory of prva

N    UDAJE        503   ----RWED
P    test          62   ----RWED
C    udaje         13   ----RWED
B    obrazovka   6912   ----RWED

4 File(s),          721408 Bytes free.
```

Z výpisu je vidieť, že súbor "udaje.N" bol premenovaný na súbor "UDAJE.N". Taktiež je zrejmé, že pri premenovaní musíte zadať celé meno súboru, teda aj príponu. Taktiež nie je možné premenovať súbor, ktorý je uložený na diskete chránenej proti zápisu.
Premenovanie súboru budete často používať aj po stlačení klávesu <SNAP>. Zresetujte počítač a potom stlačte súčasne klávesy:

<CAPS SHIFT> + <ľavá šípka> + <pravá šípka>

Na disketu sa uloží súbor s menom "SNAPSHOT00.S". Pokiaľ by ste teraz zresetovali počítač a stlačili znovu < SNAP >, na disketu sa opäť uloží súbor "SNAPSHOT00.S" a prepíše pôvodný súbor. Preto je dobré po stlačení klávesu < SNAP > súbor premenovať.
Najprv sa pozrime, čo je na diskete uložené. Zadajte príkaz CAT:

```
Directory of prva

N    UDAJE          503   ----RWED
P    test            62   ----RWED
C    udaje           13   ----RWED
B    obrazovka     6912   ----RWED
S    SNAPSHOT00   49280   ----RWED

5 File(s),          672128 Bytes free.
```

Zadajte príkaz:

```
LET FN("SNAPSHOT00.S") = "prvysnap.s"
```

a potom príkaz CAT. Na obrazovku sa vypíše:

```
Directory of prva

N    UDAJE          503   ----RWED
P    test            62   ----RWED
C    udaje           13   ----RWED
B    obrazovka     6912   ----RWED
S    prvysnap     49280   ----RWED

5 File(s),          672128 Bytes free.
```

Opäť možete použiť rôzne varianty príkazu LET FN:

```
LET FN("prva:SNAPSHOT00.S") = "prva:prvysnap.s"
LET FN("a:SNAPSHOT01.S") = "a:jetpac.s"
LET FN("b:SNAPSHOT03.S") = "b:moon.s"
```

Všimnite si, že ak sme uviedli pri mene súboru meno mechaniky, alebo meno diskety, MUSÍ byť toto meno uvedené pri starom mene súboru ako aj pri novom mene súboru. Teda nie je možné použiť nasledujúce typy príkazu:

CHYBNÉ ZADANIE PRÍKAZU LET FN

```
LET FN("a:SNAPSHOT03.S") = "moon.s"
LET FN("SNAPSHOT03.S") = "a:moon.s"
LET FN("prva:SNAPSHOT03.S") = "moon.s"
LET FN("SNAPSHOT03.S") = "prva:moon.s"
LET FN("prva:SNAPSHOT03.S") = "druha:moon.s"
LET FN("a:SNAPSHOT03.S") = "b:moon.s"
```

1.15.7. Príkaz MOVE

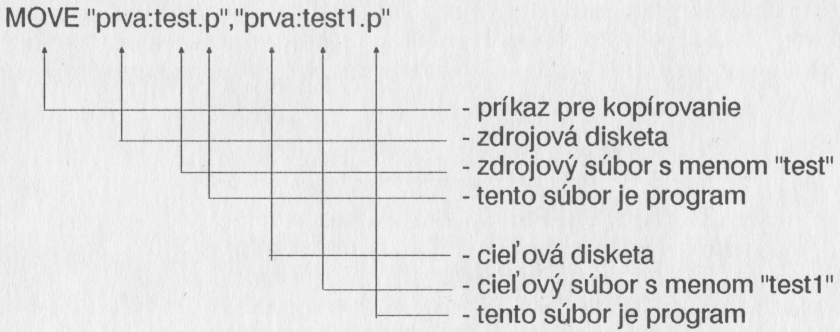
Ďalšou činnosťou, ktorú budete často vykonávať, je **kopírovanie súborov.** K tomu slúži príkaz **MOVE**. Kopírovať súbory môžete buď z jednej diskety na druhú, alebo na tej istej diskete. Pri kopírovaní nemusíte mať vytvorený súbor, do ktorého kopírujete. Tento súbor systém automaticky vytvorí pri kopírovaní po zadaní príkazu MOVE.

Disketa, z ktorej sa kopíruje, sa označuje ako **zdrojová**. Disketa, na ktorú sa kopíruje, sa potom označuje ako **cieľová**. Cieľová disketa nesmie byť chránená proti zápisu. Zdrojová aj cieľová disketa musia byť uvedené menom ("prva:", "druha:", "nova:") a nie označením mechaniky ("a:", "b:"). **Meno diskety MUSÍ byť vždy uvedené.**

Ďalej sa súbor z ktorého kopírujeme označuje ako **zdrojový** a súbor do ktorého kopírujeme sa označuje ako **cieľový**. **Meno** zdrojového aj cieľového súboru **MUSÍ byť uvedené celé**, teda aj s príponou.

Poznámka: V týchto príkladoch používame disketu s menom "prva". Pokiaľ máte disketu s iným menom, musíte v príkladoch meno "prva" nahradiť menom Vašej diskety. Meno zistíte príkazom CAT.

Najprv si uvedieme najjednoduchší príklad použitia príkazu MOVE. Zadajte:



Týmto ste vlastne vytvorili kópiu súboru "test.p" na tej istej diskete a nazvali ste ju test1.p".
Pri tomto type kopírovania musí byť **meno zdrojového súboru rôzne od mena cieľového súboru**. Je to z toho dôvodu, že inak by ste sa vlastne pokúsili kopírovať súbor "sám do seba", čo nie je možné. V tomto prípade počítač vypíše chybu:

F Invalid file name

To znamená, že ste zadali zlé meno zdrojového alebo cieľového súboru. Teraz skopírujeme súbor s menom "test.p", ktorý je uložený na diskete s menom "prva" do súboru s menom "test.p" na diskete s menom "nova".



V tomto prípade Vás však počítač požiada, aby ste disketu s menom "prva" vybrali z mechaniky a vložili do nej disketu s menom "nova". Urobí to nasledujúcim hlásením (po každej výmene je treba stlačiť klaves <R> alebo <P>):

```
Please insert volume nova
(R = Retry)
```

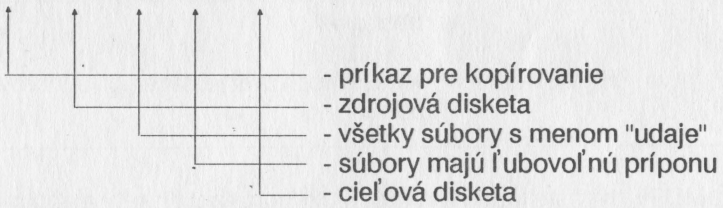
Po výmene skopíruje časť súboru a opäť vypíše žiadosť o výmenu diskety:

```
Please insert volume prva
(R = Retry)
```

Tento postup sa opakuje tak dlho, pokiaľ nie je celý súbor skopírovaný.

Pre kopírovanie väčšieho počtu súborov naraz môžete použiť variant príkazu:

```
MOVE "prva:udaje.*","nova:"
```



Týmto ste počítaču povedali, aby z diskety s menom "prva" skopíroval všetky súbory s menom "udaje" a s ľubovoľnou príponou ("*") na disketu s menom "nova".

V prípade, že chcete kopírovať z jednej diskety na druhú, potom tieto diskety nemôžu mať rovnaké meno !

Pre skopírovanie všetkých súborov z jednej diskety na druhú (napr. z diskety "prva" na disketu "nova") môžete použiť variant príkazu:

```
MOVE "prva:*.*","nova:"
```

V prípade, že máte k počítaču pripojenú druhú mechaniku B, systém sa pri kopírovaní z jednej diskety na druhú najprv pozrie, či nie sú v mechanikách A a B vložené diskety so zadanými menami. Pokiaľ áno, nevypisuje sa žiadosť o výmenu diskety.

Toto sa dá zovšeobecniť na všetky príkazy. Pokiaľ sa v príkaze odvolávate na meno diskety, systém sa vždy najprv pozrie, či disketa s daným menom nie je vložená v niektorej z mechaník A a B, a pokiaľ nie je, žiada o výmenu diskety.

Teraz uvedieme niekoľko chybných zadaní príkazu MOVE:

CHYBNÉ ZADANIE PRÍKAZU MOVE

MOVE "udaje.n","pokus.n"	- nie sú uvedené mená zdrojovej a cieľovej diskety
MOVE "a:test.p","b:"	- zdrojová aj cieľová disketa sú zle zadané (nie je použité meno diskety, ale meno mechaniky)
MOVE "prva:*.P","nova:a.s"	- pokus o kopírovanie viacerých súborov do jedného

Príkaz MOVE má ešte jednu funkciu. Slúži k určeniu diskety alebo mechaniky, na ktorú sa budú obracať všetky príkazy, v ktorých nie je uvedené meno diskety, alebo mechaniky. O takej diskete alebo mechanike potom hovoríme, že je **implicitná (implicitne zadaná)**.
Teraz si za implicitnú disketu zvolíme disketu s menom "prva". Zadajte:

```
MOVE "prva:"
```

Keď teraz zadáte príkaz:

```
LOAD *"test"
```

tak sa tento súbor nahrá z diskety s menom "prva". Pokiaľ by ste mali dve mechaniky a disketa "prva" by bola v mechanike B, potom sa systém najprv

pozrie na disketu v mechanike A. Zistí, že v nej disketa "prva:' nie je, a pozrie sa do mechaniky B. Odtiaľ potom nahrá program "test".
Keď chcete za implicitnú mechaniku zvoliť mechaniku A, potom zadajte príkaz:

MOVE "a:"

Od tohto okamihu sa budú všetky príkazy, v ktorých nie je uvedené meno diskety alebo mechaniky, obracať na mechaniku A. Podobne zvolíme ako implicitnú mechaniku B príkazom:

MOVE "b:"

1.15.8. Príkazy RESTORE* a READ*

Doteraz sme sa zoznámili s jedným spôsobom čítania a zápisu na disketu a to pomocou príkazov SAVE* a LOAD*. Tieto príkazy však umožňujú pracovať iba so súbormi.
Pokiaľ chceme priamo pracovať so sektormi diskety, vtedy musíme použiť príkazy **RESTORE*** a **READ***. Tieto príkazy umožňujú priamy zápis alebo čítanie sektoru z diskety. Najprv si vysvetlíme funkciu príkazu **RESTORE***. Príkaz má tvar:

RESTORE *meno*, *sektor*, *adresa*

kde *meno* môže byť meno mechaniky alebo diskety. V prípade, že meno je prázdny reťazec, potom sa príkaz vzťahuje na implicitnú mechaniku alebo disketu. *Sektor* je číslo sektora, do ktorého sa má zapísať obsah pamäti od adresy *adresa*.

Napíšte krátky program:

```
10 FOR i=0 to 255
20 POKE 30000+i,i
30 NEXT i
40 FOR i=0 to 255
50 POKE 30256+i,0
60 NEXT i
```

Týmto programom sme uložili do pamäti od adresy 30000 po adresu 30255 čísla 0 až 255, a od adresy 30256 po adresu 30511 číslo 0. Teraz zadajte:

RESTORE*"",1439,30000

Týmto príkazom sme do sektora s poradovým číslom 1439 zapísali obsah pamäti od adresy 30000 dĺžky 512 bytov, teda jeden sektor. Pri zápise na disketu sa vždy zapíše celý sektor (512 bytov).

V súvislosti s príkazom RESTORE* Vás chceme upozorniť na jednu veľmi dôležitú vec.

Keď neviete, čo je uložené v sektore kam chcete zapisovať, tak do sektoru nezapisujte !!!.

Môže se Vám totiž stať, že zapíšete do dôležitých oblastí diskety (viď príloha Popis fyzickej štruktúry diskety), určených pre prácu systému. Taktiež môžete prepísať časť súboru uloženého na disketu príkazom SAVE*, alebo SNAP. Preto opakujeme:

Keď neviete, čo je uložené v sektore kam chcete zapisovať, tak do sektora nezapisujte !!!.

V tomto prípade sme mohli na disketu zapísať preto, že na diskete je uložených málo programov, a my sme zapísali do posledného sektora na diskete.

Príkaz **READ*** je opačným príkazom k príkazu RESTORE*. Umožňuje z diskety načítať obsah sektora a uložiť ho do pamäti. Príkaz má tvar:

READ *meno*, *sektor*, *adresa*

kde *meno* môže byť meno mechaniky alebo diskety. V prípade, že meno je prázdny reťazec, potom sa príkaz vzťahuje na implicitnú mechaniku alebo disketu. *Sektor* je číslo sektora, z ktorého sa má čítať a *adresa* je adresa pamäti, na ktorú sa má načítaný sektor uložiť.

Zadajte:

```
READ*"",1439,31000
```

Z diskety sa načíta obsah sektora (teda 512 bytov) a uloží od adresy 31000. Pri čítaní sa opäť ako pri príkaze RESTORE načíta vždy jeden sektor, teda 512 bytov. Keď teraz napíšete program:

```
10 FOR i=0 to 511
20 PRINT PEEK(31000+i)
30 NEXT i
```

tak sa postupne na obrazovku vypíšu čísla uložené od adresy 31000 až po adresu 31511. Najprv sa vypíšu čísla od 0 po 255 a potom sa vypíše ďalších 256 núl. To je vlastne obsah pamäti od adresy 30000, ktorý ste na disketu uložili príkazom RESTORE*.
V prípade, že použijete číslo sektora väčšie, ako je počet sektorov na diskete, systém vypíše hlásenie:

```
Seek error (R=Retry)
```

a po stlačení klávesu <R> alebo <P> hlásenie:

```
p Sector not found
```

Ak zadáte príkaz CAT, zistíte, že na diskete nepribudol žiaden súbor. Je to tým, že sme na disketu zapísali priamo sektor a nie súbor.

V príkazoch READ* a RESTORE* sa môže medzi úvodzovky zadať meno diskety alebo mechaniky. Teda je možné zadať:

```
RESTORE*"a:",1400,40000
RESTORE*"prva:"1300,56000
READ*"b:",1300,28000
READ*"druha:",1,32768
```

Príkazy READ* a RESTORE* majú ešte jeden variant. Do úvodzoviek môžete zadať meno súboru, s ktorým chcete príkaz vykonať. Číslo sektor potom môžete zadať iba v tom rozsahu, ktorý prislúcha danému súboru. Vezmime si napr. súbor "udaje.n" a zadajte príkaz:

```
READ*"udaje.n",0,32768
```

Systém načíta do pamäti od adresy 32768 prvý sektor súboru "udaje.n" (sektory sú číslované od nuly). Teraz zadajte:

```
READ*"udaje.n",1,32768
```

Po odoslaní je vypísané hlásenie:

```
i Bad record number
```

Súbor "udaje.n" má dĺžku 503 bytov. Prvý sektor má dĺžku 512 bytov. Porovnaním dĺžky súboru a dĺžky sektora (503 < 512) zistíme, že súbor "udaje.n" je na diskete uložený iba na jednom sektore. Preto je chybou pokus o načítanie druhého sektora súboru tento sektor súbor nemôže vzhľadom k svojej dĺžke obsadiť a systém na to reaguje hore uvedeným hlásením (v preklade hlásenie znamená "Chybné číslo záznamu").

Opäť sa dá pri zadaní mena súboru použiť aj meno diskety, alebo mechaniky:

```
READ*"prva:udaje.n",0,32768
RESTORE*"a:test.p",0,50000
```

1.15.9. Príkaz LIST*

Príkazom **LIST*** môžete zistiť, aká je súčasná konfigurácia Vášho systému. Zresetujte počítač a zadajte:

```
LIST*
```

a na obrazovke sa vypíše:

```
MDOS Release: 1.0 (01-Sep-1992)
© Didaktik Skalica 1992

Drives defined : A:, B:
Drives installed: A:
```

```
Current device : A:

Volumes Available:
prva

Length of Program : 0
Length of Variables: 1

Top of RAM : 65367
Free Memory: 41153
```

Teraz popíšeme jednotlivé položky výpisu:

MDOS Release : verzia systému MDOS
Drives defined : v systéme sú štandardne definované dve mechaniky A a B
Drives installed : mená pripojených a zapnutých mechaník (v tomto prípade mechanika A)
Current device : meno implicitnej mechaniky (v tomto prípade mechanika A)
Volumes available : mená diskiet vložených v mechanikách; najprv sa vypisuje meno diskety v mechanike A, potom meno diskety vloženej v mechanike B (v tomto prípade sa vypíše meno diskety vloženej v mechanike A, disketa má meno *prva*)
Length of Program : dĺžka programu uloženého v počítači (po resete nie je v počítači uložený žiaden program, preto nulová dĺžka)
Length of Variables : dĺžka oblasti premenných (žiadne premenné)
Top of RAM : hodnota premennej RAMTOP (koniec programovej pamäti je po resete nastavený na hodnotu 65367)
Free memory : veľkosť voľnej pamäti pre programy v BASICu

Pomocou príkazu LIST* teda môžete zisťovať napr. mená diskiet (pre príkaz MOVE), dĺžku programu a premenných, nastavenie RAMTOPu a veľkosť voľnej pamäti.

1.15.10. Príkaz LET ATTR

Ako bolo spomenuté v kapitole 1.15.3. venovanej príkazu CAT, každý súbor má osem príznakov (atribútov), ktoré popisujú, ako sa dá so súborom pracovať. Tieto príznaky sú nastavené pri zápise súboru na disketu. K ich zmene slúži príkaz **LET ATTR**.

Pomocou príkazu môžete zmeniť nastavenie ľubovoľného príznaku na stav aktívny alebo pasívny. **Aktívny stav** je pri výpise pomocou príkazu CAT označený zodpovedajúcim **písmenom**, **pasívny** znakom '-'. Pre označenie príznakov sú použité nasledujúce písmená:

H - Hidden (skrytý súbor; príkaz CAT meno súboru nevypisuje)
S - System (systémový súbor)
P - Protected (zatiaľ nepoužité)
A - Archive (zatiaľ nepoužité)
R - Readable (súbor je možné čítať)
W - Writeable (do súboru je možné zapisovať)
E - Executable (vykonávateľný súbor)
D - Deleteable (súbor je možné na disku zrušiť)

Zadajte príkaz:

```
LET ATTR("obrazovka.b") = "RW"
```

Týmto sme nastavili aktívne príznaky pre súbor "obrazovka.b" nasledovne:

- súbor je možné čítať
- do súboru je možné zapisovať
- ostatné príznaky sú pasívne

Zadajte:

```
ERASE "obrazovka.b"
```

a počítač Vám vypíše hlásenie:

```
h File is delete protected, 0:1
```

Týmto Vám počítač oznámil, že súbor "obrazovka.b" nie je možné vymazať, pretože príznak "D" (súbor je možné zrušiť) nie je nastavený.

Príznaky sa dajú nastavovať celej skupine súborov naraz. Zadajte:

```
LET ATTR("*.*") = "WEDR"
```

Po odoslaní príkazu sa nastavia všetkým súborom na diskete príznaky "RWED". Z príkladu je tiež vidieť, že pri zadávaní znakov pre príznaky nezáleží na poradí znakov.

Príkaz je samozrejme možné doplniť o meno diskety alebo mechaniky.

```
LET ATTR("prva:*.p") = "RWE"
LET ATTR("a:udaje.*") = "RED"
```

1.15.11. Príkazy OPEN# a CLOSE#

Príkazy **OPEN#** a **CLOSE#** slúžia k otvoreniu a zatvoreniu sekvenčných súborov na diskete. Pod pojmom sekvenčný súbor si môžete predstaviť súbor, do ktorého sme čosi zapísali pomocou príkazov PRINT a LIST. Obsah tohto súboru je rovnaký, ako keby sme použili príkazy PRINT a LIST pre výpis na obrazovku. Súbor má štandardnú príponu "**q**".

K **otvoreniu súboru** slúži príkaz **OPEN#**. Každý súbor je pri svojom otvorení označený **číslom**. Pre označenie môžete používať čísla od 2 po 15. Tieto čísla budeme tiež nazývať **linky**. Teda každý súbor je pri svojom otvorení zviazaný s nejakou linkou. Číslo linky sa potom používa pri prístupe k súboru. Pod pojmom prístup rozumieme čítanie alebo zápis. K jednej linke môžu byť pripojené maximálne dva súbory. Jeden môže byť otvorený pre čítanie, druhý pre zápis.

Súbor sa nesmie pripojiť k linkám 0 a 1. Príkaz OPEN# má nasledovný tvar:

```
OPEN#n, VstupnySubor, VystupnySubor
```

kde číslo *n* je číslo linky, ku ktorej sú pripojené *vstupný* a *výstupný* súbor. V príkaze môže byť vynechaný výstupný súbor (k linke je pripojený iba vstupný súbor) a potom má tvar:

```
OPEN#n,VstupnySubor
```

alebo môže byť vynechaný vstupný súbor (k linke je pripojený iba výstupný súbor) a potom má tvar:

OPEN#*n*,,*VystupnySubor*

Všimnite si, že za číslom linky sú uvedené dve čiarky. Je to z toho dôvodu, že sme otvorili iba výstupný súbor, miesto pre vstupný súbor je "prázdne".

Teraz uvedieme jednoduchý príklad. Vytvoríme súbor "pokus.q" a zapíšeme doň čísla od 0 po 255. Napíšte program:

```
10 OPEN#4,,"pokus.q"
20 FOR i=0 TO 255
30 PRINT #4;i
40 NEXT i
50 CLOSE#
```

Program vysvetlíme po riadkoch:

10 Na diskete sa vytvorí súbor s menom "pokus.q" a otvorí sa pre výstup. Súčasne je k súboru pripojená štvrtá linka.

Poznámka: Pokiaľ na diskete súbor s menom "pokus.q" už existuje, je najprv zrušený a až potom je vytvorený nový súbor "pokus.q". Teda nie je možné zapisovať na koniec už vytvoreného sekvenčného súboru.

20 Cyklus od 0 po 255
30 Príkaz PRINT zapíše do súboru pripojeného na štvrtú linku číslo i (teda postupne čísla od 0 po 255)
40 Koniec cyklu
50 Uzatvorenie súboru (príkaz je popísaný ďalej)

Určite ste si všimli, že pri behu programu sa "čosi" zapisovalo na disketu. Práve toto "čosi" boli čísla z 30 riadku. Aby však situácia nebola taká jednoduchá, tieto čísla sa nezapisujú priamo do súboru, ale sa zapíšu najprv do **vyrovnávacej pamäti**. Táto pamäť má dĺžku 512 bytov (jeden sektor), a až po jej naplnení sa celých 512 bytov zapíše do súboru.

Prichádza na rad otázka: "Dobre, ale čo ak sa pri zapisovaní posledných čísiel celá pamäť nezaplní ? Potom sa tieto čísla do súboru nezapíšu,

pretože systém nevie, že už sú všetky". Naozaj je to tak. Posledné čísla sa do súboru nezapíšu. Aby sa do súboru zapísali všetky byty z vyrovnávacej pamäti, musíte systému povedať, že už do súboru nechcete zapisovať a súbor sa musí **uzavrieť**. K tomu slúži príkaz **CLOSE#**. Zadajte:

```
CLOSE #4
```

Po odoslaní sa do súboru zapíše i zvyšok uložený vo vyrovnávacej pamäti, súbor sa zatvorí a odpojí od svojej linky.

Príkaz môžete použiť s číslom linky, ktorú chcete uzavrieť, potom sa uzavrie iba požadovaná linka, alebo vo všeobecnom tvare:

```
CLOSE #
```

kedy sa uzatvoria všetky súbory a odpoja sa všetky linky. Linkám 0 až 3 sa priradí rovnaká hodnota ako po resete (viď druhá časť príručky).

Poznámka: Po prvom zápise do sekvenčného súboru zostáva disketa v mechanike roztočená a je očakávaný ďalší zápis do súboru. Dlhotrvajúcim točením však môže prísť k poškodeniu diskety a strate údajov na nej uložených. Preto doporučujeme súbor s ktorým nepracujete čo najskôr uzavrieť. Tým sa tiež zastaví pohyb diskety.

Zatiaľ sme používali sekvenčný súbor iba pre zápis. Ako už bolo povedané, môže se použiť i pre čítanie. Napíšte nasledujúci program:

```
10 OPEN#4,"pokus.q"
20 FOR i=0 TO 255
30 INPUT#4;x
40 PRINT x
50 NEXT i
60 CLOSE#
```

Program vysvetlíme po riadkoch:

10 Na diskete sa otvorí súbor s menom "pokus.q" a to na čítanie. Súčasne je k súboru pripojená štvrtá linka. (Súbor "pokus.q" bol vytvorený v predchádzajúcom príklade).

20 Cyklus od 0 po 255
30 Príkaz INPUT prečíta zo súboru pripojeného na štvrtú linku číslo x (teda postupne čísla od 0 po 255)
40 Vypíše sa číslo x
50 Koniec cyklu
60 Uzatvorenie súboru

Teraz uvedieme program, ktorý:

- vytvorí súbor "out.q"
- načíta zo súboru "pokus.q" čísla od 0 do 255
- číslo prevedie na reťazec a pridá k nemu reťazec "ret"
- reťazec uloží do súboru "out.q"

```
10 OPEN#15,"pokus.q","out.q"
20 FOR i=0 TO 255
30 INPUT#15;x
40 LET x$="ret"+STR$(x)
50 PRINT#15;x$
60 NEXT i
70 CLOSE#
```

Obdobne ako u všetkých predchádzajúcich príkazov je možné k menu súboru pridať meno mechaniky alebo diskety.

```
OPEN#5,"a:pokus.q","test:out.q"
OPEN#6,"disk1:in.q"
OPEN#9,,"b:out.q"
```

Poznámka: Pri zápise do sekvenčného súboru musí byť číslo linky oddelené od zvyšku príkazu znakom bodkočiarka (PRINT#3;a). Pokiaľ použijete znak čiarka (PRINT#3,a), do súboru sa zapíše jej riadiaci kód (6) a potom hodnota čísla a.

Poznámka: V dôsledku chyby operačného systému počítača Sinclair ZX Spectrum (a teda aj počítačov Didaktik Gama, Didaktik M a Didaktik Kompakt) dôjde pri pokuse o uzatvorenie už uzatvorených liniek č.4 až č.15 k strate kontroly operačného systému nad prácou počítača.

1.15.12. Príkaz PRINT*

Zatiaľ ste sa naučili, ako vytvoriť sekvenčný súbor, ako doň zapísať a z neho prečítať jeden údaj. Zo sekvenčného súboru sa však dajú prečítať a vypísať všetky údaje naraz. K tomu slúži príkaz **PRINT***. Zadajte:

```
10 CLS
20 PRINT*"out.q"
```

a na obrazovku sa vypíšu postupne reťazce od 0 po 255 tak, ako ste ich do súboru "out.q" uložili v predchádzajúcom príklade.
Pokiaľ sa hlbšie zamyslíte nad využitím sekvenčných súborov, prídete určite na nasledujúcu vec: sekvenčné súbory sa môžu využiť predovšetkým na výpisy formulárových obrazoviek a pod. Napíšte program a odštartujte:

```
10 CLS
20 PRINT " ┌──────────────────────────────
──────────┐ "
30 FOR i=1 TO 20:PRINT AT i,0;"|";AT i,31;"|":NEXT i
40 PRINT " └──────────────────────────────
──────────┘ ";
```

Na obrazovke sa vypíše obdĺžnik. Tento výpis Vám zabral 4 riadky a asi 140 bytov programu. Pokiaľ by ste mali do tohto obdĺžnika ešte niečo vypísať, program sa predĺži o ďalšie byty. Rozumnejšie je vytvoriť si sekvenčný súbor na diskete, ktorý bude obsahovať všetky znaky, ktoré použijete pri výpise príkazom PRINT (teda aj AT, INK, PAPER apod). Potom iba zavoláte na vhodnom mieste príkaz PRINT* a na obrazovku sa vypíše obsah súboru. Takto ušetríte veľa bytov programu a dokonca aj čas nutný pre výpis. Teraz si skúsime uložiť výstup programu do súboru. Zadajte:

```
OPEN#2,,"form.q"
```

Týmto sme otvorili súbor "form.q" a pripojili ho k druhej linke. Druhá linka je však štandardne napojená na obrazovku (viď druhá časť príručky). Tomuto postupu hovoríme, že sme presmerovali výstup z obrazovky na výstup do súboru. Preto teraz nemusíme písať prikaz PRINT#2, ale stačí použiť príkaz PRINT, ktorý je pôvodne určený na prácu s obrazovkou.
Všetky znaky písané na obrazovku pomocou príkazu PRINT (teda aj INK, PAPER, AT, TAB,...) budú zapísané do súboru "form.q". Zadajte:

```
RUN
```

Tým sa všetky znaky napísané programom zapíšu do súboru "form.q". Zadajte:

```
CLOSE#
```

Súbor sa uzatvorí a druhá linka je opäť pripojená na obrazovku. Zadajte príkaz:

```
PRINT#"form.q"
```

Na obrazovku sa vypíše rovnaký rámček, ako po odštartovaní vyššie uvedeného programu.

Príkaz je samozrejme možné doplniť o meno diskety alebo disku.

```
PRINT#"test:form.q"
PRINT#"a:pokus.q"
PRINT#"b:test.q"
```

1.15.13 Príkaz POKE#

Príkaz **POKE#** má rovnaký význam, ako štandardný príkaz POKE s tým rozdielom, že dáta sú zapisované do **tieňovej pamäti RAM**. Táto pamäť je uložená od adresy 14336 (3800h), má veľkosť 2048 bytov. Je určená pre spoluprácu systému s disketovou jednotkou a sú v nej uložené dôležité dáta. Príkaz má tvar:

POKE# *adresa*, *hodnota*

kde *adresa* je adresa v tieňovej RAM. Adresovanie pamäti začína adresou 0. Zadajte:

```
POKE# 97,20
```

Týmto ste na adresu 97 v tieňovej pamäti uložili hodnotu 20. Iste Vás zaujíma, prečo práve na adresu 97. Uvedieme niektoré dôležité adresy v tieňovej RAM:

Adresa Význam

0 parametre prvej disketovej jednotky, dĺžka tabuľky je 12 bytov
6 počet stôp na diskete (implicitne 80 stôp pre disketovú jednotku 3 1/2" a 40 stôp pre jednotku 5 1/4").
7 počet sektorov na stopu (implicitne 9 sektorov na stopu).
12 parametre druhej disketovej jednotky, dĺžka tabuľky je 12 bytov
18 počet stôp na diskete (implicitne 80 stôp pre disketovú jednotku 3 1/2" a 40 stôp pre jednotku 5 1/4").
19 počet sektorov na stopu (implicitne 9 sektorov na stopu).
24 parametre tretej disketovej jednotky (v systéme nie je použitá).
36 parametre štvrtej disketovej jednotky (v systéme nie je použitá).
48 zóna dĺžky 48 bytov určená pre mená štyroch diskov. Opäť je použitých iba 24 prvých bytov pre dve mená diskov.
97 byte, v ktorom je uložené číslo pre nasledujúci SNAPSHOT.
98 0-pri zápise súboru na disketu je nutné potvrdiť prepísanie starého súboru rovnakého mena a typu (pokiaľ existuje).
1-starý súbor rovnakého mena a typu je automaticky prepísaný novým.

Použitie tohto príkazu s nesprávnou hodnotou pre adresu alebo dáta môže viesť k reštartu systému, popr. k narušeniu logickej štruktúry diskety a ku strate dát uložených na diskete. Tento príkaz nedoporučujeme používať okrem hore uvedených adries.

UPOZORNENIE!

Výrobca zaručuje správnu činnosť disketových jednotiek iba pri uvedenom implicitnom nastavení počtu stôp a sektorov.

1.15.14. Kláves SNAP

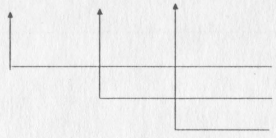
V príručke "Začíname" ste sa zoznámili s použitím klávesu <SNAP>. Teraz podrobnejšie popíšeme, čo sa vlastne po stlačení tohoto klávesu stane.

Najprv sa preruší činnosť programu, ktorý práve v počítači beží a zavolá sa podprogram na obsluhu nemaskovateľného prerušenia (obslužný program NMI). Tento program je umiestnený v pamäti ROM určenej pre MDOS od adresy 102 (66h) a vykoná nasledujúce kroky:

- na diskete vytvorí súbor s menom SNAPSHOT*xx*, kde *xx* je poradové číslo SNAPSHOTu. Toto číslo je umiestnené v systémovej premennej SNAP_N. Hodnota tejto premennej môže byť od 0 po 99 a po dosiahnutí hodnoty 99 sa opäť nastaví na hodnotu 0. Hodnota 0 je tiež nastavená pri každom resete počítača. Z tohoto dôvodu Vám doporučujeme premenovať súbory s menom SNAPSHOT*xx* čo najskôr, inak môže po resete počítača a stlačení klávesu <SNAP> dôjsť k prepísaniu starého súboru SNAPSHOT.
- do vytvoreného súboru SNAPSHOT*xx* uloží obsah celej pamäti od adresy 16384 (4000h) až po adresu 65535 (FFFFh) (dĺžka tohoto bloku je teda 49152 bytov)
- do súboru taktiež zapíše informácie o stave procesora v okamihu stlačenia klávesu <SNAP> a to:
  - obsah všetkých registrov procesora (teda aj registrov SP a PC)
  - nastavený mód prerušenia (IM0, IM1, IM2)
  - informáciu o tom, či je prerušenie povolené alebo zakázané (dĺžka tohoto bloku je 128 bytov)
  - súbor uzavrie
  - zvýši obsah systémovej premennej SNAP_N o jednotku
  - pokračuje sa vo vykonávaní pôvodného programu

Z vyššie uvedených informácií vyplýva, že súbor SNAPSHOT má dĺžku 49280 bytov (49125 + 128) a na jednu disketu môže byť uložených najviac:

INT (720*1024/49280) = 14 súborov typu SNAPSHOT



kapacita diskety v kB
1 kB = 1024 bytov
veľkosť SNAPSHOT súboru

Toto platí iba za predpokladu, že na diskete nie sú uložené ďalšie súbory. Pokiaľ chcete zistiť, koľko súborov typu SNAPSHOT sa Vám ešte zmestí na disketu, stačí, keď si necháte vypísať adresár príkazom:

CAT

a voľné miesto na disku vydelíte veľkosťou SNAPSHOT súboru (teda číslom 49280) a zaokrúhlite príkazom INT smerom dole. Môžeme napísať:

$$\text{počet SNAPSHOTov} = \text{INT}\left(\frac{\text{voľné miesto na disku}}{49280}\right)$$

Pri nahrávaní súboru typu SNAPSHOT do počítača sa postupuje opačne, ako pri ukladaní:

- od adresy 16384 sa nahrá obsah súboru SNAPSHOT*xx*, teda sa zaplní celých 48 kB pamäti od adresy 16384 (4000h) až po adresu 65535 (FFFFh).
- všetky registre procesora sa nastavia podľa uložených hodnôt
- nastaví sa mód prerušenia a prerušenie sa buď povolí alebo zakáže (opäť podľa uložených hodnôt)
- program sa odštartuje od toho istého miesta, na akom sa nachádzal pri stlačení klávesu < SNAP >

Poznámka: Nie vždy sa dá program uložený pomocou klávesu < SNAP > znova nahrať do počítača a úspešne odštartovať. Nie je to možné vtedy, ak v okamihu stlačenia klávesu < SNAP > program pracoval špecifickým spôsobom so zásobníkom (napríklad údaje v ňom uložené používal ako dáta). Preto Vám doporučujeme program ukladať pomocou klávesu < SNAP > v okamihu, keď je nahraný program v nejakej čakacej slučke, napríklad čaká na stlačenie klávesu.

Týmto sme ukončili popis všetkých príkazov BASICu určených pre prácu s disketovou jednotkou.

1.16. Práca s magnetofónom

Pre ukladanie programu môžeme tiež použiť magnetofón, pripojenie k počítaču bolo popísané v prvej príručke "Začíname".
Pre prácu s magnetofónom sú určené príkazy **SAVE, LOAD, VERIFY** a **MERGE.**
Program z počítača na kazetu nahráme príkazom **SAVE.**

Predpokladajme, že v počítači je uložený tento program:

```
 1 PRINT 1
 2 PRINT 2
10 PRINT 10
20 LET x=20
```

Tento program chcete uložiť na kazetu. Zadajte príkaz:

```
SAVE "test"
```

Test je názov programu, pod ktorým bude uložený na pásku. Názov programu môže mať až 10 znakov (písmená alebo číslice). Po odoslaní príkazu počítač vypíše:

```
Start tape, then press any key
```

čo znamená v preklade "pusti magnetofón a stlač nejaký kláves". Po stlačení klávesu uvidíte na okraji obrazovky vodorovné bledomodré a červené pruhy široké asi 1 cm, ktoré sa pohybujú pomaly hore po dobu asi 5 sekúnd. Potom nasleduje krátky úsek modrých a žltých pruhov, nasleduje obrazovka bez pruhov a hneď opäť bledomodré a červené pruhy a nakoniec modré a žlté pruhy.

Informácie sú z počítača na magnetofón vysielané v dvoch blokoch, kde každý blok má zavádzaciu časť (bledomodré a červené pruhy), po ktorej nasledujú informácie o hlavičke (modré a žlté pruhy). Prvý blok (hlavička záznamu) obsahuje názov a ďalšie informácie o zázname, druhý blok sú vlastné informácie (program a premenné).
Biela časť medzi blokmi je medzera.

Praktický postup pri nahrávaní programu bude:

1) pripojte magnetofón k počítaču
2) nastavte kazetu na úsek, na ktorý chcete nahrávať (pozor na zavádzaciu časť kazety)
3) zadajte SAVE "test" a stlačte <ENTER>
4) nastavte úroveň pre záznam podľa návodu pre magnetofón a spustite magnetofón na nahrávanie záznamu
5) stlačte nejaký kláves
6) pozorujte na obrazovke priebeh nahrávania; po ukončení nahrávania sa vypíše hlásenie:

   0 OK, 0:1

   Potom môžete zastaviť magnetofón.

Nahratý program na páske doporučujeme ihneď skontrolovať príkazom **VERIFY** nasledovným postupom:

1) previňte kazetu pred začiatok záznamu
2) zadajte príkaz

   VERIFY "test"

3) pustite magnetofón

Pokiaľ nie je na páske začiatok nahrávky, striedajú sa na okraji obrazovky farby červená a bledomodrá. Potom sa objaví rovnaký vzor pruhov ako pri zázname na pásku. Po prečítaní hlavičky sa objaví výpis:

   Program: test

a začína kontrola záznamu. Kontrola sa vykonáva tak, že záznam na páske sa porovnáva s obsahom pamäti v počítači (preto treba kontrolu vykonať ihneď bez zmeny programu v počítači).
Po úspešnej kontrole nasleduje známy výpis OK.
Ak sa tak nestalo, príčin môže byť viac. Skontrolujte prepojenie počítača a magnetofónu. Ak máte v magnetofóne reproduktor, mali by ste pri prehrávaní počuť zvuk podobný "morseovke". Keď sa neobjavia vôbec pruhy ani

výpis hlavičky, prepojenie je nesprávne alebo úroveň záznamu je zle nastavená.
Ak sa po výpise hlavičky prestanú zobrazovať pruhy, skontrolujte meno programu. Počítač totiž vypisuje všetky hlavičky na obrazovku, no čaká až na program zadaného mena.

Niekedy sa stane, že hlavička je vypísaná, ale objaví sa hlásenie:

R Tape loading error

teda k chybe došlo v priebehu porovnávania. Nastavte kazetu pred záznam a skúste znovu príkaz VERIFY. Ak to nepomôže, nahrávka na páske je zlá (špinavá hlava,nekvalitná páska,...) a treba skúsiť program nahrať znovu. Ak ani to nepomôže, skúste inú pásku.

Ak máte program nahratý na páske, pre nahratie do počítača použijete príkaz LOAD v tvare:

LOAD "test"

Ak bol program uložený na pásku riadne skontrolovaný, nemal by byť problém pri použití príkazu LOAD. Problém môže byť však pri nahrávaní programu z pásky nahratej na inom magnetofóne. Samozrejme, že programy pre iné typy počítačov (nekompatibilné so SINCLAIR) sa Vám nepodarí nahrať.

Upozornenie: Príkazom LOAD sa starý program a premenné v počítači zmažú

Ak starý program nechceme zmazať, použijeme príkaz **MERGE** v tvare:

MERGE "test"

ktorý nahradí odpovedajúce riadky programu a hodnoty odpovedajúcich premenných z nového programu, nové riadky pridá (ako keby sme riadky zadávali postupne z klávesnice).

Doteraz uvedené príkazy pre prácu s magnetofónom sú:

SAVE nahráva program a premenné na pásku

VERIFY	kontroluje program a premenné na páske s tými, ktoré sú v počítači
LOAD	z pásky načíta nový program a premenné. Súčasne maže starý program a premenné.
MERGE	je podobné LOAD s výnimkou toho, že neruší starý program a premenné, pokiaľ sú riadky inak očíslované a premenné majú iné označenie.

V každom z týchto príkazov je kľúčové slovo nasledované *menom programu*, čo môže byť ľubovoľný reťazec (môže to byť teda aj reťazcový výraz, nie iba reťazcová konštanta). Pre SAVE je to *meno programu*, pod ktorým má byť na pásku uložený, zatiaľ čo ďalšie 3 "hľadajú" program na páske. Počas hľadania sa vypisujú mená záznamov, na ktoré sa narazilo. Identifikácia mena môže byť dvojaká. Pre VERIFY, LOAD a MERGE môžete použiť prázdny reťazec ako meno, ktoré hľadáte. Potom sa počítač nezaujíma o meno, ale berie prvý program, na ktorý narazí. Ak nechceme nahrať premenné spolu s programom (môžu podstatne predĺžiť záznam), zmažeme ich príkazom CLEAR pred použitím príkazu SAVE.

Druhá možnosť príkazu SAVE je táto:

SAVE *meno* LINE *číslo*

Program je uchovaný tak, že pri spätnom čítaní pomocou LOAD sa automaticky skáče na riadok s udaným číslom, odkiaľ začína program pracovať. Zatiaľ jediným druhom informácií, ktoré sme uchovávali na páske, boli programy spoločne s ich premennými. Sú tu však ďalšie možnosti pre pole a časť pamäti.

S poľom sa pracuje trochu odlišne. Pole môžete uchovať pri použití DATA v príkaze SAVE:

SAVE *reťazec* DATA *meno poľa* ( )

Reťazec je meno v uchovanom zázname na páske a má rovnaký význam, ako keď uchovávame program alebo časť pamäti. *Meno poľa* špecifikuje pole, ktoré chcete uchovať; ide o jedno písmeno (pre číselné pole) alebo písmeno a znak $ (pre reťazcové pole). Pamätajte na zátvorky za menom

poľa. Možno si myslíte, že sú zbytočné, ale počítaču to uľahčuje prácu. Musí Vám byť jasný rozdiel medzi reťazcom a menom poľa. Pokiaľ zadáte:

```
SAVE "udaje" DATA b( )
```

potom príkaz SAVE berie pole *b* a ukladá ho na pásku pod menom "udaje". Keď napíšete:

```
VERIFY "udaje" DATA b( )
```

počítač bude hľadať číselné pole uložené na páske pod menom "udaje". Keď ho počítač nájde, oznámi:

```
Number array: udaje
```

a kontroluje ho spätne proti poľu b v počítači.

```
LOAD "udaje" DATA b( )
```

hľadá pole na páske a potom, pokiaľ je pre neho v počítači miesto, zruší v počítači pole *b* (pokiaľ existuje) a nahrá nové pole *b*. Príkaz MERGE nemôžete použiť pri uchovávaní polí.
Takisto môžete uložiť na pásku reťazcové pole. Keď ho počítač nájde na páske, vypíše:

```
Charackter array: meno
```

Keď nahrávate do pamäti reťazcové pole, neruší to len reťazcové pole s týmto menom, ale tiež reťazec s týmto menom.
Uložiť časť pamäti je možné bez ohľadu na jej ďalšie použitie. Mohol by to byť obsah obrazovky, užívateľom definovaná grafika, atď. Obsah obrazovky uložíme príkazom:

```
SAVE "obrazovka" CODE 16384,6912
```

Jednotkou pamäti je byte (číslo medzi 0 a 255), každý byte má svoju adresu (číslo medzi 0 a 65535). Prvé číslo po CODE je adresa prvého bytu, ktorý sa má nahrať na pásku a druhé je počet bytov, ktoré sa nahrajú. V našom prípade je 16384 adresa prvého bytu v pamäti obrazovky a 6912 je počet

bytov v nej obsiahnutých. Tak uložíme kópiu obsahu video pamäti. Skúste to! Meno "obrazovka" je meno, podobne ako u programu. Na nahranie späť do počítača použijeme:

LOAD "obrazovka" CODE

Môžete použiť aj tvar:

LOAD *meno* CODE *začiatok, dĺžka*

Keď počítač zistí, že dĺžka záznamu na páske je dlhšia, než v príkaze LOAD, vypíše sa správa:

R Tape loading error

Pokiaľ sa dĺžka vynechá, bude počítač čítať pásku bez ohľadu na dĺžku. Začiatok udáva adresu, kam sa má uložiť prvý byte (môže byť rozdielny od adresy v príkaze SAVE). Pokiaľ sú adresy rovnaké, môže sa adresa v príkaze LOAD vynechať (začiatočnú adresu počítač načíta z pásky).
Príkaz SAVE meno CODE 16384, 6912 pre uloženie pamäti obrazovky je tak užitočný, že môžete použiť jednoduchší príkaz:

SAVE meno SCREEN$

a pre čítanie z pásky zase:

LOAD meno SCREEN$

Tu je zriedkavý prípad, kedy VERIFY nebude pracovať, lebo VERIFY po prečítaní hlavičky z pásky napíše meno na obrazovku (tým zmení pamäť obrazovky) a tak pri verifikácii, kedy je obraz zmenený, dochádza k verifikačnej chybe. Vo všetkých ostatných prípadoch by ste mali použiť príkaz VERIFY po použití SAVE.
Meno v uvedených príkazoch je akýkoľvek reťazec, pod ktorým je informácia uchovaná na páske. Mal by sa skladať z ASCII znakov, avšak môže ich byť maximálne 10.

Poznáme 4 druhy informácií, ktoré môžu byť uchované na páske: program a premenné spolu, číselné pole, reťazcové pole, časť pamäti.

Keď VERIFY, LOAD a MERGE hľadajú informácie na páske s uvedeným menom, potom sa na obrazovke vypisujú mená záznamov na páske, ktoré sa našli. **Pred menom záznamu je informácia o type záznamu: program, number array, character array, bytes.**

Pokiaľ bolo *meno* prázdny reťazec, vezme sa do úvahy prvý záznam zadaného typu bez ohľadu na jeho meno.

SAVE uchováva informáciu na páske pod daným *menom*. Hlásenie:

F Invalid file name

sa vypíše vtedy, keď je *meno* prázdny reťazec alebo má viac ako 10 znakov. SAVE vždy vyvolá správu:

Start tape,then press any key

a počítač čaká na stlačenie klávesu.

Teraz uvedieme kompletný zoznam štyroch príkazov pre prácu s magnetofónom.

Príkaz SAVE

1. Program a premenné
   SAVE *meno* LINE *číslo riadku*
   uchováva program tak, že po LOAD automaticky nasleduje GO TO *číslo riadku*. Ak nie je daná časť LINE, program sa po LOAD iba nahrá.
2. Časť pamäti
   SAVE *meno* CODE *začiatok, dĺžka*
   uloží časť pamäti, ktorá začína na uvedenej adrese a má zadanú *dĺžku*
   SAVE *meno* SCREEN$
   je vlastne
   SAVE *meno* CODE 16384,6912
   a uloží obsah obrazovky
3. Pole
   SAVE *meno* DATA *písmeno* ( ) alebo
   SAVE *meno* DATA *písmeno$* ( )
   uloží pole, ktorého meno je zadané za DATA

Príkaz VERIFY

1. Program a premenné
   VERIFY *meno*
   kontroluje program na páske
2. Časť pamäti
   VERIFY *meno* CODE *začiatok, dĺžka*
   pokiaľ je záznam na páske dlhší ako je zadaná dĺžka, dostávame chybu. Inak porovnáva záznam s časťou pamäti počítača od zadanej adresy.
   VERIFY *meno* CODE *začiatok*
   porovnáva záznam na páske s časťou pamäti a začína od uvedenej adresy, dĺžka sa berie z hlavičky.
   VERIFY *meno* CODE
   porovnáva záznam na páske s časťou pamäti, adresa začiatku a dĺžka sa berú z hlavičky
   VERIFY *meno* SCREEN$ je to isté ako
   VERIFY *meno* CODE 16384,6912
3. Pole
   VERIFY *meno* DATA *písmeno* ( ) alebo
   VERIFY *meno* DATA *písmeno*$ ( )
   kontroluje pole na páske s poľom v pamäti.

Príkaz LOAD

1. Program a premenné
   LOAD *meno*
   ruší starý program a premenné, nahrá nový program a premenné z kazety. Pokiaľ bol program uchovaný príkazom
   SAVE *meno* LINE *číslo*
   program sa automaticky odštartuje. Chyba
   4 Out of memory
   sa objaví, keď nie je miesto pre nový program a premenné. V tomto prípade starý program a premenné nie sú zrušené.
2. Časť pamäti
   LOAD *meno* CODE *začiatok, dĺžka*
   vykoná načítanie záznamu do pamäti. Pokiaľ je *dĺžka* v hlavičke na páske viac než je špecifikované, hlási sa chyba.
   LOAD *meno* CODE *začiatok*

nahrá časť pamäti z pásky do počítača, začína sa ukladať na začiatočnej adrese
LOAD *meno* CODE
nahrá časť pamäti z pásky do počítača podľa toho, ako bol špecifikovaný v príkaze SAVE.
LOAD *meno* SCREEN$
nahrá časť pamäti z pásky do video pamäti.

3. Pole
LOAD *meno* DATA *písmeno* ( ) alebo
LOAD *meno* DATA *písmeno$* ( )
ruší akékoľvek pole nazvané *písmeno* alebo *písmeno*$ a vytvára nové pole z pásky. Chyba
4 Out of memory
sa objaví, pokiaľ nie je miesto pre nové pole. Staré pole potom nie je zrušené.

Príkaz MERGE

1. Program a premenné
MERGE *meno*
spojuje program z pásky s programom v počítači, pripisuje riadky programu alebo premenné do starého programu. Chyba
4 Out of memory
sa objaví, pokiaľ nie je dostatok miesta pre starý a nový program spolu.
2. Časť pamäti - nie je možné použiť
3. Pole - nie je možné použiť

1.17. Počítač meria čas

Dosť často budete chcieť, aby program nebežal po stanovenú dobu (čas na prezretie nakresleného obrázku, prečítanie textu, atď.) Vtedy použijeme príkaz **PAUSE** v tvare:

PAUSE *číslo*

kde *číslo* je celé číslo od 0 do 65535.

Týmto príkazom sa zastaví vykonávanie programu na čas, ktorý je určený *číslom*. Jednotkou parametra *číslo* je 1/50 sekundy, teda čas zastavenia programu v sekundách bude *číslo*/50.
Číslo = 0 znamená zastavenie vykonávania programu až do stlačenia nejakého klávesu.
Ukončenie príkazu PAUSE je možné stlačením nejakého klávesu.
Príkaz PAUSE využijeme v programe pre nakreslenie chodu hodín:

```
10 REM prvy urobime cifernik
20 FOR n=1 TO 12
30 PRINT AT 10-10*COS (n/6*PI), 16+10*SIN (n/6*PI);n
40 NEXT n
50 REM teraz nastartujeme hodiny
60 FOR t=0 TO 200000: REM t je cas v sekundach
70 LET a=t/30*PI: REM a je uhol sekundovej rucicky v radianoch
80 LET sx=72*SINa: LET sy=72*COS a
200 PLOT 128,88: DRAW OVER 1;sx,sy: REM kresli sekundovu rucicku
210 PAUSE 42
220 PLOT 128,88: DRAW OVER 1;sx,sy:REM maze sekundovu rucicku
400 NEXT t
```

Hodiny pôjdu asi 55,5 hodín, čo je dané riadkom 60, ale môžete to ľahko zmeniť. Všimnite si, ako je čas na riadku 210 zadaný. Asi ste čakali PAUSE 50, ale výpočet tiež zaberie nejaký čas. Údaj v PAUSE budete musieť zrejme opraviť podľa chyby, ktorú hodiny vykazujú, ak sa porovnávajú s reálnym časom (hodiny idú s presnosťou asi 2 %).
Ak chcete presnejšie hodiny, použite nasledujúci postup, ktorý používa obsah určitej časti pamäti, ktorá sa nastavuje riadiacim programom Vášho počítača v závislosti od času. Uchované dáta sa vyberajú pomocou funkcie PEEK. Nasledujúci výraz:

```
t=(65536*PEEK 23674+256*PEEK 23673+PEEK 23672)/50
```

je počet sekúnd od zapnutia počítača (po 3 dni a 21 hodín, potom sa mení na 0 a počíta ďalej).
Upravený program pre hodiny bude potom:

```
10 REM najskor urobime ciselnik
20 FOR n=1 TO 12
30 PRINT AT 10-10*COS (n/6*PI),16+10*SIN (n/6*PI);n
40 NEXT n
50 DEF FN t( )=INT ((65536*PEEK 23674+256*PEEK 23673+PEEK
                    23672)/50) :REM pocet sekund od zaciatku
100 REM teraz nastartujeme hodiny
110 LET t1=FN t( )
120 LET a=t1/30*PI : REM a je uhol sekundovej rucicky v radianoch
130 LET sx=72*SIN a: LET sy=72*COS a
140 PLOT 128,88: DRAW OVER 1;sx,sy: REM nakresli rucicku
200 LET t=FN t( )
210 IF t=t1 THEN GO TO 200: REM pockaj, az je cas na dalsiu rucicku
220 PLOT 128,88: DRAW OVER 1; sx,sy: REM maze staru rucicku
230 LET t1=t: GO TO 120
```

Vnútorné hodiny, používajúce túto metódu, majú presnosť asi 1%. Tieto hodiny sa však dočasne zastavia, ak použijete príkaz BEEP, alebo pri operáciach s magnetofónom. Čísla na adresách 23674, 23673 a 23672 sú uložené v pamäti počítača a sú použité pre počítanie v 50-nách sekundy. Každé je v rozmedzí od 0 do 255, po 255 sa vracia na 0. Najčastejšie sa mení číslo na adrese 23672 (najnižší rád). Vždy za 1/50 sekundy sa zväčší o 1. Keď je 255, v nasledujúcom kroku sa zmení na 0 a v rovnakom čase sa zvýši číslo na adrese 23673 o 1. Keď má toto hodnotu 255 a mení sa na 0, zároveň sa číslo na adrese 23674 zvýši o 1.
Zväčšenie hodnôt sa vykonáva automaticky prerušením tak, že vykonávanie programu v jazyku BASIC sa zastaví, vykoná sa podprogram v strojovom kóde pre zmenu systémového času a vykonávanie programu v jazyku BASIC pokračuje. Toto prerušenie je časovo také malé, že ho nezaregistrujete.

Teraz starostlivo uvažujte. Predpokladajme naše tri čísla: 0 (pre adresu 23674), 255 (pre adresu 23673) a 255 (pre adresu 23672).
To znamená, že čas v sekundách je:

```
(65536*0 + 256*255 + 255)/50=1310.7
```

Ale je tu nebezpečenstvo. Práve sa ukončilo vyhodnotenie funkcie PEEK 23674 (hodnota 0) a nastalo nasledujúce prerušenie. Hodnoty v pamäti sa zmenia na 1, 0, 0 a až potom nasleduje vyhodnotenie funkcií PEEK 23673 a PEEK 23672 (teraz majú hodnoty 0, 0). Preto platí:

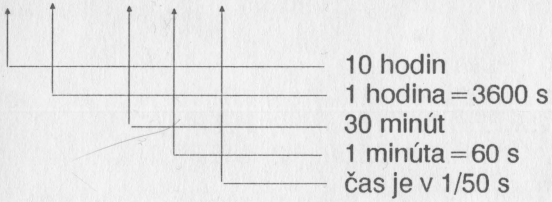
t = (65536*0 + 256*0 + 0)/50 = 0

čo je pochopiteľne chybne. Tomuto problému sa vyhneme tak, že budeme počítať čas 2 krát a vezmeme väčšiu hodnotu. Tu je úprava pre uvedený postup. Definujte:

```
10 DEF FN m(x,y) = (x + y + ABS(x-y))/2:REM vacsie z x a y
12 DEF FN u( ) = (65536*PEEK 23674 + 256*PEEK 23673 + PEEK
                  23672)/50:REM cas moze byt chybny
14 DEF FN t( ) = FN m(FNu( ),FN u( ) ): REM spravny cas
```

Hodnoty v pamäti môžete však aj nastaviť, napr. nastavíme čas na 10 hodin 30 minút dopoludnia:

(10*3600 + 30*60)*50 = 1890000 = 65536*28 + 256*214 + 208



Na nastavenie hodnôt na 28, 214, 208 použijete:

POKE 23674,28: POKE 23673,214: POKE 23672,208

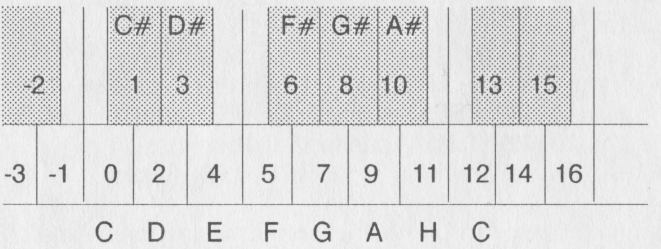
1.18. Počítač ako hudobník

Váš počítač má zabudovaný reproduktor, ktorý môžete ovládať príkazom **BEEP** v tvare:

BEEP *dĺžka, tón*

kde *dĺžka* a *tón* su aritmetické výrazy. Dĺžka trvania sa zadáva v sekundách, tón sa zadáva v poltónoch nad stredným C (tón 0). Pre nižšie noty sa používajú záporné čísla.

Nasledujúci diagram ukazuje hodnoty všetkých nôt jednej oktávy na klávesnici klavíra:



Na získanie vyšších alebo nižších nôt musíme pridať alebo odčítať 12 pre každú oktávu hore i dole. Pokiaľ máte pri programovaní vedľa seba klavír, bude Vám stačiť k práci s ladením tento diagram. Tu je jedna ukážka programu:

```
10 PRINT "Gustav"
20 BEEP 1,0: BEEP 1,2: BEEP .5,3: BEEP .5,2: BEEP 1,0
30 BEEP 1,0: BEEP 1,2: BEEP .5,3: BEEP .5,2: BEEP 1,0
40 BEEP 1,3: BEEP 1,5: BEEP 2,7
50 BEEP 1,3: BEEP 1,5: BEEP 2,7
60 BEEP .75,7:BEEP .25,8: BEEP .5,5: BEEP .5,3: BEEP .5, 2: BEEP 1,0
70 BEEP .75,7:BEEP .25,8:BEEP .5,7: BEEP .5,5: BEEP .5, 3: BEEP .5,2:
   BEEP 1,0
80 BEEP 1,0: BEEP 1,-5: BEEP 2,0
90 BEEP 1,0: BEEP 1,-5: BEEP 2,0
```

Keď odštartujete program, mali by ste počuť pohrebný pochod z Mahlerovej prvej symfónie, časť, kde škriatkovia pochovávajú muža z americkej kavalérie. Predpokladajte, že Vaša melódia je písaná v kľúči C mol, podobne, ako Mahlerova.
Pokiaľ chcete zmeniť stupnicu, je najlepšie vložiť premennú *stup* pred každou hodnotou výšky tónu:

```
20 BEEP 1, stup+0: BEEP 1,stup+2: BEEP .5,stup+3: BEEP.5,stup+2:
   BEEP 1,stup+0
```

Pred spustením programu musíte dať vhodnú hodnotu pre *stup*, tzn. 0 pre C mol, 12 pre C mol zdvihnuté o jednu oktávu, atď. Môžete prinútiť počítač, aby znel ako iný nástroj, keď *stup* je vo forme zlomku. Takisto môžete upravovať dĺžku nôt. Pretože tu išlo o pomalú skladbu, volili sme 1 sekundu pre štvrťovú notu, a založili na tom pravidlo, že napr. osminová nota znie 1/2 sekundy atď. Oveľa pružnejšie je použiť premennú *cas* tak, aby mal dĺžku štvrťovej noty a potom špecifikovať jej trvanie. Tak riadok 20 by teraz vyzeral:

```
20 BEEP cas, stup+0: BEEP cas,stup+2: BEEP cas/2,stup+3: BEEP
   cas/2,stup+2: BEEP cas,stup+0
```

Nastavte *cas* na rôzne hodnoty a môžete meniť ľahko rýchlosť hrania. Uvedomte si, že v počítači je len jeden reproduktor a tak môžete hrať v jednom okamžiku len jednu notu. Skúste programovať svoje melódie, začnite najprv celkom jednoduchou. Pokiaľ nemáte klavír a nepoznáte noty, obstarajte si jednoduchý nástroj ako je píšťala a tvorte melódie. Počítač vytvorí každý tón ako tento nástroj. Teraz zadajte:

```
FOR n=0 TO 1000: BEEP .5,n: NEXT n
```

Počítač bude hrať postupne zvyšujúce sa tóny a potom sa zastaví so správou:

```
B integer out of range
```

lebo *n* je mimo rozsah. Skúste to isté pre klesajúce tóny. Najnižšie tóny budú znieť ako klepanie, i keď sa nízke tóny tvoria rovnako, ako vyššie tóny. Len uprostred povoleného rozsahu (od -60 do 69) sú tóny vhodné pre hudbu. Nízke tóny počujeme ako klepanie, vyššie sú slabé. Teraz vložte tento riadok:

```
10 BEEP .5,0: BEEP .5,2: BEEP.5,4: BEEP .5,5: BEEP .5,7: BEEP .5,9:
   BEEP . 5,11: BEEP .5,12: STOP
```

Príklad hrá stupnicu C dur, ktorá používa všetky biele klávesy na klavíri od stredného C po nasledujúce C. Stupnica neznie rovnako ako na klavíri, nazýva sa temperovaná, pretože interval poltónu je rovnaký po celý čas. Huslista by hral stupnicu odlišne, tak, že by tóny zneli pre ucho príjemnejšie.

Môže to urobiť tým, že jemne sunie prsty po strune, čo klavirista nemôže. Základná stupnica, ktorú hraje huslista, je asi nasledujúca:

```
20 BEEP .5,0: BEEP .5,2.039: BEEP .5,3.86: BEEP .5,4.98: BEEP .5,7.02:
   BEEP .5,8.84: BEEP .5,10.88: BEEP .5,12: STOP
```

Možno, že nebudete schopní poznať rozdiel medzi stupnicami. Prvý rozdiel je v tom, že tretí tón je nižší než v prirodzene ladiacej stupnici. Pokiaľ ste špecialista, môžete v tejto tónine programovať. Nevýhodou je, že keď to pracuje dobre v tónine C, tak v niektorých stupniciach to možno urobiť len ťažko. V temperovanej stupnici to však môžete použiť vždy.
Niektoré hudby, obzvlášť indiánska, používajú intervaly menšie, než je časť tónu. Toto môžete programovať s použitím BEEP bez problémov, napr. štvrťtón nad hodnotou C má hodnotu .5.

Pokiaľ chcete počuť kvalitnejšie zvuky, musíte použiť iný než vstavaný reproduktor (signál je prítomný v zásuvke pre magnetofón).
Ďalšou možnosťou je použiť **zvukový interface MELODIK** (vyrába Didaktik Skalica), ktorý Vám zaručí perfektný zvukový doprovod veľkej časti hier. S výsledným zvukom budete iste spokojní. Tóny môžete samozrejme nahrávať na magnetofón a potom prehrávať.

1.19. Použitie tlačiarne

K pripojeniu tlačiarne slúži vstavaný paralelný interface. Programy na ovládanie tlačiarne s rozhraním CENTRONICS sú uložené na úvodnej diskete.

Pre pripojenie tlačiarne k počítaču potrebujete taktiež prepojovací kábel. Tento môže byť pre každú tlačiareň iný a preto nie je súčasťou počítača. Spôsob objednávky kábla je uvedený v príručke "Začíname". Zatiaľ uvedieme iba najdôležitejšie informácie:

- paralelný interface je osadený obvodom 8255
- po resete počítača je nastavený mód 0, porty A, B a C sú nastavené smerom dovnútra
- pri nastavení portu A dovnútra je čítaný Kempston joystick, pri nastavení portu A smerom von môžu byť dáta čítané vonkajším zariadením

Podrobnejší popis paralelného interfacea je uvedený v tretej časti príručky, venovanej technickému popisu počítača.

Ak máte tlačiareň pripojenú, môžete použiť príkazy pre tlačiareň.
Prvé dva príkazy **LPRINT** a **LLIST** sú obdobné ako PRINT a LIST, ibaže namiesto obrazovky používajú tlačiareň. Ich použitie ukazuje nasledujúci program:

```
10 LPRINT "tento program"
20 LLIST
30 LPRINT "tlaci postupnost znakov"
40 FOR n=32 TO 127
50 LPRINT CHR n;
60 NEXT a
```

Príkaz na riadku 10 vypíše na tlačiarni zadaný text.
Príkaz na riadku 20 vypíše listing programu, ďalej sa vypíše zadaný text (riadok 30).
Cyklus na riadkoch 40 až 60 vypíše postupne všetky znaky s kódom 32 až 127. Výsledok závisí od ovládacieho programu, ktorý používate. Zostáva posledný príkaz pre kópiu obrazovky:

COPY

ktorý je určený špeciálne pre tlačiareň ZX Printer (bola to dvojihličková tlačiareň, ktorá sa v zahraničí vyrábala k počítačom Sinclair).

Príkazy **OPEN** a **CLOSE** sú určené okrem práce s disketovou jednotkou aj pre otvorenie a zatvorenie kanálov. Ich použitie v súvislosti s kanálmi bude podrobne vysvetlené v druhej časti príručky.

1.20. Vstupno/výstupné miesta počítača

Procesor počítača vie čítať a zapisovať do pamäti pomocou príkazov PEEK a POKE. Procesor sa nezaujíma, či pamäť je typu ROM, RAM, alebo či pamäť nie je, vie len, že je 65536 adries pamätí a môže čítať z každej adresy pamäti a zapísať 1 byte na 1 adresovateľné miesto.

Podobne je k dispozícii 65536 miest, ktorým hovoríme **I/O miesta** (Input/Output = vstupno/výstupné). Tieto sú použité procesorom pre komunikáciu s vonkajším zariadením (klávesnica, tlačiareň,...) a môžu byť riadené v jazyku BASIC príkazom **OUT** a ich stav môže byť zisťovaný použitím funkcie **IN**.

Upozorňujeme užívateľov, že ak sú úplní začiatočníci, bolo by vhodné pre ďalšie pochopenie preštudovať kapitolu "Číselné sústavy" z druhej časti príručky.

Funkcia **IN** je podobná funkcii **PEEK** a má tvar:

IN *adresa*

kde *adresa* je v rozsahu 0 až 65535. Výsledok je hodnota bytu načítaná z vstupno/výstupného miesta s touto adresou.
Príkaz **OUT** je príkaz podobný POKE a má tvar:

OUT *adresa,hodnota*

OUT zapíše danú *hodnotu* na miesto určené *adresou*. Ako je *adresa* chápaná, záleží na technickom usporiadaní počítača.

Adresa je určená pomocou 16 bitov, ktoré označujeme ako:

A15, A14, A13, A12, ... A2, A1, A0

Vyslaná alebo prečítaná hodnota má 8 bitov a tie sa často označujú ako D7, D6, ... D1, D0. V práci so vstupmi/výstupmi je použitý rad vstupných adries, pomocou ktorých sa číta klávesnica a tiež signál z magnetofónu.

Klávesnica je rozdelená na 8 polovíc radov po 5 klávesoch.

IN 65278	číta polovicu rady od	CAPS SHIFT	do V
IN 65022	-"-	A	G
IN 64510	-"-	Q	T
IN 63486	-"-	1	5
IN 61438	-"-	0	6
IN 57342	-"-	P	7
IN 49150	-"-	ENTER	H
IN 32766	-"-	SPACE	B

Uvedené adresy vypočítame podľa vzorca 254 + 256*(255-2 ^ n) pre n = 0 až 7. V čítanom byte znamenajú bity D0 až D4 päť klávesov. V danej polovici radu je D0 pre vonkajší kláves, D4 pre najbližší kláves u stredu. Bit je 0, keď je kláves stlačený a 1, keď stlačený nie je.
Pre ilustráciu funkcie IN vyskúšajte nasledujúci program:

```
10 FOR n = 0 TO 7: REM polovicny rad
20 LET a = 254 + 256*(255-2 ^ n)
30 PRINT AT 0,0; IN a: GO TO 30
```

a skúšajte stláčať rôzne klávesy. Ak zistíte nejakú spojitosť, stlačte <BREAK>, potom zadajte NEXT n v priamom režime a pokračujte.
Podrobnejšie informácie o vstupno/výstupných miestach sú uvedené v tretej časti príručky.

1.21. Práca so strojovým kódom

Ak sa zoznámite s počítačom dôkladnejšie, budete chcieť určite používať podprogramy v strojovom jazyku, vlastné tabuľky znakov, dokonca budete sami vytvárať programy v strojovom jazyku.
Hlavnou výhodou programov v strojovom kóde je ich rýchlosť (sú až 100 krát rýchlejšie ako program v BASICu) a väčšinou zaberajú menej miesta ako program v BASICu.
Pri práci s nimi sa však objaví problém v umiestnení programov a tabuliek v pamäti tak, aby ich operačný systém nezmazal, lebo on nevie, či danú časť pamäti využívate (či v nej máte umiestnený Váš program v strojovom kóde). Vy takisto musíte v takom prípade vedieť, ktoré časti pamäti využíva počítač pre svoju prácu. O tom sa dozviete podrobnejšie v druhej časti príručky, v časti venovanej popisu pamäti.

Zatiaľ si povedzme, že pri programovaní v jazyku BASIC sa využíva časť pamäti od adresy 0 po adresu, ktorá sa nazýva **RAMTOP**. Po resetovaní počítača sa nastaví hodnota RAMTOP automaticky na hodnotu 65367, lebo nad touto adresou je uložená tabuľka znakov užívateľskej grafiky a táto je chránená. Dokonca príkaz NEW, ktorý maže pamäť RAM, pracuje len po RAMTOP, takže nemaže užívateľom definovanú grafiku. Aby ste si vytvorili priestor pre vlastné potreby, potrebujete nastaviť adresu RAMTOP. To umožňuje príkaz **CLEAR** v tvare:

CLEAR *adresa*

kde *adresa* určuje novú adresu RAMTOP. Ostatná činnosť príkazu CLEAR ostáva, ako už bolo uvedené. Zopakujme si to:

1. vymaže všetky premenné
2. zmaže obrazovku
3. nastaví kurzor pre PLOT do ľavého dolného rohu
4. vykoná sa RESTORE
5. vymaže zásobník GOSUB

Ak použijete CLEAR týmto spôsobom, môžete buď posunúť RAMTOP, aby bolo viac miesta pre BASIC (prepísaním užívateľských grafických symbolov) alebo posuniete RAMTOP dole, aby nebola horná časť pamäti mazaná príkazom NEW a aby nebola prepísaná pri práci v jazyku BASIC. Túto časť pamäti však môžeme meniť príkazom POKE a jej stav čítať funkciou PEEK. Ak je zadaná adresa príliš malá (napr. CLEAR 23000), ohlási sa chyba:

```
M RAMTOP no good
```

Takže vytvoriť miesto pre program v strojovom kóde si už vieme.
Nasledujúca časť je napísaná pre tých, ktorí poznajú strojový kód Z80, súbor inštrukcií, ktorému rozumie procesorový čip Z80. Pokiaľ máte záujem o podrobnejšie informácie, je možné čerpať zo špeciálnej literatúry. Programy v strojovom kóde sú písané v jazyku, ktorý sa volá assembler. Ak program nie je príliš dlhý, môžete preklad previesť sami (inštrukcie strojového kódu sú uvedené v prílohe).

Predpokladajme, že chcete použiť tento program v strojovom kóde:

```
ld bc, 99
ret
```

ktorý uloží do dvojice registrov bc hodnotu 99. Program sa preloží na 4 byty 1, 99, 0 (pre ld bc,99) a 201 (pre ret) .
Ak máte program napísaný v strojovom kóde, ďalšou úlohou je vložiť ho do počítača. Musíte sa rozhodnúť, kam ho v pamäti uložiť a za tým účelom si preň rezervovať v pamäti dostatok miesta.

Na vytvorenie miesta a uloženie programu v strojovom kóde môžete použiť nasledovný program:

```
 5 CLEAR 32499
10 LET a=32500
20 FOR i=0 TO 3
30 READ n: POKE a+i,n
40 NEXT i
50 DATA 1,99,0,201
```

Na odštartovanie programu v strojovom kóde sa používa funkcia **USR** v tvare:

USR *adresa*

kde *adresa* je štartovacia adresa začiatku programu v strojovom kóde. Hodnota funkcie je hodnota registrov b,c procesora Z80 pri návrate z tohto programu, preto je v rozsahu od 0 do 65535.
Pretože USR je funkcia, musí sa použiť v spojitosti s nejakým príkazom alebo môže byť časťou výrazov. Najčastejšie sa používajú tieto možnosti:

PRINT USR *adresa*

kedy sa vypíše hodnota tejto funkcie na obrazovku.

V našom prípade po zadaní:

PRINT USR 32500

sa vypíše na obrazovku číslo 99, čo je obsah registrov b,c.

Ďalšou možnosťou je použitie:

RANDOMIZE USR *adresa*

kedy sa hodnota funkcie uloží na adresu, na ktorú sa ukladá číslo z postupnosti náhodných čísiel.
Použiť môžete ešte:

RESTORE USR *adresa*

kedy sa výsledok nikde neukladá.

Poslednou možnosťou je tvar:

LET *premená* = USR *adresa*

kedy sa výsledok funkcie priradí *premenej*.

Návrat do jazyka BASIC je zabezpečený po regulérnom ukončení inštrukciou RET.

Funkcia USR sa používa tiež na získanie adresy, na ktorej je uložený znak užívateľsky definovanej grafiky (udg). Funkcia sa používa v tvare:

USR "*znak*"

kde znak určuje znak užívateľskej grafiky. Napr. prvý znak udg je určený znakom "a", "A" alebo znakom zadaným stlačením klávesu A v grafickom móde klávesnice. Po resetovaní je hodnota:

USR "a" = USR "A" = 65368

Definovanie znaku *udg* si ukážeme na príklade. Definujeme si prvý znak udg ako pole šachovnice rozmeru 8x8 bodov. Šachovnica bude daná hodnotami BIN 10101010 a BIN 01010101. Program na definovanie znaku bude:

```
1000 FOR n=0 TO 6 STEP 2
1010 POKE USR "a"+n,BIN 01010101:POKE USR "a"+n+1,
     BIN 10101010
1020 NEXT n
```

Ak tlačíte prvý znak udg (grafický mód a potom kláves <A>) červenou farbou na žltom podklade, je výsledná farba oranžová. Zmenou kombinácie farieb dostanete ďalšie "nové" farby pre farbenie väčších plôch obrazovky.

ORANŽ BARVA

ČASŤ DRUHÁ : OPERAČNÝ SYSTÉM

V tejto časti sú uvedené

- základy práce s číselnými sústavami
- mapa pamäti počítača
- popis znakového generátora a UDG grafiky
- spôsob využitia programov z pamäti ROM
- práca s V/V kanálmi
- práca s V/V miestami

2. Operačný systém

Pre tých, ktorí už zvládli základnú komunikáciu so svojim počítačom a chceli by sa o ňom dozvedieť viac, je určená táto druhá časť príručky.

Popíšeme v nej základy práce s číselnými sústavami, rozdelenie operačnej pamäti počítača, spôsob komunikácie cez V/V kanály a tiež spôsob volania niektorých podprogramov v strojovom kóde z pamäti ROM.

2.1. Číselné sústavy

Iste ste už niekde počuli, že počítač počíta v **dvojkovej sústave**. Čo to znamená? Počítač je zariadenie, ktoré vďaka svojej konštrukcii z elektronických obvodov rozoznáva len dva **základné logické stavy**, a to stav ÁNO - označme si ho logickou jednotkou, a stav NIE - ten si označíme logickou nulou. Iné stavy v počítači nastať nemôžu.

Všetka práca počítača potom prebieha len v týchto dvoch stavoch a ich kombináciach. Pretože tieto stavy sú dva, hovoríme, že počítač počíta v dvojkovej sústave a všetky príkazy, čísla a programy musia byť do tejto sústavy prevedené. O prevod sa však nestaráte Vy, ale rôzne podprogramy v operačnom systéme počítača.

Naša sústava, ktorú bežne používame, sa volá **desiatková** preto, že v nej môže nastať desať rôznych základných stavov. Tieto stavy sú 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Všetky ostatné stavy sú potom vlastne odvodené od týchto desiatich základných. Teda napr. stav 18 je kombináciou stavov 1 a 8. Ale kombináciou stavov 1 a 8 je tiež stav 81. Záleží tu teda i na pozícii základných stavov. Taká číselná sústava sa potom volá **pozičná číselná sústava**.

A na rade je otázka: "Koľko číselných sústav vlastne môže existovať ?". Existovať ich môže nekonečne mnoho a tak môžeme mať napríklad sústavu, v ktorej môže nastať 60 rôznych základných stavov (tú používali starí Mayovia).

Ak zhrnieme naše doterajšie poznatky, tak pod **číselnou sústavou so základom N** budeme rozumieť pozičnú číselnú sústavu, v ktorej rozoznávame N základných stavov. Stavom budeme hovoriť číslice, ich kombinácii potom čísla a v čísle záleží na poradí číslic.

Ďalej sa budeme zaoberať len desiatkovou, dvojkovou a šestnástkovou sústavou.

2.1.1. Desiatková sústava

Desiatková sústava je pre nás najprirodzenejšia, pretože v nej už od malička počítame. Na tejto sústave si vysvetlíme spôsob tvorby čísiel z číslic a počítanie s nimi. Tieto poznatky potom skúsime využiť pri tvorbe čísiel v dvojkovej a šestnástkovej sústave pri počítaní v týchto sústavách.

Na identifikáciu čísiel v desiatkovej sústave sa používa znak **d** (dekadické vyjadrenie) na konci čísla.

Uvažujme číslice 1,5,7 a 8. Tieto číslice napíšeme za sebou, a to tak, že 1 bude prvá zľava a 8 posledná zľava. Dostaneme číslo 1578d. Toto číslo však môžeme napísať tiež takto:

$1 * 1000 + 5 * 100 + 7 * 10 + 8 * 1 = 1578d.$

Vidíme, že náš zápis 1578d je vlastne skráteným zápisom tohto čísla. Skrátený zápis si môžeme dovoliť z jedného jediného dôvodu: vieme, že číslica na rôznom mieste (pozícii) môže vyjadrovať rôznu hodnotu. Číslica, umiestnená na prvom mieste sprava vyjadruje jednotky,na druhom desiatky, na treťom stovky atď.

Z matematiky vieme, že desať umocnené na nultú je jedna, zápis je nasledujúci:

$10^0 = 1$. Podobne $10^1 = 10$, $10^2 = 100$, $10^3 = 1000$ atď.

Naše číslo teda môžeme zapísať takto:

$1 * 10^3 + 5 * 10^2 + 7 * 10^1 + 8 * 10^0 = 1578d.$

Tento zápis je najobecnejší a môžeme z neho odvodiť pravidlá zápisu pre ostatné číselné sústavy.

Číslica pred znamienkom krát môže nadobúdať hodnoty 0 až 9, teda hodnoty všetkých číslic desiatkovej sústavy. Číslo 10 sa nazýva základ sústavy a sústava je po ňom pomenovaná desiatková sústava. Základ sústavy je umocňovaný pozíciou číslice v zápise čísla. Prvá číslica sprava však nemá pozíciu jedna, ale nula. Druhá číslica sprava má pozíciu jedna atď.

Posledný príklad:

Číslo 1230569d môžeme podľa vyššie uvedených pravidiel rozpísať takto:

$1*10^6+2*10^5+3*10^4+0*10^3+5*10^2+6*10^1+9*10^0 = 1230569d.$

2.1.2. Dvojková sústava

Základom dvojkovej sústavy je číslo 2. Číslice tejto sústavy teda môžu nadobúdať hodnoty 0 a 1. Tvorba čísiel v tejto sústave je analogická ako v sústave desiatkovej. Na identifikáciu čísiel v dvojkovej sústave budeme používať na konci čísla znak **b** (binárne vyjadrenie).

Skúsme si zapísať číslo v dvojkovej sústave: 11001100b. Toto číslo teraz rozpíšeme:

$1*2^7 + 1*2^6 + 0*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 0*2^0 =$
$1*128 + 1*64 + 0*32 + 0*16 + 1*8 + 1*4 + 0*2 + 0*1 =$
$128 + 64 + 8 + 4 = 204d.$

Podobným spôsobom sa dá vyčísliť ľubovoľné číslo v dvojkovej sústave a previesť do sústavy desiatkovej.

Opačný prevod (z desiatkovej sústavy do dvojkovej) je trocha zložitejší. Skúsime previesť číslo 204d z desiatkovej do dvojkovej sústavy. Prevod sa vykonáva delením prevádzaného čísla číslom 2, teda základom číselnej sústavy do ktorej prevádzame. Delenie sa vykonáva tak dlho, dokiaľ nie je výsledok delenia nula:

204 : 2 =	102	zvyšok: 0 ↑
102 : 2 =	51	zvyšok: 0 |
51 : 2 =	25	zvyšok: 1 |
25 : 2 =	12	zvyšok: 1 |
12 : 2 =	6	zvyšok: 0 |
6 : 2 =	3	zvyšok: 0 |
3 : 2 =	1	zvyšok: 1 |
1 : 2 =	0	zvyšok: 1 |

Teraz si napíšeme zvyšky po delení (teda 0 a 1) vedľa seba. Začnite zdola v smere šípky, spodná číslica bude zapísaná prvá zľava. Za ňou vpravo napíšte druhú číslicu zdola atď. Dostanete číslo 11001100b, teda naše pôvodné číslo.

Ostávajú nám ešte desatinné čísla. Tie sa prevádzajú obdobne, iba namiesto delenia dvoma sa číslo násobí dvoma. Pokiaľ výsledok prekročí číslo 1, do dvojkového zápisu sa zapíše 1 a od výsledku sa jednotka odpočíta. Pokiaľ výsledok neprekročí číslo 0, do dvojkového zápisu sa zapíše 0 a pokračuje sa v násobení. Výsledok sa zapíše v tom poradí, ako bol získaný. Teda za desatinnou bodkou sa zapíše prvý výsledok po náso-

bení, potom druhý atď. Uvedieme si príklad. Chceme previesť číslo 4/5d = 0.8d. Postupujeme takto:

výpočet	celá časť	desatinná časť
0.8 * 2 = 1.6	1	0.6
0.6 * 2 = 1.2	1	0.2
0.2 * 2 = 0.4	0	0.4
0.4 * 2 = 0.8	0	0.8
0.8 * 2 = 1.6	1	0.6
0.6 * 2 = 1.2	1	0.2
0.2 * 2 = 0.4	0	0.4
0.4 * 2 = 0.8	0	0.8
0.8 * 2 = 1.6	1	0.6
0.6 * 2 = 1.2	1	0.2
0.2 * 2 = 0.4	0	0.4
0.4 * 2 = 0.8	0	0.8
0.8 * 2 = 1.6	1	0.6
0.6 * 2 = 1.2	1	0.2
0.2 * 2 = 0.4	0	0.4
0.4 * 2 = 0.8	0	0.8
0.8 * 2 = 1.6	1	0.6
0.6 * 2 = 1.2	1	0.2
0.2 * 2 = 0.4	0	0.4
0.4 * 2 = 0.8	0	0.8
0.8 * 2 = 1.6	1	0.6
0.6 * 2 = 1.2	1	0.2
0.2 * 2 = 0.4	0	0.4
0.4 * 2 = 0.8	0	0.8
0.8 * 2 = 1.6	1	0.6
0.6 * 2 = 1.2	1	0.2
0.2 * 2 = 0.4	0	0.4
0.4 * 2 = 0.8	0	0.8
0.8 * 2 = 1.6	1	0.6
0.6 * 2 = 1.2	1	0.2
0.2 * 2 = 0.4	0	0.4
0.4 * 2 = 0.8	0	0.8

Dvojkový zápis 4/5d je 0.11001100110011001100110011001100b s presnosťou na 32 miest.

V ďalšom texte budeme nazývať jednotlivé číslice v dvojkovom zápise čísla **bity**. Každý bit čísla môže nadobúdať hodnoty nula alebo jedna. Osem za sebou idúcich bitov budeme nazývať **byte**. Byte môže nadobúdať hodnoty 00000000b = 0d až 11111111b = 255d. Každý bit v byte má svoje poradové číslo. Prvý bit sprava má poradové číslo 0 (nultý bit, najnižší bit, bit s najmenšiou váhou, LSB - Least Significant Bit), ďalší bit má poradové číslo 1 (prvý bit), až siedmy bit sprava má poradové číslo 7 (siedmy bit, najvyšší bit, bit s najväčšou váhou, MSB - Most Significant Bit).

Dva byty môžu vyjadriť číslo 0 až 65535 (256*256 = 64 kB), tri byty číslo 0 až 16777216 (256*256*256 = 16 MB) atď.

V prípade, že je číslo vyjadrené dvoma bytmi (tiež sa hovorí uložené na dvoch bytoch), hovoríme o "**nižšom byte**" a "**vyššom byte**" čísla. Hodnotu čísla potom dostaneme spočítaním hodnoty nižšieho bytu s 256-násobkom hodnoty vyššieho bytu. V pamäti počítača sú tieto dva byty uložené nasledovne, poradie je dané zvyklosťami firmy **Intel** (u druhých firem môže byť poradie opačné):

nižší byte	vyšší byte
adresa X	adresa (X + 1)
hodnota = nižší byte + 256*vyšší byte	

2.1.3. Šestnástková sústava

Základom tejto sústavy je číslo 16. Číslice tejto sústavy môžu nadobúdať hodnoty 0 až 15. Číslice väčšie ako deväť sa však nedajú zapísať jedným znakom a preto bola vytvorená dohoda, že sa budú označovať prvými šiestimi písmenami abecedy.

Čísla v šestnástkovej sústave budeme na konci označovať znakom **h** (hexadecimálny). Číslice šestnástkovej sústavy sú nasledovné:

desiatkovo	šestnástkovo
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Skúsme si previesť číslo 10AFh do desiatkovej sústavy. Číslo sa dá rozpísať takto:

$1*16^3 + 0*16^2 + 10*16^1 + 15*16^0 =$
$1*4096 + 0*256 + 10*16 + 15*1 =$
$4096 + 160 + 15 = 4271d.$

Podobným spôsobom sa dá vyčísliť ľubovoľné číslo v šestnástkovej sústave.

Opačný prevod je rovnaký ako u dvojkovej sústavy. Opäť postupným delením čísla v desiatkovej sústave dostaneme zvyšky po delení, ktoré reprezentujú zápis čísla v šestnástkovej sústave.

Skúsime previesť číslo 4271d z desiatkovej do šestnástkovej sústavy:

```
4271 : 16 = 266     zvyšok: 15 = F  ↑
 266 : 15 =  16     zvyšok: 10 = A  |
  16 : 16 =   1     zvyšok:  0 = 0  |
   1 : 16 =   0     zvyšok:  1 = 1  |
```

Opäť si zapíšeme zvyšky po delení zdola a dostaneme číslo v šestnástkovej sústave, zhodné s našim pôvodným, teda 10AFh.

Všetky aritmetické operácie (sčítanie, násobenie, odpočítanie atď.) sa v týchto sústavách riadia rovnakými zákonmi ako v desiatkovej sústave.

V ďalšom texte budeme chápať všetky čísla bez označenia d ako čísla dekadické.

2.2. Vnútorná forma čísla

Každé číslo, s ktorým počítač pracuje, musí byť v jeho pamäti zapísané v nejakej štandardnej forme. BASIC počítača ukladá každé číslo na päť bytov, pričom rozlišuje tzv. **malé celé čísla** z intervalu −65535 až +65535 a **ostatné čísla**. Malé celé čísla sú uložené takto:

00h	00h pre kladné čísla FFh pre záporné čísla	nižší byte čísla	vyšší byte čísla	00h

Číslo −257 je uložené nasledovne : 00FF010100b
Číslo +257 obdobne : 0000010100b

Ostatné čísla sú uložené v **semilogaritmickom tvare**. Ako tento tvar vyzerá? Číslo je prevedené na tvar: ± **m** * 2^{e}, kde ± je znamienko, **m** je **mantisa**, ležiaca v rozsahu 1/2 až 1 (nesmie byť 1 a má rozsah 4 byty) a **e** je **exponent** v rozsahu −127 až 128. Uloženie je nasledovné:

byte	1. byte	2. byte	3. byte	4. byte
exponent	mantisa			

Najvyšší bit exponenta určuje, či je exponent záporný (bit je nastavený na 0), alebo kladný (bit je nastavený na 1). Podobne najvyšší bit v prvom byte mantisy určuje, či je číslo záporné (bit je nastavený na 1), alebo kladné (bit je nastavený na 0).

Uvedieme si príklad:

Chceme zapísať číslo 1/10. Najskôr ho musíme previesť do tvaru $\pm m * 2^e$. 1/10 je menšia než 1/2 a preto ju rozšírime 4/4.
Dostaneme $1/10*4/4 = 4/40 = 4/5*2^{-3}$.

Platí: 4/5 = 0.11001100110011001100110011001100b; −3 = 11111101b

Ako sme k tomuto číslu prišli ? Číslo +3 má v dvojkovej sústave zápis 00000011b. Zmeníme všetky 0 na 1 a všetky 1 na 0. Dostaneme binárne číslo 11111100b. K tomuto číslu pripočítame číslo 1 (teda 00000001b) a dostaneme:

```
  11111100b
+ 00000001b
  11111101b
```

A to je práve binárny zápis čísla −3. Tomuto spôsobu vytvorenia čísla −3 sa hovorí, že sme vytvorili dvojkový doplnok čísla +3. Malo by teda platiť: −3 + 3 = 0.
Skúsme to:

```
  00000011b
+ 11111101b
  100000000b
```

Vo výsledku sa objavil ešte deviaty bit, ktorý označuje prenos do vyššieho rádu a hovorí, že prišlo k pretečeniu. Pri práci s osembitovými číslami tento bit zanedbávame.
Binárny zápis 11111101b predstavuje číslo −3d. Pretože je ale mantisa kladná, zmeníme prvý bit mantisy na 0. Podobne zmeníme prvý bit exponenta na 0 (exponent je záporný). Dostávame binárne číslo: 01111101 01001100 11001100 11001100 11001100b, čo je vyjadrenie 1/10 v semilogaritmickom tvare. Podobne môžeme postupovať pri prevode akéhokoľvek čísla do semilogaritmického tvaru.

2.3. Mapa pamäti

Jedna z najdôležitejších častí, ktoré počítač obsahuje, je jeho **pamäť**. V nej je uložený interpreter jazyka BASIC, operačný systém MDOS, ukladajú sa do nej informácie o zobrazení na obrazovke, o programe, vlastný program a ďalšie doplňujúce informácie.

Existujú dva základné typy pamätí. Jedným je pamäť typu **ROM** (Read Only Memory), teda pamäť, z ktorej sa dá len čítať. Druhým je pamäť **RWM** (Read Write Memory), teda pamäť, z ktorej sa dá čítať, a dá sa do nej i zapisovať. Tento typ pamäti je tiež označovaný ako **RAM**.

Pamäť si môžeme predstaviť ako rad buniek (bytov), nasledujúcich bezprostredne za sebou. V každom byte môže byť zapísané číslo od 0 do 255. Toto číslo je v byte zapísané v dvojkovej sústave.

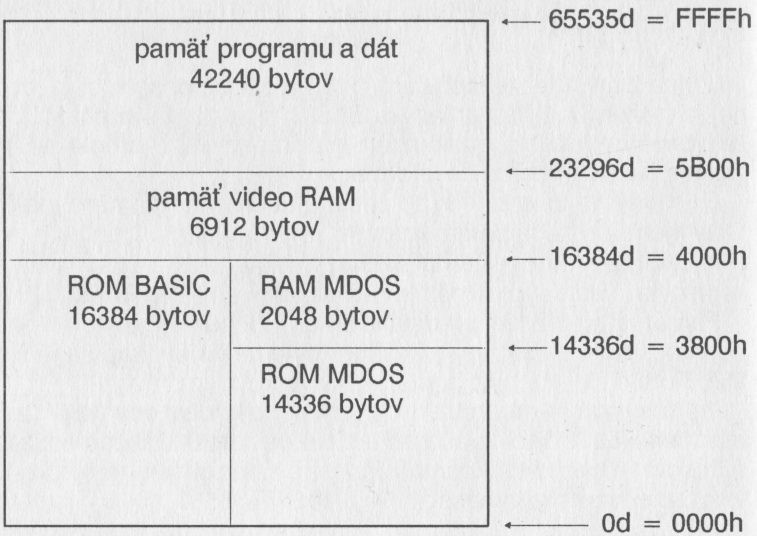
Veľkosť pamäti sa určuje v bytoch alebo **kilobytoch** (kB). Každý kilobyte má $1024 = 2^{10}$ bytov. Váš počítač má celkom 80 kB pamäti, z toho súvisle adresovateľných 64 kB pamäti. Pokiaľ by sme chceli vyjadriť veľkosť pamäti v bytoch, musíme vynásobiť $64 * 1024 = 65536 = 2^{16}$ bytov. Prvý byte pamäti má poradové číslo 0, posledný byte má poradové číslo 65535 (nemôže mať 65536, pretože medzi 0 a 65535 je práve 65536 bytov).

Celú pamäť môžeme rozdeliť na tri časti. Prvá je pamäť ROM s veľkosťou 30 kB. V nej je uložený **operačný systém** počítača, ktorý zabezpečuje interpretáciu príkazov jazyka BASIC a príkazov pre disketovú jednotku. Túto pamäť si môžete predstaviť ako dve pamäti vedľa seba. V jednej je umiestnený BASIC, v druhej MDOS. Operačný systém sa potom v závislosti na príkaze sám prepne do časti určenej pre BASIC, alebo pre MDOS.

Druhá časť pamäti s veľkosťou 48 kB je typu RAM. V tejto časti sú uložené informácie nutné pre prácu počítača. Pamäť RAM môžeme ešte rozdeliť na dve časti, a to na tzv. **video RAM** (v nej sú uložené obrazové informácie) a **pamäť programu a dát**.

Tretia časť pamäti je tieňová MDOS RAM veľkosti 2 kB, ktorú systém využíva pri práci s disketovou jednotkou.

Celá pamäť počítača má nasledovnú štruktúru:



2.3.1. Pamäť video RAM

Samotná pamäť je rozdelená na dve časti, v podstate na sebe nezávislé. Prvá časť je vyhradená pre kresbu - **pamäť kresby**, druhá pre farebné atribúty - **pamäť atribútov**.

2.3.1.1. Pamäť kresby

Pamäť kresby obsahuje každý zobraziteľný bod obrazovky a určuje, či je bod "rozsvietený" (v pamäti kresby je na mieste pre tento bod uložená jednotka), alebo "zhasnutý" (v pamäti kresby je uložená nula).

Skúste zadať POKE 16384,255. V ľavom hornom rohu obrazovky sa rozsvieti vodorovná čiarka v rozsahu ôsmich bodov (bitov). Skúste napísať do počítača nasledujúci program a zadajte RUN:

```
10 FOR i=0 TO 7
20 POKE 16384,2^i
30 PAUSE 25
40 NEXT i
```

Tento program postupne ukladá na prvý byte pamäti kresby čísla 1,2,4,8,16,32,64,128 a rozsvietený bod sa postupne presúva sprava doľava.

Pamäť pre kresbu má rozsah 6144 bytov. Toto číslo dostanete pomocou vzorca:

$$\text{pamäť kresby} = \frac{\text{počet mikroriadkov} * \text{počet bodov na riadok}}{8}$$

pamäť kresby = (192 * 256)/8 = 6144 bytov

Pamäť pre kresbu má v dôsledku použitého technického spôsobu zobrazovania trochu zvláštnu štruktúru, odlišnú od štruktúry pamäti programu a dát. Pokúsime sa ju vysvetliť najskôr na príkladoch. Napíšte tento program:

```
10 FOR i=0 TO 31
20 POKE (16384+i),255
30 NEXT i
```

Odštartujte ho pomocou RUN. Na nultom riadku obrazovky sa postupne nakreslí čiara. Týmto programom ste uložili do pamäti kresby od adresy 16384 po adresu 16415 samé jednotky (celkom 32 bytov po ôsmich bitoch, teda 256 bitov). Týchto 256 bitov predstavuje nultý mikroriadok obrazovky. Dalo by sa očakávať, že ďalší mikroriadok obrazovky začína v pamäti kresby na adrese 16416, teda ihneď za prvým. To však nie je pravda. Prvý mikroriadok obrazovky začína v pamäti kresby až na adrese o 256 väčšej, než je adresa nultého mikroriadku, teda na adrese 16640. Druhý mikroriadok obrazovky začína v pamäti na adrese o 256 väčšej, než je adresa začiatku prvého mikroriadku, teda na adrese 16896 atď.

Napíšte tento program:

```
10 FOR i=0 TO 7:REM pocet mikroriadkov=8
20 FOR j=0 TO 31:REM pocet bytov v riadku =32
30 POKE (16384+i*256+j),255:REM zaciatok mikroriadku je posunuty o
   256 bytov
40 NEXT j
50 NEXT i
```

Po vykonaní programu sa v prvých ôsmich mikroriadkoch obrazovky nakreslí čiara.

Iste si teraz položíte otázku:"Pokiaľ je prvý mikroriadok obrazovky posunutý oproti nultému o 256 bytov, tak čo je uložené na adrese za koncom nultého mikroriadku, teda na adrese 16384+32=16416 ?"

Napíšte nasledujúci program:

```
10 FOR i=0 TO 31
20 POKE (16384+i),255
30 NEXT i:REM NA OBRAZOVKE SA VYKRESLILA V NULTOM MIKRO-
   RIADKU CIARA
40 PAUSE 50
50 FOR i=32 TO 63
60 POKE (16384+i),255
70 NEXT i:REM NA OBRAZOVKE SA VYKRESLI V OSMOM MIKRO-
   RIADKU CIARA
```

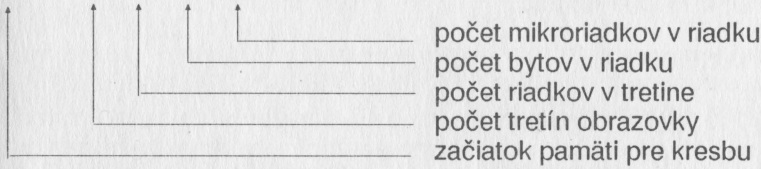
Podobne platí, že na adrese 16384 + 32 + 32 = 16448 je uložených 0 + 8 + 8 = 16 mikroriadkov atď. Napíšte do počítača tento program:

```
10 FOR i=0 TO 32*8*8-1:REM 32 BYTOV NA MIKRORIADOK 8 MIKRO-
   RIADKOV=1 RIADOK OBRAZOVKY A ZAPLNIME 8 RIADKOV OBRA-
   ZOVKY
20 POKE (16384+i),255
30 NEXT i
40 PAUSE 50
```

Uvedený program zaplní jednu tretinu obrazovky čiarami, výsledok je "začiernená" jedna tretina obrazovky. Prečo len jedna tretina? Aby toho nebolo málo, obrazovka sa ešte delí na tretiny. Prvá tretina začína na adrese 16384, druhá na adrese 18432 a tretia na adrese 20489. Posledný byte pamäti kresby je na adrese 22527.

Táto adresa je vypočítaná nasledovne:

16384 + 3 * 8 * 32 * 8 - 1 = 22527



Spôsob zápisu kresby sa môže zdať trocha zložitejší. Z dôvodu jednoduchej konštrukcie počítača bola obrazovka rozdelená na tretiny. Každá tretina má 8 riadkov, každý riadok má 8 liniek. Toto rozdelenie si všimnime pri nahrávaní úvodného obrázku k nejakej hre. Najprv sa nahrávajú jednotlivé linky prvej tretiny, potom druhej a tretej. Nakoniec sa nahrávajú atribúty, ktoré výslednú kresbu zafarbia.

Na určenie adresy bodu na obrazovke platí nasledujúca schéma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	A		B			C			D				

010 000000000000 - binárne vyjadrenie adresy 16384; 010 pri adresovaní obrazovky znamená, že sa prvé tri bity nemenia.

A - určenie tretiny počítané zhora. Pre A=4 ide o atribúty.

B - určenie čísla linky v jednom riadku (0 až 7).

C - určenie čísla riadku vo vybranej tretine (0 až7).

D - určenie čísla stĺpca (0 až 31).

Pre všetky tretiny platia rovnaké pravidlá ukladania mikroriadkov obrazovky do pamäti pre kresbu. Môžete si to vyskúšať tak, že namiesto začiatku prvej tretiny (16384) dosadíte do predchádzajúcich programov začiatky druhej (18432) a tretej tretiny (20480).

Napíšte do počítača posledný program tejto časti:

```
10 FOR i=0 TO 3*8*32*8-1
20 POKE (16384+i),255
30 NEXT i
40 PAUSE 100
```

Tento program postupne zaplní celú obrazovku (a teda i pamäť pre kresbu). Všimnite si, ako sa obrazovka zaplňuje. Najskôr sa zaplní prvá tretina, potom druhá a tretia.

Tým sme skončili popis pamäti pre kresbu a budeme sa venovať pamäti atribútov (pamäti pre farby).

2.3.1.2. Pamäť atribútov

Pamäť atribútov je druhá časť pamäti video RAM. V nej sú obsiahnuté všetky informácie o farbách na obrazovke, o blikaní a zvýšenom jase.

Pamäť atribútov začína na adrese 22528 a je dlhá 768 bytov, teda prvý voľný byte za touto oblasťou je na adrese 23296. Dĺžka vyplýva z toho, že pre celú obrazovku, ktorá má 24 riadkov a 32 stĺpcov, potrebujeme 24 * 32 = 768 bytov na atribúty a každý byte prináleží práve jednému štvorčeku obrazovky s rozmermi 8 x 8 bodov.

Skúste zadať POKE 22528,0. Štvorček v ľavom hornom rohu sa zafarbí na čierno. Zadajte:

```
FOR i=0 TO 31:POKE (22528+i),0:NEXT i
```

a odošlite riadok. Nultý riadok obrazovky sa zafarbí na čierno. Atribúty pre jeden riadok sú teda v pamäti uložené za sebou.

Ale čo ďalší riadok? Sú atribúty druhého riadku uložené bezprostredne za atribútami nultého riadku ? Napíšte program:

```
10 FOR i=0 TO 31
20 POKE (22528+i),0
30 NEXT i:REM NULTY RIADOK SA ZAFARBI NA CIERNO
40 PAUSE 50
50 FOR i=32 TO 63
60 POKE (22528+i),0
70 NEXT i:REM PRVY RIADOK SA ZAFARBI NA CIERNO
```

Z uvedeného príkladu vyplýva, že atribúty pre jednotlivé riadky sú v pamäti uložené za sebou v tom istom poradí, ako riadky na obrazovke. Napíšte ešte tento program:

```
10 FOR i=0 TO 767
20 POKE (22528+i),0
30 NEXT i
```

Po spustení sa postupne zafarbí celá obrazovka na čierno. Zresetujte počítač.

Vieme teda, ako sú atribúty pre riadky v pamäti uložené. Zatiaľ sme na ich miesto v pamäti ukladali len hodnotu nula a príslušný štvorček obrazovky sa zafarbil na čierno. Ako ale dosiahnuť iné farby, popr. blikanie a zvýšený jas? Do atribútových bytov je treba túto informáciu podľa určitých pravidiel uložiť. Pravidlá vyplývajú zo štruktúry **atribútového bytu** (ATB):

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

V počítači je každá farba vyjadrená určitým číslom podľa nasledujúcej tabuľky:

farba	kód dekadicky	kód binárne
čierna	0	000
modrá	1	001
červená	2	010
fialová	3	011
zelená	4	100
bledomodrá	5	101
žltá	6	110
biela	7	111

Z uvedenej tabuľky vyplýva, že ľubovoľná farba sa dá získať skladaním troch základných farieb, a to modrej, červenej a zelenej. Tak napr. žltá farba (kód 6) vznikne zložením červenej (kód 2) a zelenej (kód 4) farby.
Predpokladajme, že chceme mať na obrazovke modrý papier (modrá farba má hodnotu 1) so žltým atramentom (žltá farba má hodnotu 6), neblikajúci, bez zvýšeného jasu. Do pamäti atribútov musíme uložiť hodnotu: 00001110b = 14.

Pre zelený papier, čierny atrament a zvýšený jas uložíme do pamäti atribútov hodnotu: 01010000b = 96.

Obecne môžeme atribútový byte definovať nasledovne:

ATB = FLASH*128 + BRIGHT*64 + PAPER*8 + INK.

V našich dvoch predchádzajúcich príkladoch majú atribútové byty túto hodnotu:

- ATB = 0*128 + 0*64 + 1*8 + 6 = 14
- ATB = 0*128 + 1*64 + 4*8 + 0 = 96

Takto si môžete vypočítať a dosadiť do pamäti pre atribúty ľubovoľnú hodnotu a meniť tak zafarbenie obrazovky. Tento spôsob priameho dosadenia atribútov do pamäti sa používa predovšetkým v programoch v strojovom kóde, pretože je to najrýchlejšia cesta, ako zmeniť zafarbenie celej, popr. časti obrazovky.
Polohu atribútu na obrazovke možno vypočítať podľa vzťahu:

ATB	= 22528 + 32 * R + S
ATB	– vypočítaná adresa atribútu
22528	– adresa prvého atribútu na obrazovke
R	– poradové číslo riadku (0 až 23) (počítané z ľavého horného rohu)
S	– poradové číslo stĺpca (0 až 31) (počítané z ľavého horného rohu)

Po prepísaní adresy do binárneho tvaru je všetko zrejmé:

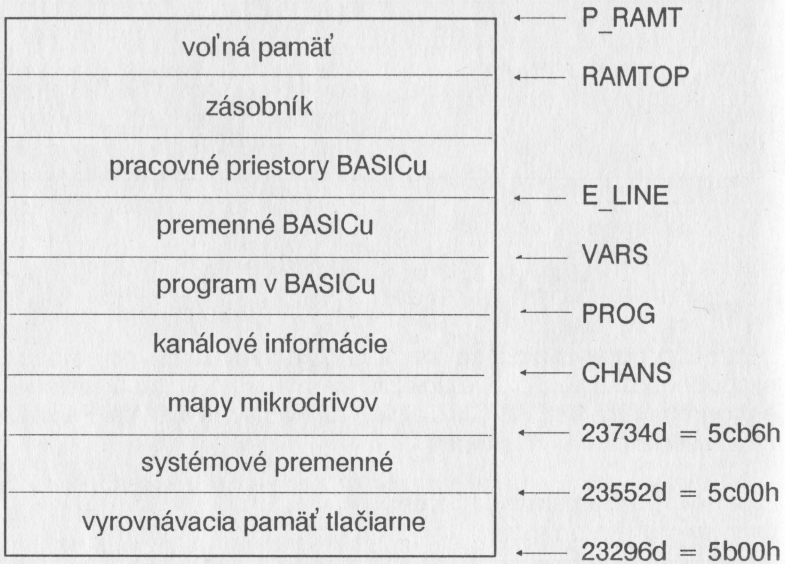
bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
hodnota	0	1	0	1	1	0	R	R	R	R	R	S	S	S	S	S
adresa	22528						riadok					stĺpec				

2.3.2. Pamäť programu a dát

Pamäť programu a dát sa rozkladá od adresy 23296 po adresu 65535. V jej spodnej časti sú uložené dôležité informácie pre prácu systému, za nimi je program v jazyku BASIC a ďalšie informácie pre systém. Pamäť môžeme rozdeliť do niekoľkých skupín:

- vyrovnávacia pamäť tlačiarne
- systémové premenné
- mapy mikrodrivov
- kanálové informácie
- program v BASICu
- premenné BASICu
- pracovné priestory BASICu
- zásobník
- užívateľská grafika
- voľná pamäť

Obsadenie pamäti môžeme znázorniť nasledovne:



Každej časti pamäti sa v tejto kapitole budeme venovať zvlášť, len užívateľskú grafiku popíšeme v kapitole samostatne. V diagrame nie je zakreslená, pretože jej umiestnenie v pamäti môže byť ľubovoľné. Názvy hraníc medzi oblasťami sú vysvetlené v prílohe. Teraz len krátko:

CHANS - adresa začiatku kanálových informácií
PROG - adresa začiatku progamu v BASICu
VARS - adresa začiatku premenných programu
E_LINE - adresa konca premenných programu v BASICu a začiatku pracovnej oblasti
RAMTOP - adresa konca pamäti vyhradenej pre BASIC
P_RAMT - adresa fyzického konca pamäti (ukazuje na poslednú adresu, z ktorej sa dá bez chyby čítať a zapisovať; väčšinou na adresu 65535)

2.3.2.1. Vyrovnávacia pamäť tlačiarne - buffer tlačiarne

Vyrovnávacia pamäť tlačiarne je uložená od adresy 23296 a dlhá je 256 bytov. Používa sa pri tlači na ZX Printer (dvojihličková tlačiareň so špeciálnym papierom, ktorá sa vyrábala v zahraničí pre počítače ZX Spectrum). V prípade tlače na ZX Printer sa do nej ukladá tlačený riadok, ale rozložený na mikroriadky (podobne ako na obrazovke). Dá sa využívať na uloženie vlastných programov v strojovom kóde.

Pri použití príkazu COPY dôjde k prepísaniu tejto oblasti, preto pozor na jeho používanie, pretože prípadný strojový kód v buffere je po zadaní príkazu COPY zrušený.

2.3.2.2. Systémové premenné

Systémové premenné sú uložené od adresy 23552 a zaberajú oblasť 182 bytov. Slúžia na riadenie práce systému. Ich rozloženie v pamäti a presný význam je uvedený v prílohe, ktorú Vám doporučujeme pozorne preštudovať.

Do tejto oblasti sa nepokúšajte zadávať hodnoty pomocou POKE, pokiaľ presne neviete, čo zadanie novej hodnoty spôsobí. Zadanie nesprávnej hodnoty obvykle skončí stlačením tlačidla RESET, popr. počítač túto činnosť vykoná za Vás.

2.3.2.3. Mapy mikrodrivov

Táto oblasť má začínať na adrese 23734 a má nenulovú dĺžku len v prípade, že je mikrodrive fyzicky pripojený a zapnutý. Viac sa s mikrodrive nebudeme zaoberať, pretože je to neperspektívne zariadenie a v zahraničí sa už dlhšiu dobu nevyrába.

2.3.2.4. Kanálové informácie

V oblasti kanálových informácií je uložená tabuľka, obsahujúca **adresy vstupných a výstupných podprogramov** (rutín) v strojovom kóde, predstavujúcich obsluhu jednotlivých **kanálov** (typov periférií). Tabuľka je po

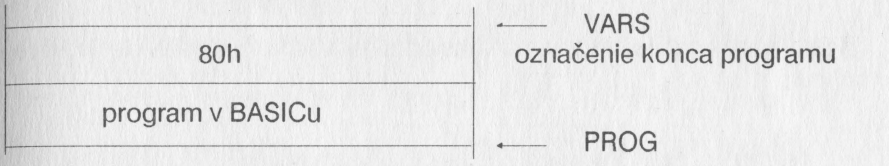
zapnutí počítača uložená od adresy 23734 a má dĺžku 20 bytov. V prípade použitia mikrodrivov sa jej začiatok úmerne posúva k vyšším adresám. Jej dĺžka je tiež premenná v závislosti od počtu kanálov, ktorých môže byť až 16. Začiatok tabuľky je uložený v systémovej premennej CHANS.
Začiatok tabuľky je od adresy uloženej v CHANS, tabuľka má nasledovný tvar:

			← PROG
kanál 3	typ tlačiareň (Printer)	P	
	adresa vstupnej rutiny	15C4h	
	adresa výstupnej rutiny	09F4h	
kanál 2	typ obrazovka (Screen)	S	
	adresa vstupnej rutiny	15C4h	
	adresa výstupnej rutiny	09F4h	
kanál 1	typ BASIC editor (woRkspace)	R	
	adresa vstupnej rutiny	15C4h	
	adresa výstupnej rutiny	0F81h	
kanál 0	typ klávesnica (Keyboard)	K	
	adresa vstupnej rutiny	10A8h	
	adresa výstupnej rutiny	09F4h	
			← CHANS

V tejto tabuľke je možné meniť adresy výstupných a vstupných rutín a tak použiť svoje vlastné rutiny napr. pre vstup z klávesnice alebo pre výstup na obrazovku a tlačiareň. Viac sa o vstupno/výstupných kanáloch dozviete v kapitole 2.7.

2.3.2.5. Program v BASICu

Adresa začiatku programu v BASICu je uložená v systémovej premennej PROG a po zapnutí počítača je nastavená na hodnotu 23755 teda bezprostredne za tabuľku kanálových informácií. So zväčšením tabuľky kanálových informácií, popr. pri použití mikrodrivov, sa začiatok programu posúva k vyšším adresám. Koniec oblasti je označený bytom s hodnotou 80h. Priestor pre program vyzerá takto:



Každý riadok programu je v pamäti uložený nasledovne:

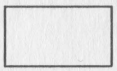
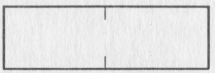
číslo riadku	dĺžka riadku	riadok programu	koncový znak
2 byty	2 byty	X bytov	1 byte = 0Dh

Číslo riadku je uložené tak ako je napísané v programe. Teda riadok s číslom 10 má v prvom byte uloženú nulu a v druhom byte uložené číslo 10. Dĺžku riadku vypočíta po odoslaní riadku operačný systém a dosadí ju na vyhradené miesto za číslo riadku. Dĺžka riadku je počítaná mimo štyri byty, určené pre číslo riadku a dĺžku riadku.

Obecne platí: dĺžka riadku (v bytoch) = X + 1

Dĺžka je už uložená v pamäti štandardne, teda najskôr nižší byte a za ním vyšší byte. Potom nasleduje vlastný riadok programu, ktorý je zakončený znakom CR (Carriage Return-návrat vozíka) s kódom 0Dh = 13. Za posledným riadkom programu (a teda i za posledným znakom CR) je ešte byte, ktorý obsahuje hodnotu 80h = 128 a označuje koniec programu v pamäti.

V ďalšom texte budeme používať značku obdĺžnika, ktorý nie je uzatvorený aspoň z jednej strany na znázornenie jedného bitu.

 = jeden byte  = dva bity

Riadok programu je v pamäti uložený v trochu inej forme, než ako ho zadáte na obrazovku. Predpokladajme, že ste zadali počítaču len jeden riadok, a to:

10 LET A=32768.

Tento riadok sa uloží do pamäti nasledovne (všetky čísla sú v desiatkovej sústave) :

0	10	15	0	241	65	61	51	50	55	54	56	14	0	0	0	128	0	13	128

10 dĺžka LET A = 3 2 7 6 8 | vnútorná forma čísla

označenie čísla

koniec riadku

koniec programu

Uloženie riadku v pamäti začína číslom riadku, za ktorým nasleduje dĺžka riadku. Všetky kľúčové slová sú zakódované do jedného bytu, tzv. **tokenu** (priradenie kódu kľúčovým slovám (tokenom) je uvedené v prílohe) a ostatné znaky v riadku sú uložené v **kóde ASCII** (viď príloha). Za každým číslom v riadku nasleduje byte s hodnotou 14, ktorý označuje, že nasleduje zápis tohto čísla vo vnútornej forme v dĺžke piatich bytov. Koniec riadku je označený znakom CR.

Z uvedeného vyplýva, že za každým číslom, použitým v programe, nasleduje byte s hodnotou 14, označujúci výskyt čísla, a za ním päť bytov , na ktorých je uložené číslo vo vnútornej forme. Pri vykonávaní programu už BASIC pracuje len s číslami vo vnútornej forme.

Tento spôsob ukladania čísiel má jednu veľkú nevýhodu, a tou je nepríjemné narastanie dĺžky programu pri používaní väčšieho množstva čísiel. Proti tomu sa však možno brániť. Stačí len v programe použiť výrazy,

ktorých uloženie v pamäti je kratšie ako päť bytov. Niekoľko výrazov Vám ponúkame, na iné iste postupom času prídete sami:

miesto	použite	ušetríte
0	NOT PI	5 bytov
1	SGN PI	5 bytov
3	INT PI	5 bytov
32	CODE " "	4 byty
255	CODE"COPY"	5 bytov

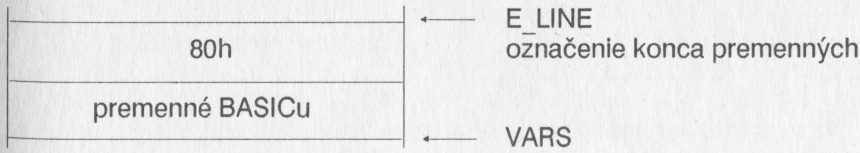
Namiesto akéhokoľvek čísla môžete použiť výraz VAL"cislo" a tým ušetríte 3 byty. Ďalšia možnosť je používanie konštánt namiesto niektorých často sa vyskytujúcich čísiel vo Vašom programe. Tak napr. do prvého riadku programu dáte príkaz:

```
LET K10 = 10
```

a všade v programe budete používať namiesto čísla 10 konštantu K10 a tým ušetríte pri každom jej použití 5 bytov. Nevýhoda použitia konštánt a výrazov je spomalenie vykonávania programu.

2.3.2.6. Premenné BASICu

Premenné, použité v programe, sú uložené ihneď za posledným riadkom programu. Ich začiatok je uložený v systémovej premennej VARS a koniec označený bytom s hodnotou 80h. Priestor pre premenné vyzerá takto:



Začiatok premenných nie je vždy na tom istom mieste. V prípade, že do programu pridáte alebo vypustíte riadok, zmení sa i dĺžka programu a začiatok premenných sa posunie.

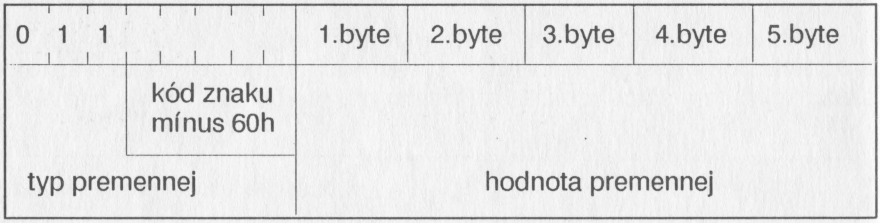
Veľkosť tejto oblasti je tiež premenná. Po príkaze RUN sa oblasť vyčistí. Pri vykonávaní programu sa každá nová premenná zaradí do oblasti premenných. Za menom premennej je potom uložená jej aktuálna hodnota a všetky aritmetické a logické operácie sú vykonávané s touto aktuálnou hodnotou.

Pri vykonávaní programu a priradení hodnoty premennej sa postupuje nasledovne:

- v programe sa zistí meno premennej
- vezme sa zo systémovej premennej VARS začiatok oblasti premenných a od tejto adresy sa hľadá meno premennej
- v prípade, že sa meno v oblasti nachádza, zistí sa z programu hodnota, ktorá má byť v premennej uložená a uloží sa do nej
- v prípade, že sa premenná v oblasti nenachádza, tak sa pre ňu vyhradí miesto, zapíše sa naň meno premennej a hodnota, ktorá sa zistí z programu.

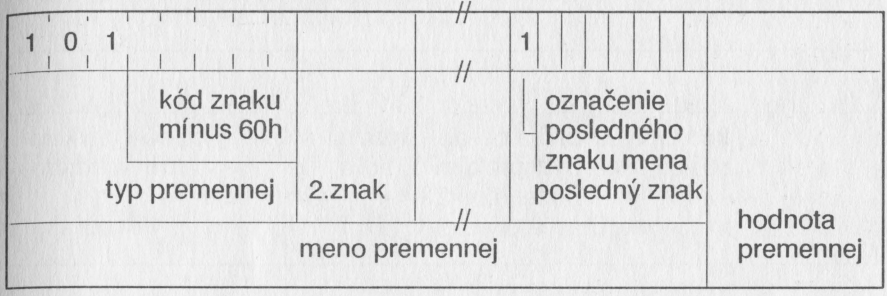
Teraz uvedieme spôsob uloženia mien a hodnôt niektorých typov premenných v oblasti premenných.

Číselná premenná (jednoznaková):

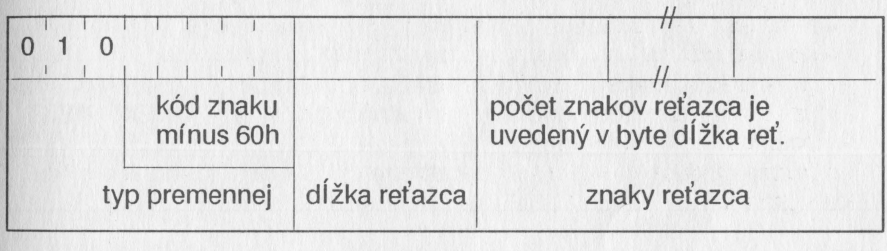


Na miesto označené "kód znaku mínus 60h" sa uloží kód znaku (písmena), ktorým je premenná označená, od ktorého sa odpočíta hodnota 60h. Napr. pre premennú, označenú písmenom "i" sa na dané miesto uloží: CODE("i") – 60h = 69h – 60h = 09h.

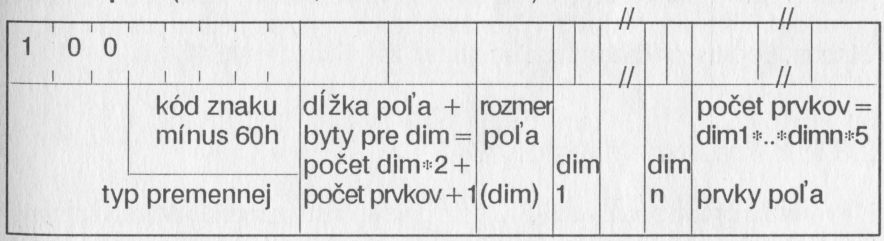
Číselná premenná (viacznaková):



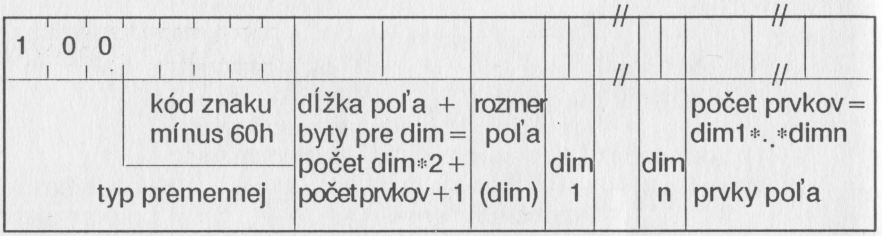
Reťazec:



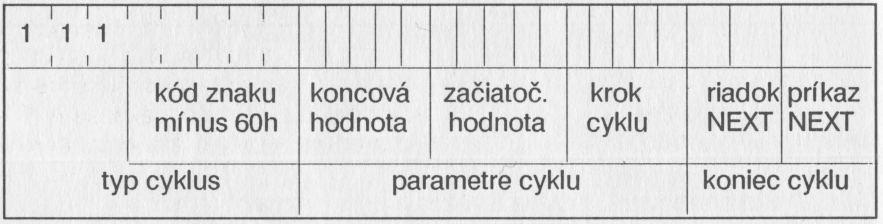
Číselné pole (n-rozmerné, kde n = 1 až 255):



Znakové pole (n-rozmerné, kde n = 1 až 255):



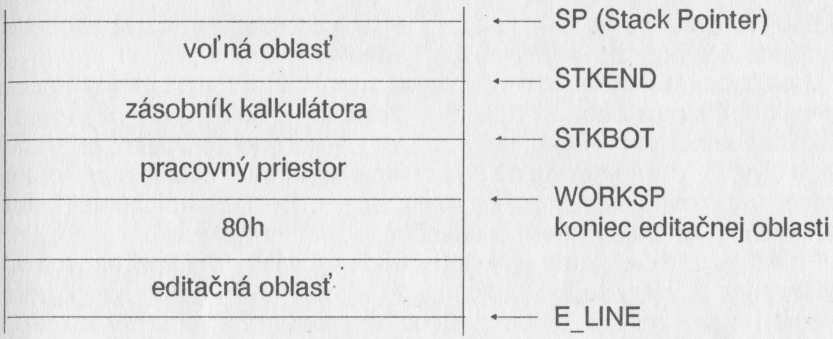
Premenná cyklu FOR .. NEXT:



Byty označené ako "riadok NEXT" a "príkaz NEXT" určujú umiestnenie príkazu NEXT v programe (teda na ktorom riadku sa nachádza a aké má poradové číslo v riadku). Pri vykonávaní programu sa potom pokračuje (po vykonaní cyklu) príkazom nasledujúcim za príkazom NEXT.

2.3.2.7. Pracovné priestory BASICu

Pre svoju prácu používa BASIC ešte ďalšie časti pamäti RAM, umiestnenej nad koncom premenných. V tejto časti sa vykonávajú všetky výpočty **kalkulátora**, je tu **zásobník kalkulátora** a prenáša sa sem práve vykonávaný riadok programu. Rozloženie oblastí pre vyššie uvedené činnosti môžeme znázorniť nasledovne:



Ani jedna z týchto troch oblastí nemá svoje pevné miesto v pamäti. Ich začiatky a konce sa neustále pri vykonávaní programu presúvajú podľa toho, ako veľkú tú ktorú oblasť počítač potrebuje. Z tohto dôvodu sú ich začiatky a konce uložené v systémových premenných E_LINE (adresa začiatku editačnej oblasti), WORKSP (adresa začiatku pracovného priestoru), STKBOT (adresa dna zásobníka kalkulátora) a STKEND (adresa vrchu zásobníka pre kalkulátor). **Vrch zásobníka procesora - Z80** je označený SP (Stack Pointer), viac sa o ňom dozviete v časti venovanej iba jemu.

2.3.2.8. Editačná oblasť

Táto časť pamäti slúži na editáciu riadkov a ich nasledovnému vykonávaniu. Znaky, ktoré sú zadávané z klávesnice sú vyhodnotené systémom. Znaky a tokeny sú uložené do editačnej oblasti v poradí, ako boli stlačené. Editačné klávesy sú vyhodnotené zvlášť a potom je v editačnej oblasti vykonaná činnosť, ktorú označujú (zrušenie znaku, posun po riadku doľava, doprava apod.). Po stlačení každého klávesu je riadok v editačnej oblasti zobrazený na obrazovku do editačnej zóny (spodná časť obrazovky). Až tam vidíte, čo ste napísali.

Po stlačení klávesu <ENTER> prebehne **syntaktická kontrola** riadku. V prípade, že je syntax správna, riadok sa z editačnej oblasti zaradí do programu (pokiaľ má číslo) alebo sa vykoná.

Zaradenie riadku do programu vykoná systém veľmi jednoducho. Zistí si, či je voľné miesto pre riadok v oblasti pre program. Pokiaľ nie, oznámi Vám to "zatrúbením". V tomto prípade neostáva nič iné, než riadok stlačením <EDIT> vymazať z editačnej oblasti a urobiť preň miesto pomocou príkazu

CLEAR (vysvetlíme v ďalšej kapitole) alebo program skrátiť výmenou čísiel za niektoré výrazy ako sme uviedli v kapitole 2.3.2.5.

V prípade, že voľné miesto je, systém zistí, či sa v programe riadok s rovnakým číslom už nevyskytuje. Pokiaľ nie, tak je nájdený riadok programu s číslom najbližším vyšším, než je číslo nového riadku. Oblasť, začínajúca týmto "vyšším " riadkom sa až po koniec programu presunie o toľko bytov smerom ku koncu pamäti, koľko predstavuje dĺžka nového riadku. Na takto vzniknuté miesto sa prenesie z editačnej oblasti nový riadok.

Pokiaľ sa už v programe vyskytuje riadok s rovnakým číslom ako má nový zadaný riadok, starý riadok sa zruší takto: oblasť nového riadku až po koniec programu sa prenesie o toľko bytov smerom k začiatku pamäti, koľko predstavuje dĺžka starého riadku. Tým prestal riadok s rovnakým číslom ako má nový riadok v programe existovať a nový riadok sa do programu zaradí.

Pri vykonaní riadku (bez čísla) vykonáva systém jednotlivé príkazy uložené v editačnej oblasti, pokiaľ nenarazí na koncový byte 80h. Potom riadok z editačnej obasti vymaže púhym presunom bytu 80h na začiatok editačnej oblasti (teda tam, kam ukazuje systémová premenná E_LINE).

Pri vykonávaní programu (príkazom RUN, GO TO) sa postupne prenášajú riadky programu do editačnej oblasti (už bez čísiel) a tam sú vykonané. Vo všetkých prípadoch práce s editačnou oblasťou sa menia systémové premenné začínajúce VARS a STKEND.

2.3.2.9. Pracovný priestor, zásobník kalkulátora

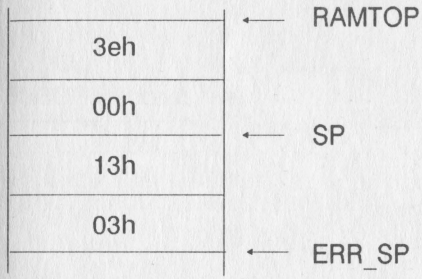
Pracovný priestor a zásobník kalkulátora sú využívané pri práci **kalkulátora**, t.j. tej časti operačného systému, ktorá má na starosti všetky aritmetické a logické operácie a prácu s reťazcami.

Umiestnenie obidvoch oblastí sa počas vkladania programu do počítača a pri vykonávaní programu mení. Voľná oblasť medzi dnom zásobníka kalkulátora a vrchom zásobníka procesora - Z80 je vyhradená práve pre tieto posuny (pracovná oblasť a zásobník kalkulátora sa pri zväčšovaní dĺžky programu posúvajú smerom k vyšším adresám).

Pri vykonávaní príkazov je tiež testovaná veľkosť voľnej pamäti medzi dnom zásobníka kalkulátora a vrchom zásobníka procesora - Z80.

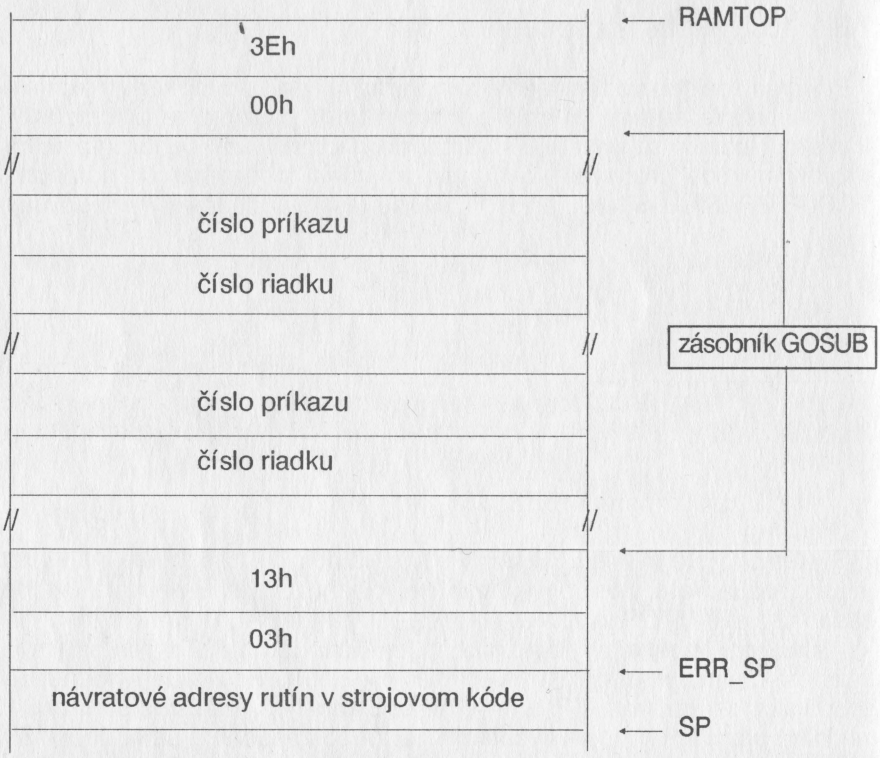
2.3.2.10. Zásobník procesora Z80

Pre prácu celého systému je veľmi dôležitá tá časť pamäti, kde sú uložené tzv. **návratové adresy volania podprogramov**. Nazývame ju zásobník. Návratové adresy sa vytvárajú automaticky ako pri volaní podprogramov v strojovom kóde procesora Z-80, tak pri volaní podrogramov pomocou GOSUB v BASICu. Štruktúra zásobníka po resete počítača je nasledovná:



Systémová premenná ERR_SP ukazuje na položku zásobníka, na ktorej je uložená adresa obsluhy ľubovoľnej chyby, SP je ukazovateľ vrchu zásobníka a RAMTOP ukazovateľ konca pamäti pre BASIC. Koniec zásobníka (jeho dno) je označený bytom obsahujúcim hodnotu 3Eh. Tento byte je tiež posledným bytom využiteľnej pamäti pre BASIC. Pri vkladaní hodnôt do zásobníka sa hodnota SP zmenšuje (zásobník sa zväčšuje smerom k začiatku pamäti RAM), pri vyberaní hodnôt zo zásobníka sa hodnota SP zväčšuje (zásobník sa zmenšuje smerom ku koncu pamäti RAM).

Pri výskyte nejakej chyby, ktorá vyvolá výpis chybového hlásenia, vyzdvihne systém obsah dvoch bytov adresovaných premennou ERR_SP a na túto adresu predá riadenie. Zmenou obsahu tejto položky zásobníka je možné dosiahnuť napr. reset počítača po stlačení BREAK pri nahrávaní programu, tak ako to poznáme u niektorých hier. Stačí len na tieto dva byty uložiť nuly. Pri vykonávaní riadku v editačnej oblasti zásobník vyzerá nasledovne:



Adresa obsluhy chyby je nastavená na hodnotu 1303h. Na tejto adrese je podprogram obsluhujúci všetky možné chyby a vypisujúci v spodnej časti obrazovky chybové hlásenie.

Ďalej sa v zásobníku objavila položka **zásobník GOSUB**. Tá slúži na určenie miesta návratu v programe pri použití príkazu GOSUB, teda pri volaní podprogramu v BASICu. Na zásobník sa uloží číslo aktuálneho riadku (dva byty) a číslo príkazu (jeden byte) nasledujúceho za príkazom GOSUB, teda miesto v programe, kde sa má po vykonaní podprogramu (po príkaze RETURN) pokračovať. Zásobník GOSUB môže obsahovať viacej ukazovateľov návratových miest v programe, pričom táto štuktúra zodpovedá vnorenému volaniu podprogramov GOSUB. V prípade, že nebol volaný žiadny podprogram GOSUB, zásobník GOSUB je prázdny.

Ukazovateľ vrchu zásobníka môže pri vykonávaní programu meniť svoju hodnotu podľa toho, aké podprogramy v strojovom kóde systém práve využíva a tiež v prípade použitia užívateľských podprogramov v strojovom kóde (podprogramy v strojovom kóde budeme v ďalšom texte nazývať rutiny). Po skončení týchto rutín však musí byť ukazovateľ nastavený na to isté miesto, aké mal pred ich vyvolaním.

Systémová premenná RAMTOP označuje koniec pamäti pre BASIC. Jej hodnota sa dá meniť príkazom:

CLEAR *cislo*

kde *cislo* je nová hodnota RAMTOP (konca pamäti). Zmenou RAMTOPu môžete pamäť pre BASIC zmenšovať alebo zväčšovať, v závislosti na dĺžke Vášho programu. Po resete počítača je RAMTOP nastavený na hodnotu 65367.

Príkaz **CLEAR** vykonáva tieto činnosti:

- vymaže všetky premenné z oblasti premenných
- vymaže obrazovku
- v systémových premenných nastaví súradnicu posledného kresleného bodu na hodnoty 0,0
- vykoná RESTORE
- vymaže zásobník GOSUB
- v prípade použitia tvaru CLEAR *cislo* nastaví RAMTOP (pokiaľ je to možné) na hodnotu *cislo*. Nová hodnota RAMTOPu musí ležať medzi STKEND a P_RAMT.

Zmena RAMTOPu môže niekedy priniesť problémy. Skúste zadať počítaču CLEAR 23850. Teraz skúste do editačného riadku niečo napísať. Odpoveďou Vám bude len zatrúbenie editora, ktorý Vám takto signalizuje, že niečo nie je v poriadku. Zadaním hodnoty 23850 ste totiž vymedzili oblasť pre BASIC na púhych 95 bytov (za predpokladu, že program v BASICu začína na adrese 23755). Do tejto oblasti musí počítač umiestniť všetky vyššie spomenuté pracovné oblasti a neostáva mu miesto na editačnú oblasť, teda do nej nemôže umiestniť Vami zadávaný kláves. Neostáva nič iné než počítač zresetovať.

Pokiaľ sa Vám stane táto vec po odoslaní nejakého riadku programu, musíte ho z programu vypustiť, zmeniť RAMTOP, popr. skrátiť program a riadok do neho znovu zadať.

2.3.2.11. Voľná pamäť

Na adrese o jednu väčšej, než je hodnota RAMTOP, začína voľne použiteľná pamäť, ktorá je zhora ohraničená adresou uloženou v systémovej premennej P_RAMT.

Túto časť pamäti BASIC nepoužíva, je však prístupná cez príkaz POKE, PEEK a môžete do nej nahrávať súbory uložené na diskete príkazom SAVE "meno" CODE X,Y. Má jednu veľkú výhodu; jej obsah nemaže príkaz NEW (ten maže pamäť len po RAMTOP). Väčšinou sa používa na uloženie programov v strojovom kóde a na uloženie užívateľskej grafiky.

2.3.2.12. Umiestnenie užívateľskej grafiky

Počítač disponuje okrem **základnej sady znakov**, uloženej v pamäti ROM, ešte ďalšou sadou znakov označovaných ako **užívateľská grafika** (User Define Graphic - **udg**). Táto sada môže byť umiestnená kdekoľvek v pamäti RAM počítača a obsahuje 21 znakov, ktorých tvar si môže užívateľ sám definovať. Adresa začiatku udg je umiestnená v systémovej premennej UDG, dĺžka je 168 bytov.

Po resete počítača je premenná UDG nastavená na hodnotu 65368, teda na adresu prvého bytu za RAMTOPom. Pokiaľ chcete vo svojich programoch používať užívateľskú grafiku, doporučujeme umiestniť ju vždy až za RAMTOP, aby nebola premazaná BASICom, popr. príkazom NEW. Viacej sa o tvorbe znakov užívateľskej grafiky dozviete až po zoznámení sa s generátorom znakov umiestneným v pamäti ROM. Týmto sme skončili popis pamäti počítača.

2.4. Znakový generátor

Pri výpise znaku musí byť do pamäti kresby umiestnená na miesto, kde má byť znak vypísaný, **matica znaku** obsahujúca rozkreslenie znaku na jednotlivé body. Matica znaku má rozmery 8x8 bitov a v každom bite je umiestnená informácia, či daný bod (priradený bitu) má byť rozsvietený (bit má hodnotu 1) alebo zhasnutý (bit má hodnotu 0).

Matice znakov musia byť vytvorené pre všetky zobraziteľné znaky, teda od znaku medzery (kód 32) až po znak copyright (kód 127) a uložené v pamäti, odkiaľ sa pri výpise znaku kopírujú do pamäti kresby. Celková dĺžka pamäti, nutná pre matice všetkých znakov, je 96 * 8 = 768 bytov a oblasť vyhradená pre matice znakov sa nazýva znakový generátor (pomocou neho sú generované znaky na obrazovke).

Po resete počítača je znakový generátor umiestnený v pamäti ROM od adresy 3D00h = 15616 a táto adresa, zmenšená o 256, je umiestnená v systémovej premennej CHARS. Z toho tiež vyplýva, že znakový generátor môže byť umiestnený i v pamäti RAM, ale do systémovej premennej CHARS musí byť vložená adresa začiatku generátora znakov, zmenšená o 256.

Skúste teraz zadať POKE 23606,255:POKE 23607,0. Po odoslaní riadku sa nevypíše žiadne hlásenie, len v editačnej oblasti sa objaví riadok zložený z akýchsi hieroglyfov. Vložením týchto dvoch čísiel do premennej CHARS ste totiž zmenili začiatok generátora znakov na adresu nula. A na tejto adrese je uložená rutina v strojovom kóde, žiaden generátor znakov. Zresetujte počítač.

Ostáva len povedať, ako taký vlastný generátor znakov vytvoríte. Základom je vytvoriť matice všetkých znakov. Tie potom za sebou uložíte po jednotlivých bytoch do pamäti.

Maticu znaku si môžete predstaviť ako šachovnicu s veľkosťou 8x8. Do nej musíte zakresliť znak. Predpokladajme, že chcete vytvoriť maticu znaku s kódom 33, čo je výkričník.

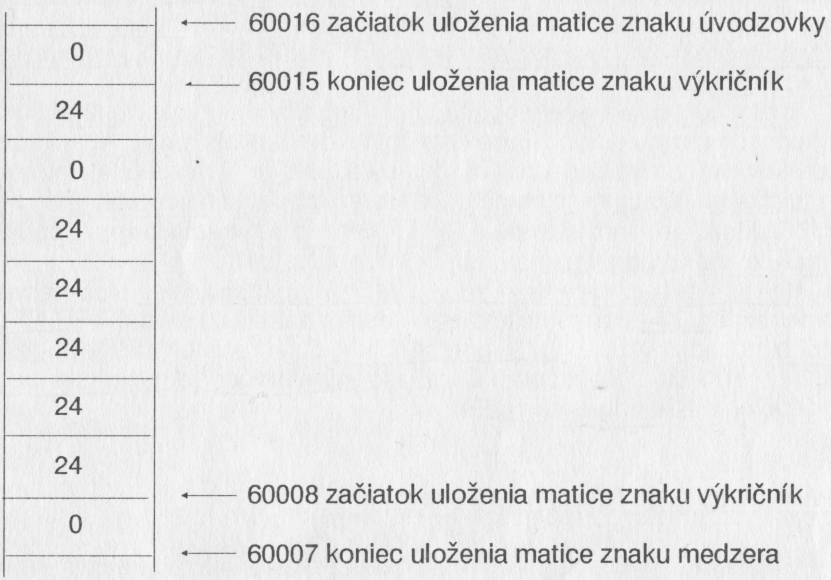
Nakreslite si šachovnicu:

	7	6	5	4	3	2	1	0	bit
0	0	0	0	1	1	0	0	0	BIN 00011000 = 24
1	0	0	0	1	1	0	0	0	BIN 00011000 = 24
2	0	0	0	1	1	0	0	0	BIN 00011000 = 24
3	0	0	0	1	1	0	0	0	BIN 00011000 = 24
4	0	0	0	1	1	0	0	0	BIN 00011000 = 24
5	0	0	0	0	0	0	0	0	BIN 00000000 = 0
6	0	0	0	1	1	0	0	0	BIN 00011000 = 24
7	0	0	0	0	0	0	0	0	BIN 00000000 = 0

byte

Týmto sme vytvorili maticu znaku "!". Predpokladajme, že generátor znakov má byť uložený od adresy 60000. Od adresy 60000 po adresu 60007 je uložená matica znaku medzery (8 bytov), od adresy 60008 po adresu 60015 je uložená matica znaku výkričník, od adresy 60016 po adresu 60023 matica znaku úvodzovky hore atď.

Časť pamäti vyzerá nasledovne:



Podobným spôsobom môžete vytvoriť matice všetkých znakov od medzery až po copyright a uložiť ich na príslušné adresy v generátore znakov.

Adresu začiatku osmice bytov vyjadrujúcich maticu znaku dostanete týmto výpočtom:

Adresa = AdresaGeneratoraZnaku + 8*(CODE"znak" – 32)

V našom príklade: Adresa = 60000 + 8*(CODE"!" – 32) =
60000 + 8*(33 – 32) =
60000 + 8 = 60008

2.5. Užívateľská grafika (udg)

Počítač disponuje okrem základnej sady znakov ešte ďalšou sadou. Tá sa nazýva užívateľsky definovaná grafika (udg) a slúži na výpis znakov, ktoré si sami vytvoríte pre použitie vo vlastných programoch.

Znak užívateľskej grafiky je možné vypísať po prechode do grafického módu (stlačením <CAPS SHIFT> + <9>) stlačením jedného z klávesov

<A> až <U>. Užívateľská grafika má teda rozsah 21 znakov a zaberá v pamäti miesto 168 bytov. Po resete počítača je uložená od adresy 65368 a na túto adresu je nastavená systémová premenná UDG. Zmenou jej hodnoty je možné meniť adresu začiatku udg, vždy ju však doporučujeme umiestniť nad RAMTOP.

Znaky udg sa po resete počítača rovnajú stlačenému klávesu. Pri tvorbe vlastných znakov udg postupujeme úplne rovnako, ako pri vytváraní nových znakov pre znakový generátor. Jediný rozdiel je pri vypočítavaní adresy začiatku matice znaku v pamäti, pre ktorej vyčíslenie môžete použiť funkciu USR, ktorá pri použití tvaru PRINT USR "*znak*" **vracia adresu začiatku matice znaku udg** v pamäti.

Uvažujme, že chcete nájsť adresu začiatku matice znaku, uloženého pod klávesom <C>. Po resete počítača napíšte nasledujúci riadok: PRINT USR "C" a na obrazovku sa vypíše adresa 65384, čo je začiatok matice udg znaku uloženého pod klávesom "C". Podobným spôsobom získate adresy všetkých matíc udg znakov.

2.6. Funkcia USR, rutiny z ROM

V tejto krátkej časti sa budeme venovať niektorým zaujímavým rutinám (podprogramom) z pamäti ROM, teda z operačného systému počítača, ktoré by Vám mohli uľahčiť Vaše počiatočné kroky pri práci v strojovom kóde.

Na začiatok pripomeňme funkciu USR. Táto funkcia slúži na volanie podprogramov v strojovom kóde. Pokiaľ jej argument označíme ako **x**, tak x môže byť ľubovoľný celočíselný výraz z intervalu 0 až 65535 (v prípade použitia u udg môže byť argumentom znak uložený pod klávesom <A> až <U>. V tejto časti budeme uvažovať ako argument iba číslo). Pri interpretácii sa vyhodnotí argument *x* a riadenie je predané na adresu pamäti *x*. Návrat zo strojového kódu musí byť zakončený inštrukciou RET a v prípade, že bolo zakázané prerušenie inštrukciou DI, musí byť opäť pred návratom do BASICu povolené inštrukciou EI.

Pri volaní funkcie USR je hodnota argumentu *x* predaná do registrového páru BC a po návrate je výsledkom funkcie posledná hodnota tohto registrového páru.

Počiatočnú prácu v strojovom kóde Vám môžu uľahčiť nasledujúce rutiny z ROM:

RUTINY Z ROM

- vymazanie celej obrazovky - okrem horných desiatich riadkov
  RANDOMIZE USR 3652

- posun celej obrazovky o jeden riadok hore:
  RANDOMIZE USR 3280
  USR 3232 - SCROLL?
- posun celej obrazovky o 22 riadkov smerom hore:
  RANDOMIZE USR 3330

- výpis dĺžky voľnej pamäti pre program v BASICu:
  PRINT 65535 - USR 7962

- reset počítača:
  RANDOMIZE USR 0

- čakanie na stlačenie ľubovoľného klávesu; po jeho stlačení sa vypíše dĺžka čakania v 1/50 sek.:

  PRINT USR 7997 - USR 7997

- výpis textového reťazca - 203Ch (8252):

```
:
LD A,02h                (kanál pre výpis na obrazovku)
CALL 1601h              (otvorenie kanála)
LD DE,počiatočná adresa textu v pamäti
LD BC,dĺžka textu v bytoch
CALL 203Ch
:
```

- pípnutie - 03B5h (949)

```
:
LD DE,dĺžka tónu
LD HL,výška tónu
CALL 03B5h
:
```

- save na pásku - 04C2h (1218):

```
:
LD IX,počiatočná adresa úseku pamäti pre SAVE
LD DE,dĺžka úseku v bytoch
LD A,značkový byte      (obvykle hlavička 0, telo 255)
CALL 04C2h
:
```

- load z pásky - 0556h (1366): RANDOMIZE USR 1366

```
:
LD IX,počiatočná adresa
LD DE,dĺžka
LD A,značkový byte      (obvykle hlavička 0, telo 255)
SCF
CALL 0556h
:
```

- zmena farby bordera - 2298h (8859):

```
:
LD A,farba bordera (môže byť z intervalu<0 až 7>)
CALL 2298H
:
```

- vykreslenie bodu so súradnicami X, Y — 22E5h (8933):

```
:
LD B,Y          (súradnica môže byť z intervalu <0 až 175>)
LD C,X          (súradnica môže byť z intervalu <0 až 255>)
CALL 22E5H
:
```

- cls (vyčistenie obrazovky) - 0D68h (3432): RANDOMIZE USR 3432

```
:
LD A,02h (kanál pre výpis na obrazovku)
CALL 1601h (otvorenie kanála)
CALL 0D68H
:
```

- zistenie výskytu bodu so súradnicami X,Y - 22CEh (8910):

```
:
LD B,Y          (súradnica môže byť z intervalu<0 až 175>)
LD C,X          (súradnica môže byť z intervalu <0 až 255>)
CALL 22CEH      (v A reg. je výsledok: 1/0 – bod je/nie je)
:
```

Používanie podprogramov z ROM disketovej jednotky je podstatne zložitejšie, pretože MDOS v sebe nemá zabudovaný príkaz pre odpojenie systémovej ROM a pripojenie MDOS ROM.

Existuje niekoľko ciest, ako ovládať disketovú jednotku zo strojového kódu. Uvedieme príklad, ktorý využíva interpreter BASICu. Jeho činnosť je nasledujúca:

- do editačnej zóny určenej systémovou premennou E-LINE vloží príkaz pre disketovú jednotku
- nastaví všetky potrebné systémové premenné
- zavolá interpreter BASICu na adrese 18bah

Príklad (preložený pomocou programu MRS):

```
 3  600a
 4  600a                org     #a000
 5  a000
 6  a000        eline   equ     #5c59
 7  a000        worksp  equ     #5c61
 8  a000        chadd   equ     #5c5d
 9  a000        subppc  equ     #5c47
10  a000        nsppc   equ     #5c44
11  a000        runprg  equ     #1b8a
12  a000        setmin  equ     #16b0
13  a000        list    equ     #f0
14  a000        cat     equ     #cf
15  a000        print   equ     #f5
16  a000        chr     equ     #c2
17  a000        rem     equ     234
18  a000        cls     equ     251
19  a000        errsp   equ     #5c3d
```

```
20  a000                 errnum equ   #5c3a
21  a000                 cr     equ   #0d
22  a000                 read   equ   #e3
23  a000                 val    equ   #b0
24  a000
25  a000
26  a000  fb             p      ei
27  a001  115aa0                ld    de,text1
28  a004  cd08a0                call   prikaz
29  a007  c9                    ret
30  a008
31  a008   e5            prikaz push  hl
32  a009  d5                    push  de
33  a00a  c5                    push  bc
34  a00b  cd4ba0                call  settxt
35  a00e  2a595c                ld    hl,(eline)
36  a011  225d5c                ld    (chadd),hl
37  a014  fd3600ff              ld    (iy+#00),#ff
38  a018  fd360a01              ld    (iy+#0a),#01
39  a01c  fdcb01fe              set   7,(iy+#01)
40  a020
41  a020  2a3d5c                ld    hl,(errsp)
42  a023  2295a0                ld    (olderr),hl
43  a026  2132a0                ld    hl,error
44  a029  e5                    push  hl
45  a02a  ed733d5c              ld    (errsp),sp
46  a02e  e1                    pop   hl
47  a02f
48  a02f  cd8a1b                call  runprg
49  a032  cdb016         error  call  setmin
50  a035  2a95a0                ld    hl,(olderr)
51  a038  223d5c                ld    (errsp),hl
52  a03b  fd7e00                ld    a,(iy+#00)
53  a03e  3c                    inc   a
54  a03f  fd3600ff              ld    (iy+#00),#ff
55  a043  fd360a00              ld    (iy+#0a),#00
56  a047  c1                    pop   bc
57  a048  d1                    pop   de
58  a049  e1                    pop   hl
```

```
59  a04a  c9                ret
60  a04b
61  a04b  2a595c      settxt ld    hl,(eline)
62  a04e  1a          settx1 ld    a,(de)
63  a04f  77                 ld    (hl),a
64  a050  23                 inc   hl
65  a051  13                 inc   de
66  a052  fe80               cp    #80
67  a054  20f8               jr    nz,settx1
68  a056  22615c             ld    (worksp),hl
69  a059  c9                 ret
70  a05a
71  a05a
72  a05a  f02a        text1  db    list,'*'
73  a05c  0d80               db    cr,#80
74  a05e
75  a05e  cf22613a    text11 db    cat,'"a:*.*"'
          2a2e2a22
76  a066  0d80               db    cr,#80
77  a068
78  a068  e3          text2  db    read
79  a069  22623a22           db    '"b:",'
          2c
80  a06e  30                 db    '0'
81  a06f  0e                 db    #0e
82  a070  0000               dw    #00
83  a072  0000               dw    #00
84  a074  00                 db    #00
85  a075  2c                 db    ','
86  a076  223022             db    '"0"'
87  a079  0000               dw    #00
88  a07b  0080               dw    #8000
89  a07d  00                 db    #00
90  a07e  0d80               db    cr,#80
91  a080
92  a080  f52330      text3  db    print,'#','0'
93  a083  0e                 db    #0e
94  a084  0000               dw    #00
95  a086  02 kanal           db    #02
```

```
 96    a087  0000              dw    #00
 97    a089  3b                db    ';'
 98    a08a  c230              db    chr,'0'
 99    a08c  0e                db    #0e
100    a08d  0000              dw    #00
101    a08f  00          kod   db    #00
102    a090  0000              dw    #00
103    a092  3b                db    ';'
104    a093  0d80              db    cr,#80
105    a095
106    a095  0000        olderr dw     #00
107    a097
108    a097              z
109    a097
110    a097                    end
```

eline	5c59	*	worksp	5c61	*	chadd	5c5d	*	z	a097	*
list	00f0	*	cat	00cf	*	text2	a068	*	errnum	5c3a	*
nsppc	5c44	*	p	a000	*	settxt	a04b	*	settx1	a04e	*
cr	000d	*	errsp	5c3d	*	print	00f5	*	olderr	a095	*
error	a032	*	text1	a05a	*	rem	00ea	*	subppc	5c47	*
cls	00fb	*	prikaz	a008	*	runprg	1b8a	*	chr	00c2	*
setmin	16b0	*	text3	a080	*	kod	a08f	*	kanal	a086	*
read	00e3	*	val	00b0	*	text11	a05e	*			

error:0

Veríme, že postupom času prídete na rad rozmanitých rutín z pamäti ROM operačného systému, ktoré budete s úspechom využívať v programoch.

2.7. Práca s V/V kanálmi

V tejto časti sa budeme viacej venovať práci s V/V kanálmi počítača, pomocou ktorých sa dajú obsluhovať rôzne **periférne zariadenia** (niekedy ich označujeme tiež ako vstupno-výstupné zariadenia, V/V zariadenia, periférie).

Systém, ktorý umožňuje pripojenie týchto periférií, je úplne univerzálny. K počítaču totiž možno pripojiť V/V zariadenie s ľubovoľným druhom komunikácie. Pre toto prepojenie musíte poznať, ako pracujú **kanály** a **linky**

Vášho počítača, pretože ten so svojimi perifériami komunikuje práve pomocou týchto kanálov a liniek.

Linka je cesta, ktorou sa posielajú jednotlivé spracúvané znaky z počítača do periférie, alebo sú perifériou vysielané a počítač ich prijíma. Typ periférie je označovaný ako kanál, pričom kanál nemá nič spoločné s hardwareovou konštrukciou periférie a ide len o programové zaistenie komunikácie medzi hlavným programom a vstupno/výstupnými programami pre obsluhu periférií. Ako už viete z kapitoly 2.3.2.4., každý kanál obsahuje adresu vstupnej rutiny, výstupnej rutiny a typ periférie, pre ktorú je kanál určený.

Tabuľka liniek (jej začiatok je uložený v systémovej premennej STRMS = 23568) má dĺžku 38 bytov (19 dvojbytových hodnôt) a sú v nej uložené informácie o všetkých linkách. Prvé tri hodnoty prináležia linkám −3 až −1, ktoré používa BASIC (pre editor, tlač hlásenia o chybe atď.). Tieto linky nie sú obvyklými príkazmi prístupné a nie je dobré do nich zasahovať ani príkazom POKE.

Štvrtá až devätnásta hodnota (pre linky nula až pätnásť) určujú, na ktorý kanál je linka napojená nasledovne: k hodnote uloženej v systémovej premennej CHANS (začiatok kanálov) sa pripočíta hodnota uložená v tabuľke liniek na mieste prislúchajúcom danej linke, a výsledkom je adresa kanála, na ktorý je linka pripojená.

Tabuľka liniek po resete obsahuje tieto hodnoty:

linka	posun	typ kanálu
15	0	uzatvorený
.	.	.
.	.	.
4	0	uzatvorený
3	16	tlačiareň
2	6	obrazovka
1	1	klávesnica
0	1	klávesnica
-1	11	BASIC editor
-2	6	obrazovka
-3	1	klávesnica

← STRMS

Pre prácu s linkami slúžia dva príkazy BASICu. Prvý z nich je príkaz **OPEN#**. Pomocou neho môžete priradiť ľubovoľnej linke 0 až 15 ľubovoľný dostupný kanál. Po príkaze OPEN #5,"s" je linke s poradovým číslom 5 priradený kanál typu obrazovka. Tabuľka liniek po tomto príkaze vyzerá nasledovne:

linka	posun	typ kanálu
15	0	uzatvorený
.	.	.
.	.	.
5	6	obrazovka
4	0	uzatvorený
3	16	tlačiareň
2	6	obrazovka
1	1	klávesnica
0	1	klávesnica
-1	11	BASIC editor
-2	6	obrazovka
-3	1	klávesnica

← STRMS

Z uvedeného príkladu vyplýva, že na jeden kanál môže byť nasmerovaných viacej liniek, ale jedna linka nemôže súčasne ukazovať na viacero kanálov.

Druhým príkazom pre prácu s linkami je príkaz **CLOSE#**. Tento príkaz uzatvára linku, teda vlastne dosadí v tabuľke liniek na miesto vyhradené danej linke nuly a tým linka prestane byť aktívna. Pokiaľ zadáte CLOSE #5, tabuľka liniek vyzerá rovnako, ako pred otvorením linky. Teraz môžete linku opäť otvoriť, napr. pre typ kanála tlačiareň príkazom OPEN #5,"p".

Ostáva už len popísať, akým spôsobom sa dá napojiť do daného kanálového systému ľubovoľná vstupná alebo výstupná rutina. Je treba uložiť jej adresu na patričné miesto v tabuľke kanálov. Uvedieme príklad na napojenie výstupnej rutiny pre tlačiareň na tretiu linku (tá je tlačiarni štandardne vyhradená).

```
open_p ld     hl,(2*6+STRMS)    posun od CHANS do HL
       ld     (de), CHANS       začiatok kanálov do DE
       add    hl, de            adresa kanála v HL
       push   hl                uschovaj ju
       ld     hl,outp-open_p    relatívna adresa OUTP oproti
                                začiatku tejto rutiny
       add    hl,bc             adresa OUTP v HL
       pop    de                späť adresa kanála
       ex     de,hl             vymeň DE a HL
       ld     (hl),d            vyšší byte OUTP do tabuľky
                                kanálov
       dec    hl
       ld     (hl),e            nižší byte OUTP do tabuľky
                                kanálov
       ret

outp   :                        vyslanie znaku na
       :                        periférne zariadenie
       ret                      návrat
```

Po vykonaní tejto rutiny (RANDOMIZE USR OPEN_P) je nastavená adresa výstupnej rutiny tretieho kanála na adresu OUTP a tá zaisťuje výpis všetkých znakov (poslaných cez tretiu linku) na tlačiareň.

Zatiaľ sme sa zaoberali len možnosťou presmerovať nejakú linku na už existujúci kanál. Čo je však treba urobiť, pokiaľ chceme otvoriť ešte ďalšie kanály ? Odpoveďou by Vám mohla byť následujúca rutina, ktorá vytvorí miesto pre ďalšie dva kanály (kanály 4 a 5) a vykoná ich inicializáciu. Po vykonaní rutiny môžete používať tvary PRINT #3, PRINT #4 a PRINT #5, pričom výstupné rutiny týchto kanálov sú odlišné a v praxi to znamená, že môžete mať pripojené tri rôzne zariadenia typu tlačiareň pre výstup (napr. na kanáli 3 tlačiareň, na kanáli 4 dierovač diernej pásky a na kanáli 5 zapisovač). Táto rutina je relokovateľná, otvorí štvrtý kanál pre vstup/výstup a piaty kanál tiež pre vstup/výstup.

```
zac    push   bc                v reg.páre BC adresa začiatku rutiny
       ld     hl,(23631)        v HL adresa začiatku kanálov
       ld     bc,#14            4h=20d,teda posun pre 4. kanál
       add    hl,bc a           adresa 4. kanála
       push   hl                uschovaj ju
```

```
        ld      a,(hl)          obsah prvého bytu 4. kanála
        cp      #80             je kanál už vytvorený ?
        jr      nz,obn          pokiaľ áno,obnov len adresy rutín
        ld      bc,10           miesto pre dva kanály = 2*5 = 10 bytov
        call    #1655           vytvor miesto pre dva kanály
obn     pop     de v DE         adresa 4. kanálu
        pop     hl v HL         adresa začiatku tejto rutiny
        ld      bc,tabk-zac     v BC rel. posun tabuľky oproti začiatku
        add     hl,bc v HL      adresa tabuľky nových hodnôt kanálov
        ld      bc,10           v BC dĺžka tabuľky - 10 bytov
        ldir                    prenos tabuľky do kanálovej oblasti
        ld      hl,23568 v HL   začiatok oblasti liniek
        ld      bc,14 v BC      posun pre 4. linku ( 2*7 = 14)
        add     hl,bc v HL      adresa 4. linky
        ld      bc,21           4.linka smeruje na 4.kanál (21 = 1 + 4*5)
        ld      (hl),c          inicializuje 4. linku pre 4. kanál
        inc     hl
        ld      (hl),b
        ret                     koniec inicializačnej rutiny

tabk    dw      outpunch        adresa rutiny výstupu pre 4. kanál
        dw      inpunch         adresa rutiny vstupu pre 4. kanál
        db      "P"             typ tlačiareň
        dw      outplott        adresa rutiny výstupu pre 5. kanál
        dw      inplott         adresa rutiny vstupu pre 5. kanál
        db      "P"             typ tlačiareň
```

Týmto príkladom skončíme prácu s V/V kanálmi. Na ich lepšie pochopenie Vám doporučujeme vytvoriť si niekoľko vlastných programov pre vstup a výstup. Nepotrebujete k tomu žiadne periférne zariadenie, stačí len vytvoriť rutiny, z ktorých jedna niekam do pamäti zapisuje (napr. príkazom PRINT #3; text) a druhá tento text zasa z pamäti číta (napr. príkazom INPUT #3; a$).

2.8. Práca s V/V miestami

Okrem zápisu a čítania hodnôt z pamäti môže procesor zapisovať a čítať ešte z ďalších miest, ktoré sa nazývajú V/V miesta. Celkom ich je (podobne ako pamäťových miest) 65536 a používajú sa pri komunikácii procesora s perifériami. Inštrukcie určené pre prácu s V/V miestami sú inštrukcie OUT a

IN, veľmi podobné inštrukciam POKE a PEEK. Inštrukcia OUT zapisuje na V/V miesto danú hodnotu. Jej tvar je: OUT *adresa, hodnota* kde *adresa* je z rozsahu 0 až 65535 (teda dva byty) a *hodnota* z rozsahu 0 až 255 (teda jeden byte). Čo sa pri použití tejto inštrukcie stane?

Počítač má celkom 16 adresových liniek, ktorými adresuje pamäť a porty od 0 do 65535. Ďalej má 8 dátových liniek, na ktorých sa môže objaviť hodnota 0 až 255 (týchto liniek je 8, pretože Váš počítač je osembitový, u šestnásťbitového počítača je 16 dátových liniek atď.) a pomocou ktorých sa uskutočňuje výmena dát medzi pamäťou a procesorom a medzi V/V miestami a procesorom. V/V miesto, na ktoré je pripojené fyzické V/V zariadenie sa, nazýva **port**.

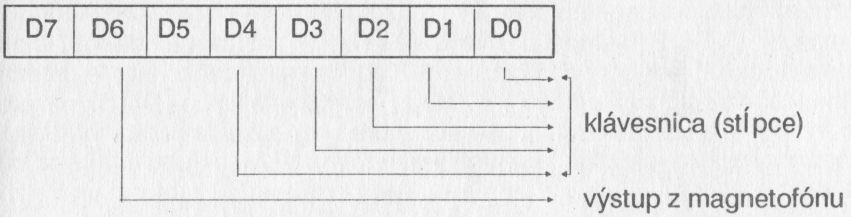
Po inštrukcii **OUT** *adresa, hodnota* sa na adresových linkách objaví zadaná adresa a na dátových linkách zadaná hodnota. Periféria, ktorá je na port s vyslanou adresou pripojená, si prečíta hodnotu na dátových linkách a ďalej ju (už nezávisle na počítači) spracuje. Týmto spôsobom, teda zápisom nejakých hodnôt na port, ku ktorému je pripojená nejaká periféria, môže počítač s touto periférioou komunikovať.

Veľmi podobne pracuje aj funkcia **IN**, ktorá sa používa v tvare: **IN** *adresa*. Po dekódovaní inštrukcie IN procesor načíta hodnotu z dátových liniek práve v čase, kedy sa na adresových linkách objaví adresa V/V miesta. Takto môže periféria pripojená na nejaký port zasielať počítaču hodnoty. Pomocou portov teda prebieha komunikácia počítača s vonkajšími perifériami. Okrem toho počítač pomocou portu komunikuje s vnútornými perifériami (napr. obvod ULA1, radič diskov, interface 8255).

2.8.1. Zákaznícky obvod ULA1

Zákaznícky obvod zaisťuje zobrazovanie na obrazovke, komunikáciu počítača s magnetofónom, zvuk reproduktora a načítanie klávesnice. Zákaznícky obvod má adresu **254**.

Funkcia IN načíta s portu 254 hodnotu, v ktorej význam jednotlivých bitov je nasledujúci:



Celá klávesnica je rozdelená do ôsmich radov po piatich klávesoch (päť stĺpcov). Každý rad je možno čítať na inej adrese, pričom päť stĺpcov predstavuje päť nižších dátových bitov (D0 až D4), na bitoch D5 a D7 je hodnota 1, hodnota bitu D6 môže byť 0, alebo 1.

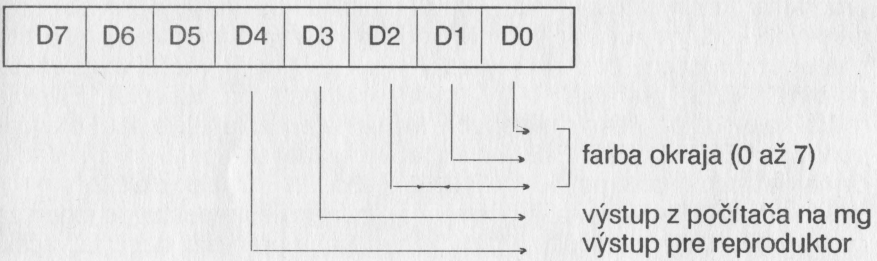
Klávesnicu si môžeme znázorniť nasledovne:

port	vyšší byte polriadok	nižší byte stĺpec	D4	D3	D2	D1	D0
65278	11111110	11111110	V	C	X	Z	CAP.S
65022	11111101	11111110	G	F	D	S	A
65410	11111011	11111110	T	R	E	W	Q
63486	11110111	11111110	5	4	3	2	1
61438	11101111	11111110	6	7	8	9	0
57342	11011111	11111110	P	O	I	U	Y
49150	10111111	11111110	H	J	K	L	ENTER
32766	01111111	11111110	B	N	M	SYM.S	SPACE

Jednotlivé bity v načítanom byte sú nastavené na 0, pokiaľ bol niektorý kláves v stĺpci stlačený, a na 1 pokiaľ žiaden kláves v danom stĺpci stlačený nebol.

Bit D6 slúži k nahrávaniu dát z magnetofónu do počítača. Signál, ktorý bol zaznamenaný na magnetofónovú kazetu, sa sníma a podľa jeho hodnoty sa nastaví bit D6. Pri vstupe logickej nuly sa bit D6 nastaví na hodnotu 0, pri vstupe logickej jednotky na hodnotu 1. Hodnota načítaného bitu je určená dĺžkou trvania signálu logickej nuly alebo jednotky na vstupe D6, Signál pre hodnotu jedna je dvakrát dlhší ako signál pre hodnotu nula načítaného bitu. Z ôsmich načítaných bitov sa vytvorí jeden byte a takto sa do počítača nahrá celý blok dát.

Príkaz OUT na port 254 slúži k ovládaniu farieb okraja, reproduktora a pre nahrávanie dát z počítača na magnetofón. Pri inštrukcii OUT je význam bitov vo vyslanom byte nasledujúci:



Farba okraja môže byť v rozsahu čierna (hodnota 0) až biela (hodnota 7).

Pre nahrávanie z počítača na magnetofón slúži bit D3. Spôsob nahrávania z počítača je obdobný ako pri nahrávaní z magnetofónu. Hodnota každého bitu z bytu je vysielaná po určitý čas na bit D3. Logické hodnoty 0, 1 z tohto bitu sú upravené na signály s príslušnou napäťovou úrovňou, ktoré sa vyšlú na vstup magnetofónu pre nahrávanie. Týmto postupom sa všetky byty nahrávaného bloku dát zapíšu na pásku.

Reproduktor je ovládaný striedaním jednotky a nuly na bite D4 portu 254. Správnym striedaním núl a jednotiek je dosiahnutý žiadaný tón.

2.9. Interface

Pre styk počítača s perifériami sa môže použiť technický alebo programový prostriedok, ktorý prácu s perifériou uľahčí. Tento prostriedok sa nazýva **interface** (interfejs), čo môžeme preložiť ako "medzitvár", u nás sa používa termín **rozhranie**.

Interface je teda akýmsi rozhraním medzi počítačom a perifériou, pričom zabezpečuje to, že počítaču sa javí určitá skupina periférií ako rovnaká, má rovnakú tvár. Ich technická konštrukcia však môže byť úplne odlišná.

Podobne interface zabezpečuje, že periférii sa javí istá skupina počítačov rovnako. Potom môžeme hovoriť o **type interface**, umožňujúcom komunikáciu medzi určitým typom počítačov a periférií. Najrozšírenejšími typmi

sú **interface paralelný** a **interface sériový**. Nazývajú sa podľa typu prenosu dát medzi počítačom a perifériou.

Paralelný interface sa vyznačuje tým, že všetkých osem bitov dátového bytu je prenášaných naraz. Výhodou je jednoduchá obsluha zo strany programátora a rýchlosť prenosu, nevýhodou nutnosť mať pre prenos osem dátových vodičov a u niektorých typov paralelných interface môže byť prenos uskutočnený iba do malej vzdialenosti, asi 2 m. Pri väčšej vzdialenosti sa objavujú pri prenose chyby (malá odolnosť voči rušeniu). Existuje niekoľko typov paralelných interface, medzi najpoužívanejšie patrí pravdepodobne CENTRONICS. Elektronický obvod, ktorý sa v paralelných interface najviac používa, je obvod MHB 8255 alebo Z80PIO, ale i ďalšie.

Sériový interface prenáša osem bitov jedného bytu postupne jeden za druhým. Používa sa väčšinou tam, kde je treba prenášať dáta na väčšiu vzdialenosť (podľa typu sériového interface na niekoľko desiatok metrov, v prípade počítačových sietí spojených cez telekomunikačnú sieť až na vzdialenosť niekoľko tisíc kilometrov) s vylúčením možných chýb.

Na prenos sa používajú dva a viacej vodičov. Typy sériových interface môžu byť napr. RS 232 a V24. Elektronický obvod zaisťujúci sériovú komunikáciu je obvod MHB 8251, alebo Z80SIO, ale i ďalšie.

Okrem **technického interface** môže existovať ešte **programový interface**. Ten môže byť použitý všade tam, kde je treba pripraviť dáta do formy určenej príslušnému typu periférie. Typickým príkladom môže byť súbor rutín v strojovom kóde, ktoré umožnia nahrávať dáta z počítača na magnetofón a späť.

Váš počítač v sebe obsahuje paralelný interface osadený obvodom 8255. Popis obsadenia portov je uvedený v tretej časti príručky.

ČASŤ TRETIA : TECHNICKÝ POPIS

V tejto časti sú uvedené

- popis CPU
- popis zákazníckeho obvodu ULA1
- popis radiča floppy disku
- popis pamäti
- popis paralelného interface

3. Technický popis

Hlavné časti počítača DIDAKTIK KOMPAKT sú:

- centrálna procesorová jednotka CPU
- zákaznícky obvod ULA1
- pamäť ROM
- pamäť RAM
- radič floppy disku
- dekodér stavu zberníc
- mechanika floppy disku
- vstupno-výstupný blok obsahujúci:
  - klávesnicu
  - interface pre magnetofón
  - obvody VIDEO, VF a RGB
  - akustický výstup
  - paralelný interface
  - vstupy pre joystick
- napájací zdroj

3.1. Centrálna procesorová jednotka (CPU)

Predstavuje ju 8-bitový mikroprocesor Z80. Komunikáciu procesora s ostatnými časťami mikropočítača zabezpečujú 3 zbernice: adresná, dátová a riadiaca. Dátová zbernica je osembitová, obojsmerná a sprostredkúva prenos dát medzi jednotlivými časťami systému. Adresná zbernica je 16-bitová a umožňuje adresovať 64 kB pamäti. Riadiaca zbernica obsahuje signály vystupujúce z procesora (výstupné) ako aj vstupujúce do procesora (vstupné).

Výstupné riadiace signály sú:

$\overline{WR}$ - zápis

$\overline{RD}$ - čítanie

$\overline{IORQ}$ - komunikácia s vstupno-výstupným zariadením

$\overline{MERQ}$ - komunikácia s operačnou pamäťou

$\overline{BUSAC}$ - signalizuje odpojenie procesora od zberníc

Vstupné signály sú:

$\overline{INT}$ - požiadavka na prerušenie činnosti procesora

$\overline{NMI}$ - požiadavka na prerušenie činnosti procesora (nemaskovateľná)

$\overline{WAIT}$ - požiadavka na zastavenie činnosti procesora

$\overline{BUSRQ}$ - požiadavka na odpojenie procesora od zberníc

CPU pracuje s frekvencou 4MHz generovanou zákazníckym obvodom.

3.2. Zákaznícky obvod ULA1 (U-106-47)

Zákaznícky obvod vykonáva niekoľko funkcií:

- zabezpečuje cyklické čítanie pamäti video RAM a generuje video signály R,G,B,Y a synchronizačný signál S. Signály R,G,B sú generované frekvenciou 8 MHz, čo pri pravidelnom striedaní svetlých a tmavých bodov môže v systéme PAL spôsobiť výskyt farebných pruhov
- generuje signály pre činnosť pamätí RAM
- generuje periodický signál WAIT, pomocou ktorého dovoľuje CPU prístup do pamätí RAM len vtedy, keď zákaznícky obvod nečíta dáta pre zobrazovanie. Zákaznícky obvod ovláda prideľovanie dátovej zbernice a tým

zabezpečuje zdieľanie pamäti RAM. Tento spôsob prideľovania zberníc nezaručuje presné generovanie časových slučiek
- generuje taktovaciu frekvenciu 4MHz pre CPU
- generuje signál INT s periódou 20 ms - využíva sa pre obsluhu klávesnice
- obsahuje vstupno-výstupné porty pre ovládanie klávesnice, pre komunikáciu s magnetofónom a akustický výstup. Vzhľadom k odlišnej frekvencii od mikropočítača ZX Spectrum nemusia byť generované tóny úplne presné.

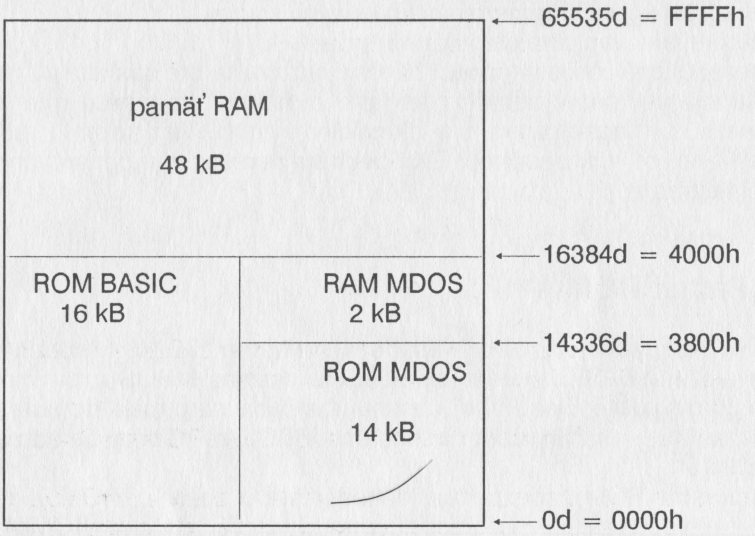
3.3. Pamäť ROM

ROM má kapacitu 32 kB a je osadená obvodom 27C256. V spodných 16 kB je uložený BASIC, v horných 14 kB je uložený MDOS a zvyšné 2 kB ostávajú nevyužité. Časť ROM obsahujúca Basic sa pripája do adresného priestoru 0h – 3FFFh, časť obsahujúca MDOS sa pripája do adresného priestoru 0h – 37FFh.

Prepínanie ROM zabezpečuje dekodér stavu zberníc. ROM je možné odpojiť privedením úrovne logickej jednotky na pin označený ROMDI vyvedený na systémový konektor.

3.4. Pamäť RAM

RAM je dynamická s kapacitou 64 kB osadená dvoma obvodmi s organizáciou 64kx4. Basic využíva 48 kB od adresy 4000h po FFFFh. MDOS využíva 2 kB RAM od adresy 3800h po 3FFFh a adresu 66h, na ktorej je uložená inštrukcia RST 0 pre SNAP.
Ostatná časť pamäti RAM nie je využitá.



3.5. Radič floppy disku

Radič zabezpečuje komunikáciu medzi počítačom a mechanikou floppy disku. Je osadený obvodom WD2797. Okrem príslušných registrov (adresy sú v prílohe) obsahuje separátor dát s fázovým závesom. Radič pracuje v režime dvojitej hustoty (MFM) s dvojstrannými disketami. Komunikácia radiča s CPU využíva NMI. DIDAKTIK KOMPAKT môže pracovať s dvoma mechanikami. Mechanika A (3,5") je vstavaná v počítači, disketovú jednotku B možno pripojiť cez konektor EXTENDED. Tá môže byť osadená mechanikou 3,5" alebo 5,25". Možno použiť disketové jednotky DIDAKTIK (D40, D40B, D80 alebo D80B). Popis konektora EXTENDED je uvedený v prílohe.

Ovládanie výberu mechaník, motorov a signálu NMI sa realizuje cez port na adrese −137d. Jednotlivé bity sú aktívne pri logickej úrovni 1 a majú nasledujúci význam:

D0 - DS0	výber mechaniky A
D1 - DS1	výber mechaniky B
D2 - MO0	zapnutie motora mechaniky A
D3 - MO1	zapnutie motora mechaniky B
D4 - NC	nevyužitý
D5 - NC	nevyužitý
D6 - DRQ	pripojenie DRQ radiča na NMI
D7 - INTRQ	pripojenie INTRQ radiča na NMI

3.6. Dekodér stavu zberníc

Zabezpečuje prepínanie ROM, t.j. prechody medzi časťou ROM BASIC a časťou ROM MDOS. Dekodér pracuje nasledovne:

- po prečítaní inštrukcie z adresy 00h alebo 08h sa ROM prepne do časti s MDOS
- po prečítaní inštrukcie z adresy 1700h sa ROM prepne do časti Basic
- po stlačení klávesu <SNAP>, ktorý vyvolá signál NMI a nasledovný skok na adresu 66h, dekodér zablokuje ROM a z RAM sa načíta inštrukcia RST 0. Toto nastane len vtedy, ak je aktívna ROM v časti Basic.

Dekodér je realizovaný programovateľnými obvodmi typu GAL.

3.7. Klávesnica

Je zapojená do matice 8x5 na horných 8 bitov adresnej zbernice a na 5 dátových bitov zákazníckeho obvodu. Klávesnica je osadená 45 klávesami.

4 klávesy (šípky) ovládajú logický obvod, ktorý simuluje postupné stlačenie klávesu <CAPS SHIFT> a príslušného klávesu 5, 6, 7 a 8. Kláves <RESET> je zapojený tak, že generuje signál RESET len pri súčasnom stlačení s klávesom <CAPS SHIFT>. Signál SNAP sa generuje súčasným stlačením klávesov <CAPS SHIFT> + <šípka vľavo> + <šípka vpravo>.

3.8. TV , VIDEO a RGB

DIDAKTIK KOMPAKT umožňuje pripojenie TV prijímača, prípadne monitora rôznymi spôsobmi. K tomuto účelu sú na zadnom čele počítača tri konektory: TV, MONITOR a AV – RGB.

Na **konektor TV** je privedený VF signál, určený pre pripojenie k anténnemu vstupu TV-prijímača. Kábel na prepojenie je súčasťou dodávky počítača. Signál je v pásme UHF.

Na **konektor MONITOR** je privedený videosignál pre pripojenie monitora (farebného aj monochromatického) alebo TV- prijímača, vybaveného vstupom pre video. Videosignál je v norme PAL, úroveň videosignálu je 1V. Tento výstup poskytuje vyššiu kvalitu obrazu než pripojenie cez anténny vstup TV prijímača.

Konektor AV – RGB obsahuje signály jednotlivých farebných zložiek R-red (červená), G-green (zelená), B-blue (modrá), signál video (využíva sa z neho synchronizačný signál) a zvukový signál z reproduktora, prípadne zo zvukového interface MELODIK. K tomuto výstupu je možné pripojiť TV-prijímač alebo monitor s RGB vstupom. **Tento spôsob pripojenia poskytuje najkvalitnejší obraz.** Úroveň výstupných signálov R, G, B a VIDEO je 1V. Na tento výstup nie je možné pripojiť monitory so vstupnou úrovňou TTL. Na konektor AV – RGB je vyvedené ešte napätie + 12V a signál pre odpojenie reproduktora v počítači (uzemnením pinu č. 5).
Rozmiestnenie konektorov a signálov je v prílohe.

3.9. Magnetofón

Pripojenie počítača k magnetofónu sa uskutočňuje štandardným prepojovacím káblom (nie je súčasťou dodávky) cez konektor MG.
Minimálne vstupné napätie z magnetofónu do počítača je 200 mV. Minimálne výstupné napätie z počítača do magnetofónu je 0,5 V. Popis konektora je uvedený v prílohe.

3.10. Konektory pre pripojenie joystickov

V počítači sú na zadnom čele dva 9-pinové konektory CANON, do ktorých je možné priamo zapojiť dva joysticky, jeden typu Sinclair joystick B (right) a druhý typu Kempston.

Konektor pre Sinclair joystick je paralelne pripojený k časti klávesnice vyberanej adresou A12 (klávesy 6 až 0). Pripojenie Sinclair joysticka je v nasledujúcej tabuľke, v zátvorkách sú uvedené klávesy na klávesnici, ktorých stlačenie vyvolá rovnakú činnosť, ako pohyb joystickom.

V prípade že ľubovoľný program má možnosť ovládania definovaním klávesov, stačí tieto definovať podľa popísaného pripojenia, a program je potom možné ovládať Sinclair joystickom.

Pripojenie Sinclair joysticku:

číslo vývodu	signál
1	od seba (8)
2	k sebe (9)
3	vľavo (6)
4	vpravo (7)
5	NC
6	akcia (0)
7	NC
8	spoločný vodič (A12)
9	NC

Pripojenie Kempston joysticku je v nasledujúcej tabuľke, v zátvorkách sú uvedené hodnoty načítané po pripojení Kempston joysticku z portu 31 inštrukciou IN. Na konektor KEMPSTON je možné pripojiť aj joystick s "autofire".

Pripojenie Kempston joysticku:

číslo vývodu	signál
1	od seba (8)
2	k sebe (4)
3	vľavo (2)
4	vpravo (1)
5	NC
6	akcia
7	+5V
8	spoločný vodič - GND
9	NC

Poznámka: KEMPSTON joystick je pripojený cez odpory na port A obvodu 8255. Z toho vyplývajú nasledujúce obmedzenia:
- KEMPSTON joystick nie je možné využívať, ak je port A nastavený ako výstupný
- KEMPSTON joystick nebude fungovať, pokiaľ je na port A zapojené iné periférne zariadenie (napr. tlačiareň), kde sú vstupy ošetrené odpormi s nízkymi hodnotami.

3.11. Systémový konektor

Systémový konektor sa nachádza na zadnej strane počítača. Sú na ňom vyvedené adresná, dátová, riadiaca zbernica a pomocné signály. Je riešený ako priamy konektor s 2 x 28 vývodmi s rozostupom 2,54 mm.

Vývody sú na obidvoch stranách počítačovej dosky. Vrchnú stranu dosky budeme nazývať **strana súčiastok** (strana A) a spodnú stranu dosky budeme nazývať **strana spojov** (strana B) (viď obr. v prílohe).

Konektor je odvodený od systémového konektora počítača ZX Spectrum, má však niektoré odlišnosti. Vzhľadom k inému napájaniu nie sú na konektor vyvedené napätia −5V, +9V, +12V ako aj signály U, V, Y a video.
Naviac je vyvedený signál $\overline{\text{ROMCS}}$, umožňujúci jednoduché dekódovanie vonkajšej pamäti ROM a signál SOUND, ktorým možno previesť zvukový signál z interfacea MELODIK na konektor RGB-pin audio.

3.12. Paralelný interface

Paralelný interface je realizovaný obvodom 8255. Výstupy z interface sú vyvedené na priamy konektor umiestnený na zadnej strane počítača. Priamy konektor je identický s priamym konektorom INTERFACE M/P. Popis konektora je v prílohe.
Porty obvodu 8255 sú umiestnené na nasledujúcich adresách:

- adresa 31d (1Fh) - port A
- adresa 63d (3Fh) - port B
- adresa 95d (5Fh) - port C
- adresa 127d (7Fh) - riadiace slovo

Z uvedeného vyplýva, že pre DIDAKTIK KOMPAKT je možné využiť prepojovacie káble k interface určené pre DIDAKTIK M a disketovú jednotku D40, resp. D80.

Názov: Didaktik Kompakt - Príručka užívateľa
Autori : RNDr. Roman Iška, RNDr. František Palkovič
Vydal : DIDAKTIK v. d.
Pod Kalváriou 22,
909 01 SKALICA
Tlač: DIDAKTIK v. d. SKALICA
Rozsah: 217 strán
Vydanie: prvé, október 1992

© 1992 DIDAKTIK SKALICA

DIDAKTIK