sinclair

ZX SPECTRUM

PROGRAMOVÁNÍ V BASICU

S Y N C L A I R ZX S P E C T R U M - PROGRAMOVÁNÍ V BASICU

Napsal: STEVEN VICKERS
Editoval: ROBIN BRADBEER
DRUHÉ VYDÁNÍ 1983

O B S A H

PŘÍLOHY

KAPITOLA 1 - ÚVOD

I když jste četli úvodní knihu nebo přicházíte přímo sem, měli byste si uvědomit, že příkazy se provádějí přímo a instrukce, které začínají číslem řádku, jsou uchovány pro pozdější využití. Měli byste si rovněž uvědomit příkazy PRINT, LET, /které jsou užity u všech počítačů užívající BASIC/ a BORDER, PAPER a BEEP /které jsou užity u Spectra/.

Tento manuál BASICU začíná opakováním některých důležitých věcí z úvodní knížky, ale mnohem podrobněji, přesně Vám říká, co můžete a co ne. Na konci každé kapitoly naleznete cvičení. Nepodceňujte je, mnohé z nich výstižně ilustrují body z textu. Projděte si je a udělejte co Vás zajímá nebo to čemu jste dobře nerozuměli. Ať děláte cokoliv, používejte vždy počítač. Pokud si kladete otázku co se stane, když natypujete to a to, bez obav typujte a odpověď si přečtete na obrazovce. Kdykoliv Vám manuál něco říká, že je třeba něco natypovat, vždy si řekněte, co byste mohli natypovat místo toho a pokuste se o vlastní náhradu. Čím více vlastních programů napíšete, tím více porozumíte počítači. Na konci tohoto programového manuálu jsou přílohy zahrnující sekci, která se dotýká organizace paměti, jak počítač manipuluje s čísly a řadu příkazů ilustrující schopnosti ZX Spectra.

K L Á V E S N I C E

Znaky ZX Spectra neobsahují pouze prosté symboly (písmena, číslice, atd.), ale taky složeniny (klíčová slova, jména funkcí ...) a tyto všechny se napíší stisknutím 1. klávesy. Aby se získaly všechny tyto funkce a příkazy, mají některé klíče až pět nebo i více funkcí, které se získají stlačením tlačítek (tj. stlačením současně CAPS SHOFT nebo SYMBOL SHIFT v době, kdy se požaduje daná funkce) a částečně různými mody počítače. Mod je ukazován kurzorem, je to svítící znak, ukazující, kde bude vložen další znak z klávesnice.

Mod K (pro klíčová slova) se automaticky nastaví L modem, když počítač očekává příkaz nebo řádek programu (spíše než INPUT) a z pozice na řádku ví, že by měl očekávat číslo řádku nebo po THEN i po : (mimo řetěz znaků). Pokud není stlačen SHIFT, bude další klíč jako klíčové slovo (psané na klapce) nebo číslo. L mod se normálně objeví ve všech ostatních časech. Pokud není stlačen SHIFT, bude další tlačítko interpretováno jako hlavní symbol tlačítka, jen v malém množství případů jako písmena. V K a L modu bude SYMBOL SHIFT a tlačítko interpretováno jako červený znak na tlačítku a CAPS SHIFT a číslicové tlačítko bude vzato jako kontrolní funkce napsaná bíle nad tlačítkem CAPS SHIFT a jiné tlačítko nemá účinek klíčovych slov v K modu a v L modu se mění malá písmena na velká.

C mod (pro velká písmena) je užit k získání dalších znaků, zvláštních symbolů. Objeví se, když jsou společně stlačeny obě tlačítka se SHIFT a trvá, když je drženo 1. tlačítko v tomto modu, písmeno dává 1. znak nebo symbol (vyznačen zeleně nad tlačítkem), pokud není SHIFT a další (zobrazeno červeně pod) pokud je stlačeno současně se SHIFTEM. Číslicové tlačítko dává symbol, když je stlačeno současně s SYMBOL SHIFT, jinak dává kontrolu posloupností.

G (grafika) mod se objeví po GRAPHICS (CAPS SHIFT a 9) stlačení a trvá do dalšího stlačení nebo pokud není stlačena 9. Číslicový klíč dává grafickou mozaiku a každý ze znaků kromě V, W, X, Y, Z má grafiku definovanou uživatelem. Pokud je kterékoliv tlačítko drženo déle než 2 - 3 vteřiny, opakuje se.

Vstup z klávesnice se během typování objeví v dolní části obrazovky, každý znak (jednoduchý nebo symbol) se vkládá před kurzor. Kurzorem můžeme pohybovat vlevo (CAPS SHIFT a 5) nebo vpravo (CAPS SHIFT a 8). Znak před kurzorem může být zrušen DELETE (CAPS SHIFT a Ø).

POZNÁMKA: celý řádek může být zrušek stlačením EDIT (CAPS SHIFT a 1) a pak následným ENTER.

Když je stlačeno ENTER, řádek je vyvolán, vstoupí do programu, nebo je využit jako vstup, pokud obsahuje symbolickou chybu, objeví se svítící ? vedle chyby. Jakmile se vstoupí do programu, výpis je zobrazován ve vrcholu obrazovky. Poslední vložený řádek se nazývá běžný řádek a je označen symbolem > , ten může být posunut použitím tlačítek ⇩ (CAPS SHIFT a 6) a ⇧ (CAPS SHIFT a 7). Jakmile je stlačeno EDIT (CAPS SHIFT a 1), běžný řádek je shozen dolů, kde může být editován. Provádí-li se příkaz nebo běží-li program, vstup je zobrazen v horní části obrazovky a zůstává tam, až se vstoupí do programu nebo je stlačeno ENTER

a prázdný řádek nebo ⇩ (⇧). Ve spodní části se objeví zpráva, daná kodem (číslo nebo písmeno) - popsánov dodatku B. Zpráva zůstává na obrazovce až do doby stlačení dalšího tlačítka (idikuje K mod). V některých případech je CAPS SHIFT a SPACE ve stejné funkci jako BREAK, zastavuje počítač se zprávou D nebo L. To je 1. na konci řádku, probíhá-li program,
2. když počítač užívá magnetofon nebo tiskárnu.

TELEVIZNÍ OBRAZOVKA

Má 24 řádků, každý je dlouhý 32 znaků a je rozdělen na 2 části. Horní část 22 řádků zobrazuje výpis programu nebo výstup. Během tisku horní části se dojde na konec a počítač se zastaví se zprávou SCROLL? Po stlačení tlačítka N, SPACE nebo STOP bude program zastaven zprávou D - BREAK - CONT, každé další jiné tlačítko způsobí "scrolling".

Spodní část je využita pro vstup příkazů, řádků programu a vstup dat a taky pro tisk zpráv. Spodní část se skládá ze dvou řádků, ale při typování je rozšířena. Když se dosáhne pozice tisku v horní polovině, provede se vymazání poloviny horní řádky.

KAPITOLA 2 - KONCEPT PROGRAMOVÁNÍ V BASICU

SHRNUTÍ
Programy
Čísla řádků
Editování programů při použití ⇧ , ⇩ a EDIT
RUN, LIST
GO TO, CONTINUE, INPUT, NEW, REM, PRINT
STOP v data INPUT
BREAK

Natypujte tyto 2 řádky programu pro počítač, aby vytiskl součet dvou čísel.

```
20 PRINT a
10 LET a=10
```

a obrazovka vypadá takto:

```
10 > LET a=10
20   PRINT a
```

K

Jak jistě víte, tyto řádky začaly čísly a nebyly proto ihned provedeny, ale uchovány, jako řádky programu. Všimněte si, že čísla řádků řídí pořádek v programu, je to důležité pro jeho běh a taky se to odráží v pořadí řádků, které nyní můžete vidět na obrazovce. Nyní pouze typujte příkaz:

```
15 LET b=15
```

Pokud by byly řádky číslovány bez mezer, nebylo by možné mezi dva napsané řádky již vložit další řádek, proto je výhodné typovat místo 1, 2, ... 10, 20, ... avšak maximálně 9999. To je důvod proč typovat po 10. Nyní potřebujeme změnit řádek 20 na

```
20 PRINT a+b
```

Mohli byste typovat vše znovu, ale je snadnější využít možnosti EDIT jak bylo popsáno v úvodu. Šipka > u řádku 15 se nazývá programový kurzor a řádek, na který ukazuje se nazývá běžný řádek. Je to obvykle poslední řádek, který byl vkládán, ale můžete použít ⇩ nebo ⇧ ke změně programového kurzoru dolů nebo nahoru (zkuste a nechte programový kurzor na řádku 20). Když mačkáte EDIT, objeví se kopie běžného řádku ve spodní části obrazovky - ve vašem případě je to kopie řádku 20. Držte tlačítko⇨ tak dlouho, až se L kurzor posune na konec řádku a pak typujte + b (bez ENTER). Řádek na konci by měl nyní ukazovat:

```
20 PRINT a + b
```

Po zmáčknutí ENTER se nahradí starý řádek 20 a obrazovka vypadá:

```
10 LET a = 10
15 LET b = 15
20> PRINT a + b
```

K

Nechte provést program příkazu RUN a ENTER. Zobrazí se součet. Nechte běžet program znovu a natypujte:

```
PRINT a + b
```

Proměnné jsou zde stále, i když program skončil. Existuje užitečná metoda použitím EDIT k získání spodní části obrazovky. Něco natypujte (bez ENTER) a pak rozhodněte, že to nechcete. Jedna z cest, jak to provést je držet tlačítko DELETE až je řádek zrušen. Existuje i následující možnost. Pokud stlačíte EDIT, text ve spodní části obrazovky bude nahrazen kopií běžného řádku. Pokud nyní stlačíte ENTER, běžný řádek bude vzán zpět do programu bez změny a nechá spodní část obrazovky volnou. Pokud vložíte řádek s chybou, např.

```
12 LET b =8
```

vejde tento řádek do programu a až pak si uvědomíte chybu. K vynechání tohoto řádku typujte:

```
12    (a ENTER)
```

Nyní si s údivem všimnete, že programový kurzor zmizel. Můžete si představit, že je někde mezi 1Ø. a 15. řádkem. Pokud stlačíte ⇧, bude se nacházet na řádku 1Ø, pokud stlačíte ⇩, bude na řádku 15. Typujte:

```
12    (a ENTER)
```

Znovu bude programový kurzor mezi řádky 1Ø a 15. Nyní stlačte EDIT a dolů bude kopie řádku 15. Když je programový kurzor mezi řádky pak EDIT dává dolů řádek s vyšším číslem. Natypujte ENTER, k vyčištění spodní části obrazovky. Pak typujte:

```
3Ø    (a ENTER)
```

Nyní je programový kurzor za koncem programu. Pokud stlačíte EDIT, bude dolů shozen řádek 2Ø. Konečně typujte:

```
LIST 15      a na obrazovce vidíte:
15 > LET b=15
2Ø   PRINT a+b
```

Řádek 1Ø není zobrazen, ale je stále ve vašem programu. Můžete si to ověřit stlačením ENTER. Pouhým výsledkem příkazu LIST 15 je, že se provede výpis programu začínající řádkou 15 a dále je na této řádce programový kurzor. Pokud máte velmi dlouhý program, pak LIST bude mnohem užitečnější způsob posunu programového kurzoru než ⇧ a ⇩ . Ilustruje to další použití řádků programu. Na čísla řádků se můžeme odkazovat v programu podobně jako na jména proměnných. LIST bez čísla provede výpis programu od jeho počátku. Další příkaz, který byl v úvodní knize, je NEW. Ten ruší jakýkoliv starý program a proměnné v počítači. Nyní pečlivě natypujte tento program ke přepočtu FAHRENFEITOVÝCH stupňů na CELSIOVY.

```
1Ø REM temperature conversion
2Ø PRINT "deg F", "deg C"
3Ø PRINT
4Ø INPUT "ENTER deg F", F
5Ø PRINT F, (F-32)*5/9
6Ø GO TO 4Ø
```

Natypujte text na řádce 1Ø. Ačkoliv GO TO má mezeru jde o jedno klíčové slovo (na G). Nyní spusťte program. Uvidíte hlavičku, ticknutou na obrazovce příkazem 2Ø. Ale k čemu je řádek 1Ø? Počítač ho úplně ignoroval. REM v řádku 1Ø je poznámka nebo připomínka a slouží k zapamatování, k čemu je vlastně program. Příkaz REM se skládá z REM a dále následuje cokoliv – počítač to bude ignorovat až ke konci řádku. Nyní se počítač dostal na řádek 4Ø, příkaz INPUT a čeká až natypujete hodnotu proměnné F; je to proto, že kurzor se změnil v K místo L. Vložte číslo a stlačte ENTER. Počítač nyní zobrazí výsledek a čeká další číslo. Je to dáno řádkem 6Ø (GO TO 4Ø). Místo toho by se počítač zastavil. Vložte další teplotu. Zopakujete-li několikrát cyklus, můžete se dívat, zda počítač neukončí výpočet, ale není tomu tak. Program lze ukončit natypováním STOP, když se počítač ptá na další číslo. Objeví se zpráva H STOP in INPUT in line 4Ø:1, což říká, proč se zastavil běh programu a kde (první příkaz na řádku 4Ø). Pokud chcete pokračovat ve výpočtu typujte CONTINUE a počítač se zeptá na další číslo. Použijeme-li CONTINUE počítač si zamatuje číslo řádku poslední zprávy, kterou vám poslal a která zakončila Ø OK a vrací se zpět na tuto řádku. V našem případě to vyvolá skok na řádek 4Ø, tj. na příkaz INPUT. Nahraďte řádek 6Ø GO TO 31 nebudete pozorovat zmíny

rozdíl v běhu programu. Pokud číslo řádku odpovídá neexistujícímu číslu, provede se skok na číslo nejbližší vyšší. Totéž provádí příkaz RUN (ve skutečnosti odpovídá RUN Ø). Nyní typujte čísla, až se obrazovka úplně zaplní. Když je plná, počítač přetypuje celou horní část o jeden řádek, aby se uvolnil řádek pro již provedený výpočet a horní řádek se ztratí. Nazývá se to "scrolling". Pokud vás to unavilo, zastavte program pomocí příkazu STOP a získáte výpis programu pomocí ENTER. Podívejte se na příkaz PRINT na řádku 5Ø. Znaménko (,) je zde velmi důležité a měli byste pamatovat, že má mnohem větší význam a mnohem přísnější pravidla než znaménka v angličtině. Čárky se užívá, chceme-li uskutečnit začátek tisku na začátku nebo uprostřed obrazovky, podle textu. Tak čárka v řádku 5Ø způsobí, že teplota ve stupních celsia se tiskne uprostřed řádku. Při použití středníku bude další číslo nebo řetězec tištěn ihned po předcházejícím. Můžete to vyzkoušet na řádku 5Ø, pokud je čárka nahražena středníkem. Další znak, který můžete v příkazu PRINT použít je apostrof. Ten zařídí, aby další tisk byl proveden na počátku dalšího řádku. Totéž provádí i příkaz PRINT, takže se apostrofu moc nepoužívá. Příkaz PRINT na 3Ø. řádku produkuje tedy vynechání jednoho prázdného řádku. Abychom viděli, jak to funguje, nahraďte řádek 5Ø takto:

```
5Ø PRINT F,
5Ø PRINT F;
5Ø PRINT F'
5Ø PRINT F
```

Příkaz s čárkou dává tisk do dvou sloupců, příkaz se středníkem dává vše dohromady, příkaz s apostrofem dává nový řádek. Pamatujte na rozdíl mezi čárkou a středníkem v příkazu PRINT, taky to nesplete s dvojtečkou, která je užita jako oddělení příkazů na jednom řádku. Nyní typujte tyto zvláštní řádky:

```
1ØØ REM tento zdvořilý program si pamatuje vaše jméno
11Ø INPUT n$
12Ø PRINT "Hello";n$;"!"
13Ø GO TO 11Ø
```

Toto je samostatný program, ale je možné ho držet v paměti počítače společně s předešlým programem. Spusťte tento nový program příkazem RUN 1ØØ. Protože vstup do programu je řetězec místo čísel, tisknou se dvojí uvozovky - to vám připomíná a šetří typování. Zkuste to s jakýmkoliv jménem. Nyní dostanete opět řetězec se 2. "", ale nemusíte je užít, pokud nechcete. Zkuste tento příklad: Vymažte uvozovky (pomocí DELETE) a natypujte n$. Protože zde nejsou "" počítač ví, že má udělat stejnou kalkulaci. Kalkulace v tomto případě má najít hodnotu řetězové proměnné n$, což je jméno, typované v minulém zpracování. Příkaz INPUT se tedy ve skutečnosti chová jako LET n$=n$ a tak hodnota n$ není změněna. V dalším kole pro porovnání typujte n$, ale bez vymazání uvozovek. Nyní vás poplete, že proměnná n$ má hodnotu "n$". Pokud chcete použít STOP pro vstup řetězce, musíte prvně posunout kurzor zpět na počátek řádku s použitím ⇦.

Nyní se podívejte zpět na příkaz RUN 1ØØ. Tento zajistí skok na řádek 1ØØ a mohli bychom říci, že je stejný s GO TO 1ØØ. V tomto případě je tomu skutečně tak, ale existuje zde rozdíl. RUN 1ØØ prvně vynuluje proměnné a obrazovku a pak pracuje jako GO TO 1ØØ. GO TO 1ØØ však nic nemění. Mohou nastat příležitosti, kdy chcete, aby program probíhal bez změny proměnných, zde je nezbytné použít GO TO, protože RUN by byl katastrofální. Další rozdíl je v tom, že RUN může typovat bez čísla řádku a běh programu začně na řádku č. 1. daného programu. GO TO musí mít vždy udán skok cílové adresy. Občas chybně napíšete program, který pak nemůžete zastavit a sám se taky nezastaví. Typujte:

```
2ØØ GO TO 2ØØ     a     RUN 2ØØ
```

Pokud vypnete proud, bude program neustále probíhat, ale existuje méně drastické řešení. Stlačte CAPS SHIFT spolu s SPACE. Program se zastaví s hlášením L BREAK. Program neustále sledujete, zda-li nejsou zmáčknuta tato tlačítka, pokud jsou, zastavuje program. Tlačítko BREAK lze taky použít při již započatém nahrávání na magnetofon nebo při tisku. CONTINUE pak zajistí další pokračování v práci po BREAK. Nechte znovu běžet program a když se zeptá na vstup, typujte:

```
n$   (po předešlém vymazání uvozovek)
n$     je zde nedefinovaná proměnná a vy obdržíte zprávu:
```

2 Variable not found (proměnná nenalezena). Pokud nyní natypujete
LET n$="něco určitého" (což má svou vlastní zprávu Ø OK, Ø:1)a CONTINUE shledáte, že můžete použít n$ jako vstup dat bez problémů. V tomto případě udělá CONTINUE skok na příkaz INPUT na řádku 11Ø. Protože zpráva příkazu LET byla OK, provede se příkaz na řádce 11Ø. To je jedna z možností použití. Pokud se program pro chybu zastaví, pak můžete udělat hodně věcí, aby se program upravil a další pak zajistí CONTINUE.
Jak bylo řešeno dříve, zpráva L BREAK jako program je speciální, protože CONTINUE pak zajistí opakování příkazu, kde se program zastavil.
Automatický listing (ten, který není výsledkem příkazu LIST) vás může poplést. Pokud natypujete program o 5Ø řádcích - samé REM

```
1 REM
2 REM
  .
  .
5Ø REM      můžete experimentovat.
```

První věc, kterou si pamatujte, že běžný řádek se bude vždy objevovat na obrazovce obvykle ve středu. Typuj:

```
LIST    (a ENTER)
```

Když se objeví scroll? (protože se zaplnila obrazovka), stlačte n (jako ne). Počítač odpoví "D BREAK - CONT repeats" jko byste natypovali BREAK. Mohlo by vás zajímat co se stane zmáčknete-li X místo n. Nyní stlačte ENTER a dostanete automatický listing, měli byste vidět řádky 1 až 22. Nyní typujte:

```
23 REM   dostanete na obrazovku řádky 2 až 23. Typujte nyní
28 REM   dostanete řádky 7 až 28.
```

V obou případech jste po natypování nového řádku pohli programovým kurzorem a tak jste uskutečnili nový listing. Snad to vypadá sporně. Počítač skutečně chce, aby vám dal,co žádáte. Počítač neuchovává pouze záznam o běžném řádku, ale také o horním řádku na obrazovce. Když dělá výpis nejdříve porovná horní řádek s řádkem běžným. Pokud příjde horní řádek později, pak zde není instrukce, aby se tam zastavil a tak použije běžného řádku pro nový počátek a provede výpis. Jinak je běžná metoda, že se provádí výpis od prvního po běžný řádek a když je to nezbytné, provede se scrolling. To vše dává široký přehled o délce a pokud je odpověď příliš dlouhá, posune horní řádek dolů, blíže k řádku běžnému. Nyní začíná pracovat od této horní řádky. Když dojde na konec programu nebo zaplní zcela obrazovku a běžný řádek byl zobrazen, zastaví se. Jinak se provádí scrolling, až je běžný řádek zobrazen a pro každý další řádek porovná horní řádek tak, že se přibližuje běžnému řádku. Experimentujte s posunem běžného řádku typováním.

```
číslo řádku  REM
```

LIST posouvá běžný řádek, ale na horní, takže se posloupnost výpisu může lišit. Např. typujte LIST, abyste získali výpis od řádku Ø a stlačte ENTER. Na obrazovce by se měly objevit řádky 1 až 22. Typujte:

```
LIST 22
```

což dává řádky 22 až 43. Když stlačíte znovu ENTER dostanete znovu řádky 1 - 22. Ukazuje to, že se lépe pracuje s programy krátkými než s delšími. Použijte program, který se skládá pouze z REM a typujte

```
LIST
```

Počítač se na konci obrazovky zastaví se zprávou "scroll?", dejte n. Typujte:

```
CONTINUE
```

CONTINUE je zde trochu zvláštní, protože horní část obrazovky je blenková, ale vy můžete obnovit řádek typováním BREAK. Výsledkem je, že LIST byl prvním příkazem na řádku a CONTINUE opakuje tento příkaz. Naneštěstí první příkaz na řádku je nyní CONTINUE a tak počítač zůstává tam kde byl a provádí CONTINUE tak dlouho, až ho zastavíte. Můžete typovat tuto náhradu LIST s

```
: LIST
```

pro který CONTINUE dává zprávu Ø OK (protože CONTINUE skáče na druhý příkaz v řádku, který je brán jako konec) nebo

```
::LIST
```

pro který CONTINUE dává "N Statement last" (protože CONTINUE skáče na třetí příkaz, který neexistuje).

Viděli jste použití příkazů PRINT, LET, INPUT, RUN, LIST, GO TO, CONTINUE, NEW a REM. Všechny mohly být použity v programu nebo jako příkazy - to platí téměř o všech příkazech ZX Spectra BASIC. RUN, LIST, CONTINUE a NEW nejsou často používány v programech, ale smí být užity.

Cvičení:

1. Dejte příkaz LIST do programu. Při chodu pak dochází k listování.
2. Napište program se vstupem cen a tiskem daně (např. 15 %). Příkazem PRINT vyzvěte kněte oznámení, co se provádí a ptejte se na ceny zdvořile. Modifikujte program tak, že můžete zadávat rovněž velikost daně.
3. Napište program k tisku celkového součtu čísel na vstupu (rada: máte dvě proměnné nazvané TOTAL, začínající Ø a položka. Vstupuje položka, přičtěte ji k totalu a obě vytiskněte a pokračujte dále).
4. Co by znamenaly příkazy v CONTINUE a NEW v programu? Znáte v tomto případě jejich užití?

KAPITOLA 3 - ROZHODOVÁNÍ

SHRNUTÍ: IF, STOP
=,<,>,<=,>=,<>

Všechny programy, které jste doposud viděli, byly snadno určeny - procházely přímo po instrukcích a pak se vracely zpět na počátek. Je to velmi nepraktické. V praxi má počítač očekávat, že bude sám provádět rozhodování a podle toho jednat. Použitá instrukce má formu ... IF něco je pravda nebo není pravda THEN dělej něco. Např. užitím NEW vymažte předšlý program a natypujte a spusťte tento program (je to hra - dvě osoby).

```
1Ø REM  hádání čísla
2Ø INPUT a:CLS
3Ø INPUT "hádej číslo", b
4Ø IF b=a THEN PRINT "to je správné": STOP
5Ø IF b<a THEN PRINT "je příliš malé, zkuste znovu"
6Ø IF b>a THEN PRINT "je příliš velké, zkuste znovu"
7Ø GO TO 3Ø
```

Je vidět, že příkaz IF se skládá z IF podmínka THEN .... , kde .... znamená sekvenci příkazů, oddělených dvojtečkami v obvyklé formě. Vyhodnocení testu bude pravdivé nebo nepravdivé. Když je pravdivé, tak se příkazy ve zbytku řádku po THEN provádí, ale když je nepravdivé, tak se přeskočí a program začíná na další instrukci. Nejjednoduchší podmínky porovnávají dvě čísla nebo dva řetězce, mohou testovat, zda jsou si čísla rovny nebo zda je jedno větší než druhé a mohou testovat, zda dva řetězce jsou rovny nebo si alfabeticky předcházejí. Používají se vztahy =,<,>,<=,>=,<>

= znamená rovno - I když symbol je stejný jako v příkazu LET, má zde zcela jiný význam.

< (SYMBOL SHIFT a R) znamená je menší než
1< 2, -2 < -1, -3 < 1 jsou pravdivá, ale
1< 0, 0< -2 jsou nepravdivá
řádek 4Ø porovnává a s b. Pokud platí rovnost, program se zastaví po STOP. Zpráva dole na obrazovce bude: 9 STOP, statement, 3Ø:3 ukazuje, že třetí příkaz na rádce 3Ø způsobil zastavení programu, t.j. STOP. Řádek 5Ø určuje, když b je menší než a, řádek 6Ø určuje, když b je větší než a. Je-li jedna z těchto podmínek pravdivá, tiskne se příslušný komentář a program pokračuje na řádce 7Ø, která říká počítači, aby se vrátil na řádek 3Ø a začal vše znovu. CLS - čistí obrazovku, příkaz ve 2Ø řádku měl jiné osobě zabránit přečíst informace vstupující do počítače.

> (SYMBOL SHIFT a T) znamená je větší než
a znamená je jako < , ale smysl je obrácený. Symboly se dají lehce pamatovat, prože menší symbol je označen koncovkou.

<= (SYMBOL SHIFT a Q) - natypujete jako < a následně =; znamená menší než nebo rovno, tzn., že je stejné jako s výjimkou toho, že relace je pravdivá i tehdy, když jsou si čísla rovny. Tak 2<=2 je pravdivá, ale 2 2 je nepravdivá.

>= (SYMBOL SHIFT a E) - znamená je větší nebo rovno a je podobný jako>.

<> (SYMBOL SHIFT a W) - znamená nerovno a stejí jako protiklad =. Matematici píší obvykle ≤ , ≥ , ≠. Také píší výrazy jako 2<3<4, což znamená 2<3 a 3<4, ale v BASICU to není možné.
Poznámka: V jiných verzích jazyka BASIC může mít podmínka IF tvar
IF podmínka THEN číslo řádku. U ZX Spectra píšeme obdobně:
IF podmínka THEN GO TO číslo řádku.

Cvičení:
1. Zkuste tento program:
10 PRINT "x":STOP:PRINT "y"
Když ho spustíte, bude zobrazeno x a zastaví se se zprávou:
9 STOP statement, 10:2. Nyní typujte
CONTINUE
Mohli byste očekávat skok zpět na příkaz STOP - CONTINUE obvykle opakuje řádek ve zprávě. Avšak zde by to nebylo užitečné, protože počítač by se mohl opět zastavit bez zobrazení y. Proto je zařízeno, že po zprávě 9 CONTINUE se jde na další příkaz po příkazu STOP - tak v našem případě po CONTINUE tiskne počítač y a program je ukončen.

KAPITOLA 4 - CYKLY

SHRNUTÍ: FOR, NEXT
TO, STEP

Předpokládejme, že chcete uložit pět čísel a sečíst je dohromady. Jedna z cest (nemusíte typovat, pokud se vám nezdá rozumné) je napsat

```
10 LET total=0
20 INPUT a
30 LET total=total+a
40 INPUT a
50 LET total=total+a
60 INPUT a
70 LET total=total+a
80 INPUT a
90 LET total=total+a
100 INPUT a
110 LET total=total+a
120 PRINT total
```

Tuto metodu není dobré praktikovat. Je pouze ukázková a uvědomte si jak by bylo nudné a zdlouhavé psát program, který sečte 10 čísel. Sečíst 100 čísel by bylo téměř nemožné . Mnohem efektnější je deklarovat počítač do 5. a pak zastavit program, jako v tomto příkladě:

```
10 LET total=0
20 LET count=1
30 INPUT a
40 REM count=je číslo udávající, kolikrát se má provést vstup
50 LET total=total+a
60 LET count=count+1
70 IF count<=5 THEN GO TO 30
80 PRINT total
```

Všimněte si, jak je snadné změnit řádek 70 tak, aby se sečetlo 10 nebo i 100 čísel. Tento druh počítání je velmi užitečný a tak existují dva speciální příkazy, které to usnadňují. Jsou to příkazy FOR a NEXT. Užívají se vždy společně! S použitím těchto příkazů pak program vypadá takto:

```
10 LET total=0
20 FOR c=1 TO 5
30 INPUT a
40 REM c = je číslo, kolikrát se má provést vstup
50 LET total=total+a
60 NEXT c
80 PRINT total
```

(k získání tohoto programu stačí editovat řádky 20, 40, 60, 70 předešlého programu).

Všimněte si, že jsme změnili count na c. Proměnná cyklu - nebo kontrolní proměnná příkazu FOR - NEXT musí mít 1 písmeno jako jméno. Efekt programu je, že c probíhá přes hodnoty 1 (inicializační), 2, 3, 4, 5 (limit) a pro každý řádek 3Ø, 4Ø a 5Ø se provádí příkazy. Až c nabude hodnotu 5, cyklus se ukončí a je proveden řádek 8Ø. Zvláštním případem je, že kontrolní proměnná nemusí jít po jedné. Můžete to změnit příkazem STEP v části příkazu FOR. Nejobvyklejší forma příkazu FOR je:

FOR kontrolní proměnná = inicializační hodnota TO limit STEP krok,

kde kontrolní proměnná má 1 písmeno a inicializační hodnota, limit a krok jsou výroky, které počítač umí vyhodnotit jako čísla - vlastní čísla (kódy), součty, jména numerických proměnných. Řádek 20 nahraďte:

2Ø FOR c=1 TO 5 STEP 3/2

hodnota c se bude pohybovat od 1 přes 2,5 a 4. Všimněte si, že se nemusíte omezit na celá čísla a taky, že kontrolní proměnná nemusí nabýt přesně limitu - cyklus probíhá tak dlouho dokud je hodnota kontrolní proměnné menší nebo rovna limitu. Zkuste program, který tiskne čísla od 1 do 1Ø v opačném pořadí:

```
1Ø FOR n=1Ø TO 1 STEP -1
2Ø PRINT n
3Ø NEXT n
```

Řekli jsmemi, že cyklus probíhá tak dlouho, až je hodnota kontrolní proměnné rovna nebo menší než limit. V tomto případě je však toto pravidlo nesmyslné. Pravidlo je modifikováno: pokud je krok záporný, program provádí smyčku tak dlouho, až je kontrolní proměnná větší nebo rovna limitu. Buďte opatrní provádíte-li dvě a více smyček FOR - NEXT najednou, zvláště jsou-li vzájemně do sebe vnořeny. Zkuste program tisknoucí čísla pro úplnou stavbu domina:

```
1Ø FOR m = Ø TO 6
2Ø FOR n = Ø TO m
3Ø PRINT m;";";n;" ";      } n-loop (cyklus)
4Ø NEXT n                                    } m-loop (cyklus)
5Ø PRINT
6Ø NEXT m
```

Jak vidíte, n-smyčka je celá uvnitř m-smyčky - jsou přesně odděleny. Musíte dát pozor, když máme dvě FOR -NEXT smyčky vzájemně se překrývající:

```
 5 REM tento program je chybný
1Ø FOR m = Ø TO 6
2Ø FOR n = Ø TO m          } m-cyklus
3Ø PRINT m;";";n;" ";                  } n-cyklus
4Ø NEXT m
5Ø PRINT
6Ø NEXT n
```

Dvě FOR - NEXT smyčky musí být buď jedna uvnitř druhé, nebo úplně zvlášť. Důležitá věc je vyhnout se skoku dovnitř FOR - NEXT cyklu z venku. Kontrolní proměnná je správně nastavena pouze tehdy, když je proveden FOR i NEXT, vynechá-li se jeden nebo druhý příkaz, počítač je zmaten. Chyba se pravděpodobně projeví hlášením NEXT without FOR nebo variable not found. (NEXT bez FOR nebo nenalezena proměnná). Zde je příklad užití FOR a NEXT v jednom příkazu:

FOR m = Ø TO 1Ø : PRINT m : NEXT m

Příklad můžete použít jako (přirozenou) cestu k získání omezení, kdy nemůžete použít GO-TO dovnitř příkazu - protože řádek nemá číslo.

FOR m = Ø TO 1 STEP Ø : INPUT a : PRINT a : NEXT m

Kde Ø zde působí, že se příkaz bude neustále opakovat. Tento druh programování se nedoporučuje, protože při chybě je příkaz ztracen a budete ho muset typovat znovu, příkaz CONTINUE zde nepomůže.

Cvičení:

1. Kontrolní proměnná nemá jméno a hodnotu jako obyčejná proměnná, ale taky limit, krok a odkaz na příkaz vztažený k FOR. Přesvědčte se, že když je použit příkaz FOR, jsou dostupny všechny informace (použijte inicializační hodnotu jako první hodnotu proměnné) a taky že tyto informace jsou dostatečné, aby příkaz NEXT poznal o kolik zvýšit hodnotu, zda skončit zpět a na kterou adresu.
2. Nechte běžet třetí program a pak typujte PRINT c. Proč je odpověď 6 a ne 5? (Odpověď: příkaz NEXT v řádce 6Ø se provádí 5x a pokaždé se přičte 1 k c.

Při posledním cyklu se c zvýší na 6 a příkaz NEXT rozhodne nevrace se, ale pokračovat na další řádce - c nabyla tímto limitní hodnoty.) Co se stane, když typujete v řádku 2Ø STEP 2?

3. Změňte třetí program tak, aby se počítač zeptal, kolik čísel chcete sečíst. Co se stane, když vložíte Ø, znamenající, že nechcete sčítat? Mohli byste očekávat, že to bude pro počítač problém, zda pochopí, co tím myslíte. (Počítač musí udělat výběr pro příkaz NEXT s, který není nezbytný. Na vše je ve skutečnosti pamatováno.
4. V řádku 1Ø programu 4 změňte 1Ø na 1ØØ a spusťte program. Vytiskne čísla od 1ØØ do 79 a pak se zeptá "scroll?" Dává tím možnost vidět čísla, která jsou nad místem, kde se bude provádět "scroll". Pakud zmáčknete n, STOP nebo BREAK, program se zastaví se zprávou D BREAK - CONT repeats. Když stlačíte kterékoliv jiné tlačítko, bude tištěno dalších 22 řádků s novým dotazem na konci.
5. Zrušte řádek 3Ø ve 4. programu. Bude-li běžet takto spravený program, vytiskne se první číslo a program se skončí se zprávou Ø OK. Když natypujete NEXT n
program udělá jednou cyklus a vytiskne se další číslo.

KAPITOLA 5 - PROCEDURY

SHRNUTÍ: GO SUB, RETURN

Některé části programu mají podobné funkce a vy je musíte znovu typovat, dvakrát nebo i více, ale není to nezbytné. Můžete typovat řádky jednou ve formě podprogramů a při užití (celku) pouze provést instrukce uvnitř podprogramu. Slouží k tomu dva příkazy: GO SUB (GO to SUB routine) a RETURN. Mají formu:

GO SUB n

kde n je číslo prvního řádku podprogramu. Je to stejné jako užití GO TO n s výjimkou, že si počítač, kde je příkaz GO SUB a po provedení podprogramu se vrací zpět. Pamatuje

RETURN

dává zpět návratovou adresu GO SUB a provádí další příkaz. Jako příklad je uveden program, který hádá čísla:

```
1Ø REM    upravená hra
2Ø INPUT a : CLS
3Ø INPUT "Guess the number",b
4Ø IF a = b THEN PRINT "CORECT  (správně)": STOP
5Ø IF a<b THEN GO SUB 1ØØ
6Ø IF a>b THEN GO SUB 1ØØ
7Ø GO TO 3Ø
1ØØ PRINT  "Zkus znovu"
11Ø RETURN
```

Příkaz GO TO v řádce 7Ø je velmi důležitý, protože jinak program půjde do procedury a způsobí chybu (7 RETURN without GO SUB), když se dojde na příkaz RETURN. Zde je další program užívající příkaz GO SUB:

```
1ØØ LET x = 1Ø
11Ø GO SUB 5ØØ
12Ø PRINT s
13Ø LET x = x + 4
14Ø GO SUB 5ØØ
15Ø PRINT s
16Ø LET x = x + 2
17Ø GO SUB 5ØØ
18Ø PRINT s
19Ø STOP
5ØØ LET s = Ø
51Ø FOR y = 1 TO x
52Ø LET s = s + y
53Ø NEXT y
54Ø RETURN
```

Poběží-li program, podívejte se, zda s ním umíte pracovat. Podprogram začíná v řádce 5ØØ. Podprogram může úspěšně volat další procedury i sebe(ta se nazývá rukarsivní). Nebojte se několika úrovní!

KAPITOLA 6 - READ, DATA, RESTORE

SHRNUTÍ: READ, DATA, RESTORE

V některých předchozích programech jsme mohli vidět, že informace nebo data vstupovala přímo do programu použitím příkazu INPUT. Obvykle je to únavný způsob, zvláště vstupuje-li do programu hodně dat. Hodně času můžete ušetřit použitím příkazů READ, DATA, RESTORE.

```
1Ø READ a, b, c
2Ø PRINT a, b, c
3Ø DATA, 1Ø, 2Ø, 3Ø
4Ø STOP
```

Řádek READ se skládá z READ, za kterým následuje seznam jmen proměnných, oddělených čárkami. Příkaz pracuje podobně jako INPUT s tím rozdílem, že místo typování jmen proměnných počítač hledá hodnoty v příkazu DATA. Každý příkaz DATA je seznam vyjádření - numerické nebo řetězové proměnné, oddělené čárkami. Můžete je umístit kamkoliv v programu - počítač je ignoruje, kromě případu, kdy je použit příkaz READ. Když počítač poprvé provádí READ, vezme první vyjádření ze seznamu DATA. Příště bere druhé atd., až je vyčerpán příkaz READ. Příkaz READ pracuje přes seznam DATA. Všimněte si, že je mrháním času dávat příkazy DATA přímo, protože READ je nenajde. DATA příkazy musí vstupovat přímo do programu. Podívejte se, jak pracuje program, který jste právě natypovali. Řádek 1Ø říká počítači, aby četl tři různé data a nazval je a, b, c. Řádek 2Ø pak provádí PRINT těchto položek. Příkaz DATA na řádku 3Ø dává hodnoty a, b, c. Řádek 4Ø zastavuje program. Abyste viděli v jakém pořadí probíhá práce, změňte řádek 2Ø takto:

```
2Ø PRINT b, c, a
```

Informace v DATA může být část cyklu FOR...NEXT. Typujte:

```
1Ø FOR n = 1 TO 6
2Ø READ D
3Ø DATA 2, 4, 6, 8, 1Ø, 12
4Ø PRINT D
5Ø NEXT n
6Ø STOP
```

Když je program spuštěn vidíte, že příkaz READ čte přes seznam DATA. Příkazy DATA mohou také obsahovat řetězové proměnné. Např.

```
1Ø READ d$
2Ø PRINT "The date is", d$
3Ø DATA "June 1 st, 1982"
4Ø STOP
```

Je to jednoduchý způsob vybírání vyjádření ze seznamu DATA. Začíná na počátku a pracuje až dosáhne konce. Můžete však přinutit počítač, aby udělal skok v seznamu DATA použitím příkazu RESTORE, které udává číslořádku, donutí změnit posloupnost příkazů READ tak, aby začal na stanoveném řádku. (Pokud vynecháte číslo řádku v RESTORE počítač skočí na první řádek programu.) Zkuste program:

```
1Ø READ a, b
2Ø PRINT a, b
3Ø RESTORE 1Ø
4Ø READ x, y, z
5Ø PRINT x, y, z
6Ø DATA 1, 2, 3
7Ø STOP
```

Požadovaná data z řádku 1Ø jsou a = 1, b = 2. RESTORE vrací hodnoty a dovoluje, aby byly x, y, z čteny od prvního čísla příkazu DATA. Spusťte program bez řádku 3Ø a ověřte si, co se stane.

KAPITOLA 7 - VÝRAZY

SHRNUTÍ: operace +, -, *, /
vyjádření, vědecký význam, jména proměnných.

Už jste viděli některé způsoby jak umí ZX Spectrum počítat s čísly. Používá čtyři aritmetické operace +, -, *, /, (pamatujte si, že * se používá pro násobení a / se používá pro dělení) a umí nalézt hodnotu proměnné, která má jméno. Příklad:

```
LET tax = sum * 15/1ØØ
```

ukazuje velmi důležitý fakt, že se dají kalkulace spojovat. Taková kombinace jako sum * 15/1ØØ se nazývá výraz, který říká, aby hodnota pojmenovaná sum byla násobena 15 a dělena 1ØØ. Pokud tuto problematiku zatím neznáte, doporučujeme Vám podívat se do úvodní knihy, kde je ukázáno jak ZX Spectrum pracuje s čísly a pořadí jak vyhodnocuje matematické výrazy.

Shrnutí:

První se provádí násobení a dělení. Mají vyšší prioritu než sčítání a odečítání. Pro srovnání - násobení a dělení mají stejnou prioritu. Z toho vyplývá, že násobení a dělení se provádí v pořadí zleva doprava. Sčítání a odečítání se provádí později, tyto operace mají opět stejnou prioritu, proto se provádí zleva doprava. Potřebujete však znát, které operace mají nižší a které vyšší prioritu. Počítač přiděluje operacím čísla mezi 1 - 16, které udávají pricritu každé operaci. Násobení a dělení mají prioritu 8, sčítání a odečítání má prioritu 6. Tento pořádek počítání je vždy stejný, ale můžete ho změnit použitím závorek. Vše co je v závorkách se provádí první a pak se s tím zachází, jako s jednoduchým číslem. Výrazy jsou užitečné, protože když počítač od vás očekává číslo, můžete místo něj dát výraz. Existuje několik výjimek tohoto pravidla, na ty bude v každém případě upozorněno samostatně. Můžete rovněž sčítat mnoho řetězců (nebo řetězových proměnných) a můžete samozřejmě použít také závorek. Nyní bycho si měli říci, jaká jména proměnných můžete použít. Jak jsme již řekli, jména řetězových proměnných musí být jednoduchá písmena, zatím co jména numerických proměnných mohou být mnohem složitější. Používají jakékoliv písmena nebo i čísla, pokud je první písmeno. Můžete použít i mezery k lepšímu čtení výrazu, ale ty se při vnitřním zpracování neužívají. Také není rozdílu, zda je jméno natypováno velkými či malými písmeny. Zd jsou příklady jmen proměnných, která jsou přípustná;

```
x
t42
toto jméno je tak dlouhé, že ho nebudete umět typovat znovu bez chyby
now we are six
nOWWeaReSiX          } považuje se za identické
```

Nyní jsou uvedeny nepřípustná jména proměnných:

```
2ØØ1                  (začíná číslem)
3 bears               (začíná číslem)
M*A*S*H               (* není znak ani číslice)
Fotherington-Thomas   (- není znak ani číslice)
```

Numerické vyjádření může být reprezentováno číslem s exponentem, odkazujeme na úvodní část. Zkuste následující:

```
PRINT 2.34eØ
PRINT 2.34e1
PRINT 2.34e2
až po
PRINT 2.34e15
```

Vidíte, že počítač začíná užívat vědeckých výrazů. Je to proto, že při psaní čísla může být použito pouze 14 znaků. Podobné zkuste

```
PRINT 2.34e-1
PRINT 2.34e-2   atd
```

PRINT dává pouze 8 platných číslic. Zkuste:

```
PRINT 4294967295,4294967295-429e7
```

Toto ukazuje, že počítač může uložit jen čísla 4294967295 i když není připraven je celá zobrazit. ZX Spectrum užívá desetinnou aritmetiku, což zamená, že ukládá odělene mantisu a exponent. Není to vždy přesné, zvlaště pro celá čísla. Typujte:

```
PRINT 1e1Ø+1-1e1Ø,1e1Ø-1e1Ø+1
```

Čísla jsou uchována s přesností na 9,5 číslica tak 1e1Ø je příliš velké, aby bylo správně uchováno. Nepřesnost je větší než 1 a tak se výsledek příkladu 1e1Ø a 1e1Ø + 1 stáva záporným. Typujte další příklad:

```
PRINT 5e9+1-5e9
```

Vlivem zaokrouhlení součtu a rozdílu vychází výsledek roven 2. Číslo 5e9+1 a 5e9+2 jsou z hlediska počítač stejná. Největší celá čísl , které může být uložena je 4 294 967 295.

Řetězec " " bez znaků se nazývá prázdný nebo nulový řetězec. Pamatujte, že mezery jsou významné a prázdný řetězec není stejný jako řetězec bez mezer. Typujte:

```
PRINT "Have you finished"Finnegans Wake"yet?"
```

Na obrazovce můžete vidět, že každé dvojité uvozovky se tisknou jen jednou, ale vy je musíte natypovat dvakrát.

KAPITOLA 8 - ŘETĚZCE

SHRNUTÍ: Spojování řetězců, užití TO. Toto není součást standartního BASICU.

Pokud je dán řetězec, pod řetězec se skládá z několika jeho znaků, ve stejném sledu. Tak je "string" podřetězec "bigger string", ale "b string"a "bid reg" nejsou. Existuje výraz nazvaný "slicing", pro popis podřetězců a tento může být aplikován na sporné řetězcové vyjádření. Základní podstatou je

řetězcové vyjádření (začátek TO konec)

Tak např.

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

Když vynecháte začátek, očekává se, že je 1, když vynecháte konec, pak je řetězec ponechán do konce. Tak

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

(Můžete psát poslední příklad jako "abcdef"().) Obdobná forma postrádá TO a má právě jen jedno číslo.

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

Avšak normálně, jak začátek, tak konec musí odpovídat existující části řetězce, tomuto pravidlu je jedno nadřezení, pokud je začátek větší než konec, výsledkem je prázdný řetězec. Tak

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

dává chybu 3 subscript wrong, protože řetězec obsahuje příliš mnoho znaků, a 7 je příliš mnoho, ale

```
"abcdef"(8 TO 7)=""     a
"abcdef"(1 TO Ø)=""
```

Začátek a konec nesmí být záporná čísla, jinak dostaneme tato chybové hlášení B integer out of range. Tento program dokumentuje tato pravidla

```
1Ø LET a$="abcdef"
2Ø FOR n=1 TO 6
3Ø PRINT a$(nTO 6)
4Ø NEXT n
5Ø STOP
```

Po skončení chodu programu natypujte NEW. Typujte nový program:

```
1Ø LET a$="ABLE WAS I"
2Ø FOR n=1 TO 1Ø
3Ø PRINT a$(n TO 1Ø),a$((1Ø-n)TO 1Ø)
4Ø NEXT n
5Ø STOP
```

Pro řetězcové proměnné nemusíme pouze vybírat podřetězce, ale také je měnit. Např. typujte:

```
LET a$="I'm the ZX Spectrum"        a potom
LET a$ (5 TO 8)="xxxxxx"            a dále
PRINT a$
```

Všimněte si, že podřetězec a$(5 TO 8) je dlouhý pouze 4 znaky, proto byly použity pouze první 4 hvězdičky. To je charakteristické při změně podřetězců, podřetězec musí být přesně stejné délky jak byl dříve. Přesvědčte se, že je tomu tak. Příliš dlouhý řetězec, který je přiřazován je doplněn zprava mezerami - pokud je krátký. Nazývá se to "Procrustean assignment" (hospodský donutil své hosty, aby přespali v krátkých postelích buď s roztaženýma nebo pokrčenýma nohama). Nyní zkuste typovat:

```
LET a$()="Hello there"          a
PRINT a$;"."
```

Uvidíte něco podobného jako dříve (tentokrát s mezerami, protože a$() pracuje jako podřetězec).

```
LET a$="Hello there"
```

bude uděláno přesně. Komplikované řetězcové vyjádření bude kolem sebe potřebovat závorky, protože z nich může být dále vybíráno. Zkuste:

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

Cvičení:

1. Zkuste napsat program, který vytiskne den týdne použitím výběru z řetězce. Řetězec bude znít: SunMonTuesWedThursFriSat.

KAPITOLA 9 - FUNKCE

SHRNUTÍ: DEF
LEN, STR$, VAL, SGN, ABS, INT, SQR
FN

Uvažujte stroj na výrobu klobásek. Na jednom konci vložíte kus masa, pohnete pákou a na druhém konci vypadne klovása. Z vepřového jsou vepřové, z ryb rybí, z hovězího hovězí. Od stroje na klobásy se počítač liší tím, že pracuje s čísly, s řetězci místo masa. Vložte jednu hodnotu (nazvanou argument), udělejte nějakou operaci a dostanete další hodnotu (výsledek).

```
MASO ..... STROJ NA KLOBÁSY ..... KLOBÁSY
ARGUMENTY ..... FUNKCE ..... VÝSLEDEK
```

Různé argumenty dávají různé výsledky a pokud je argument naprosto nevhodný, funkce se neprovede a počítač hlásí chybu.
Stejně jako máme různé stroje, jeden na klobásy, druhý na oblečení, třetí na konzervy, budou různé funkce různě počítat. Každá bude mít vlastní hodnotu, která ji odliší od ostatních. Funkci použijete tak, že natypujete výraz, kde je jméno funkce, pak následuje argument a když je výraz vyhodnocen uvidíte výsledek. Jako příklad si uveďte funkci LEN, která určuje délku řetězce. Jejím argumentem je řetězec, jehož délku chcete zjistit. Např. typujte:

```
PRINT LEN "Sinclair"
```

počítač napíše 8, což je počet písmen ve slově (k získání LEN jako u většiny funkčních jmen, musíte užít rozšířený mod: stlačte CAPS SHIFT a SYMBOL SHIFT společně, aby se změnil kurzor z L na E a pak stlačte tlačítko K). Pokud funkce a operace do jednoho výrazu spojíte, pak funkce budou zpracovány před operacemi. Toto pravidlo můžete samozřejmě měnit použitím závorek. Např. jsou zde dva výrazy, které se rozlišují pouze v závorkách a kalkulace se v každém případě provádějí ve zcela obráceném pořadí. (Stává se však, že výsledky jsou stejné.).

```
LEN "Fred" + LEN "Bloggs"          LEN ("Fred"+"Bloggs")
4+LEN "Bloggs"                     LEN ("FredBloggs")
4+6                                LEN "FredBloggs"
10                                 10
```

Zde jsou další funkce:
STR$ konvertuje čísla na řetězce: argumentem je číslo a výsledkem je řetězec, který by se měl objevit na obrazovce, pokud bylo číslo zobrazeno příkazem PRINT. Zkuste typovat:

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

mělo by to mít přesný význam jako natypovat

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

nebo můžete napsat

```
PRINT LEN STR$ 100.0000
```

a dostanete odpověď 3,"100" = STR$ 100.0000.
VAL je obdoba STR$, ale opačného významu, mění řetězec na čísla. Např.

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

VAL je obrácené STR$, protože máte-li nějaké číslo, aplikujte na něj STR$ a potom zpětně VAL dá původní číslo. Pokud se však vezme řetězec a v něm se aplikuje VAL a potom STR$, ne vždy dostane původní řetězec. VAL je velmi mocná funkce, pro toto řetězec, který je argumentem není omezen na pouhé číslo, ale může být použito i numerické vyjádření. Např.

```
VAL "2*3"=6                        nebo dokonce
VAL ("2"+"*3")=6
```

Zde jsou spojeny dvě důležité věci. Zaprvé argument VAL je zpracován jako řetězec. Řetězcové vyjádření "2" + "3" je nejprve spojeno v řetězec "2+3". Pak je z řetězce sestaven výraz, jako kdyby nebylo uvozovek a je pronásobeno 2*3 s výsledkem 6. Nyní se vás pokusíme znásť:

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

(Pamatujte, že vnitřní uvozovky v řetězci musí být psány dvojitě. Když se podíváte na předchozí uvidíte, že uvozovky je třeba psát 4x nebo i 8x.)
Existuje podobná funkce jako VAL, ale asi méně užitečná, nazývá se VAL$. Jejím argumentem je stále řetězec, ale výsledek je rovněž řetězec. Tato funkce funguje jako VAL ve dvou krocích, prvně je argument jako řetězec, pak jsou odstraněny uvozovky a vše je chápáno jako číslo. VAL$ funguje stejně, ale výsledek je považován opět za řetězec. Typujte:

VAL$ """Fruit punch""="Fruit punch" a teď typujte:

LET a$="99"

a tisknčte následující: VAL a$, VAL "a$",VAL"""a$""",VAL$ a$,VAL$"a$", VAL$"""a$""". Některé z těchto příkladů budou pracovat, některé ne. Zkuste najít odpověď (mějte chladnou hlavu).
SGN je sing funkce (občas volané signum). Je to první funkce, kterou vidíte, že nemá nic společného s řetězci, protože její argument i výsledek jsou čísla. Výsledek je +1, když je argument kladný, 0 když je argument 0 a -1 když je argument záporný.
ABS je další funkce, jejíž argument i výsledek je číslo. Mění argument na kladné číslo (což je výsledek), tím že zapomeneme sign. Tak např.:

ABS-3.2 = ABS 3.2 = 3.2

INT znamená "celá část"; jde o číslo kladné nebo záporné. Mění desatinné číslo na celé tím, že zanedbává desetinnou část. Např.

INT 3.9 = 3

Pozor na záporná čísla, protože vždy zaokrouhluje dolů.

INT - 3.9 = -4

SQR vypočítává odmocninu čísla - výsledek je to, co násobíme stejným číslem.

SQR 4 = 2 protože 2*2=4
SQR 0.25 = 0,5 protože 0.5*0.5=0.25
SQR 2 = 1.4142136 (asi) protože 1.4142136*1.4142136=2.0000001

Pokud umocňujete číslo (i záporné), odpověď je vždy kladná. Tzn., že záporná čísla není možné odmocňovat a pokud tak učiníte dostanete zprávu "Chybný argument".
Také si můžete definovat vlastní funkci. Možná jména jsou FN následovaná znakem (pokud je výsledek číslo) nebo FN následovaná znakem $ (pokud je výsledkem řetězec). Zápis argumentu je nutno psát do uzavřených závorek. Funkci definujete tím, že napíšete příkaz DEF někde do programu. Např. je zde definice funkce, jejímž výsledkem je kvadrát argumentu.

10 DEF FN s(x)=x * x:REM čtverce x

DEF získáme v rozšířeném modu užitím SYMBOL SHIFT a 1. Když to typujete, počítač vám dá FN automaticky, protože v příkazu DEF vždy ihned následuje FN, s pak dokončuje jméno funkce. X v závorce je jméno, kterým se odkazuje na argument funkce. Můžete proto užít jakékoliv jméno (pokud je argument řetězec, pak písmeno následované $). Rovnítko potom znamená skutečnou definici funkce. Funkce může být definována jakýmkoliv výrazem a můžete se na ni odkazovat jako na argument, jako by to byla obyčejná proměnná. Když vložíte tuto řádku, můžete vyvolat funkci stejně jako kteroukoliv počítačovou funkci tím, že natypujete její jméno, FN s, pak následuje argument. Zopakujte si, že když sami definujete funkci, musí být argument uzavřen v závorkách.

PRINT FN s(2)
PRINT FN s(3+4)
PRINT 1+INT FN s (LEN"chicken"/2+3)

Pokud jste jednou napsali odpovídající řádek s DEF do programu můžete svobodně užívat vlastní funkční vyjádření, jak to dělá počítač.
Pamatujte: V některých dialektech BASICU musíte dokonce uzavírat argument počítačových funkcí do závorek. BASIC ZX Spectra to nevyžaduje.
INT vždy zaokrouhlujte dolů, aby zaokrouhlení bylo provedeno k nejbližšímu číslu je třeba použít konstanty 0.5. Můžete napsat svou funkci:

```
2Ø DEF FN r(x)=INT (x+Ø.5):REM  zaokrouhlení k nejbl. číslu
```

Pak dostanete:

```
FN r(2.9) = 3                    FN r(2.4) = 2
FN r(-2.9) = -3                  FN r(-2.4) = -2
```

Porovnejte výsledky a odpovědi, které dostanete při použití INT místo FN r.
Typujte a prověřte následující:

```
1Ø LET x=Ø: LET y=Ø: LET a=1Ø
2Ø DEF FN p(x,y)=a+x*y
3Ø DEF FN q()=a+x*y
4Ø PRINT FN p(2,3),FN q()
```

V tomto programu je hodně záludných bodů. Prvně, funkce není omezena na 1. argument, může mít více nebo dokonce žádný, ale stále musíte typovat závorky. Zadruhé, nezáleží na umístění DEF v programu. Po provedení řádky 1Ø se přeskočí na řádek 4Ø. Ten se provede, musí být součástí programu, nemůže to být samostatný příkaz. Zatřetí x a y jsou jména proměnných v programu a jména argumentu dočasně zapomínají, že jsou zvány x a y. Není žádný argument zvaný a, stále se opakuje proměnná a. Tak při provedení FN p(2,3) a má hodnotu stále 1Ø, protože se jedná o proměnnou, x má hodnotu 2, protože je to první argument, y má hodnotu 3, protože je to druhý argument. Výsledkem je 1Ø+2*3=16. Když se provádí FN q(), funkce nemá žádné argumenty, a, x, y, jsou proměnné a mají hodnoty 1Ø, Ø, Ø. Odpověď je v tomto případě 1Ø+Ø*Ø=1Ø. Nyní změňte řádek 2Ø na:

```
2Ø DEF FN p(x,y)=FN q()
```

Tentokrát FN p(2,3) bude mít hodnotu 1Ø, protože FN q se bude vracet k hodnotám x a y a argument FN p není užit. Některé BASICY (ne ZX Spectrum) mají funkce LEFT$, RIGHT$, MID$ a TL$.

LEFT$ (a$,n) dává podřetězec z a$ mající prvních n znaků.
RIGHT$ (a$,n) dává podřetězec z a$ mající posledních n znaků.
MID$ (a$, n1, n2) dává podřetězec z a$, který má n2 znaků a začíná na pozici n1.
TL$ (a$) dává podřetězec z a$, který se skládá ze všech znaků mimo prvního.

Můžete napsat uživatelsky definované funkce, které dělají totéž:

```
1Ø DEF FN t$(a$)=a$(2 TO):REM TL$
2Ø DEF FN I$(a$,n)=a$(TO n):REM LEFT$
```

Zkontrolujte, že tyto funkce pracují s řetězci o délce Ø nebo 1. Všimněte si, že naše FN I$ má dva argumenty, jeden je číslo a druhý je řetězec. Funkce může mít až 26 numerických argumentů (proč 26?) a současně až 26 řetězcových argumentů.

CVIČENÍ:

1. Používejte funkci FN s(x) = x*x pro testování SQR: měli byste shledat, že FN s(SQR x)=x, pokud dosadíte jakékoliv kladné číslo místo x a dále SQR FN a(x)=ABS x , kde x je kladné nebo záporné číslo (proč ABS?).

KAPITOLA 10 - MATEMATICKÉ FUNKCE

SHRNUTÍ: Umocňování, PI EXP, LN, SIN, COS, TAN, ASN, ACS, ATN

Tato kapitola pojednává o matematice, kterou provádí ZX Spectrum. Je docela možné, že nikdy z toho nic nepoužijete, ale raději tuto kapitolu vynechávejte. Zahrnuje operaci umocňování ↑, funkce EXP, LN, trigonometrické funkce SIN, COS, TAN a jejich inverzní funkce ASN, ACS, ATN.

↑ a EXP. Můžete umocňovat čísla na mocninu - to znamená násobit číslo samo sebou tolikrát, kolik určuje mocnina. Normálně se mocnina píše nad první číslo pro počítač je to obtížné a tak užíváme symbol ↑.

```
2 ↑ 1=2
2 ↑ 2 = 2*2 = 4
2 ↑ 3 = 2*2*2 = 8
2 ↑ 4 = 2*2*2*2 = 16
```

Toto byl elementární příklad, a ↑ b znamená násobit a b-krát, obvykle to má smysl, když b je celé kladné číslo. Najdeme definici, že to pracuje pro další hodnoty b. Uvažujme pravidlo:

a ↑ (b+c) = a ↑ b * a ↑ c

(Všimněte si, že jsme dali ↑ vyšší prioritu než násobení a dělení, takže když je několik operací ve vyjádření, ↑ se provádí před * a /. Nemůžeme uvažovat, že b a c jsou celá kladná čísla, pracuje to obecně.

a ↑ Ø = 1
a ↑ (-b) = 1/a↑ b
a ↑ (1/b) = b-tý kořen a říkáme číslu, které musíme násobit b-krát, aby vzniklo a.
a ↑ (b * c) = (a↑ b)↑ c

Pokud jste nic z toho dříve neviděli, nesnažte se to pamatovat. Stačí budete-li znát:

a ↑ (-1) = 1/a
a ↑ (1/2) = SQR a

a doufejme, že když znáte tyto funkce, bude mít pro vás smysl vše ostatní.
Vyzkoušejte si tento program:

```
1Ø INPUT a, b, c
2Ø PRINT a ↑ (b+c),a↑ b * a↑ c
3Ø GO TO 1Ø
```

Jistě je pravidlo, které jsme napsali dříve správné a každé číslo tištěné počítačem bude stejné. Další typický příklad pro užití této funkce je počítání úroků. Předpokládejte, že máte v bance uloženy peníze s 15 % úrokem za rok. Pak za rok nebudete mít pouze 100 %, ale taky 15 % úrok, který vám dá banka, dá dohromady 115 %. Sumu peněz násobíte 1,15 a máte celkovou částku za rok. Po dalším roce to bude stejné, budete mít 1,15 * 1,15 = 1,15↑ 2 = 1,3225 krát větší částka, než původní suma peněz. Obecně po y letech budete mít 1,15 y krát to s čím ste začali.

```
FOR y = Ø TO 1ØØ:PRINT y,1Ø*1,15↑ y:NEXT y
```

Viděli jste, že když začínáte s 1Ø Kčs roste suma tím více, čím více času uplyne, neuvažujeme-li však inflace a měny. Tento druh chování, když po prvním intervalu času se nějaká hodnota násobí sama sebou se nazývá exponenciální úloha a počítá se pomocí umocnění prvého čísla na necelou část. Předpokládejme, že jste natypovali

```
1Ø DEF FN a(x)=a ↑ x
```

zde je a více či méně první, hodnotou odpovídá úroku, který se nemění tak často. Je jistá hodnota, která dělá funkci FN a zvláště pěknou pro čí matematika a má jako základ přirozených logaritmů e. ZX Spectrum má funkci nazvanou EXP definovanou:

EXP x = e↑ x

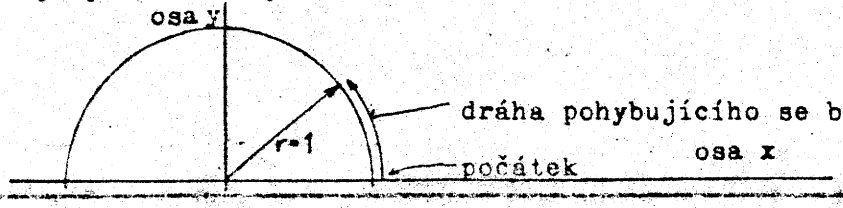
Naneštěstí samotné e není zvláště pěkné číslo, je to infinitní číslo. Můžete vidět prvních několik jeho čísel. Typujte:

```
PRINT EXP 1
```

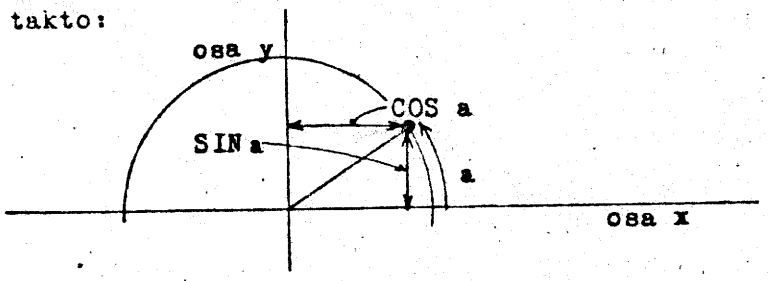
protože EXP 1 = e↑ 1 = e. Je to pouze přibližné vyjádření. e se nedá nikdy vyčíslit přesně.
LN je inverzní k exponenciální funkci. Logaritmus se základem a čísla x je výraz , na který musíme umocnit a, abychom získali číslo x a píše se $\log_a x$. Tak je definice $a^{\log_a x} = x$; dále taky platí $\log (a^x) = x$. Dobře jistě znáte logaritmy se základem 1Ø, to jsou běžné logaritmy. ZX Spectrum má funkci LN, která počítá logaritmy se základem e. Jsou to logaritmy přirozené. K výpočtu logaritmu o jiném základě použijte vztahu $\log_a x$ = LN x/LN a.

PI

Když je dána kružnice, můžete spočítat její obvod násobením průměru číslem PI. Podobně jako e, je i PI infinitní číslo, začíná 3.141592653589. Číslo PI získáte v rozšířeném modu a stlačením M. Zkuste PRINT PI.
SIN, COS, TAN, ASN, ACS, ATN. Trigonometrická funkce měří souřadnice pohybujícího se bodu po kružnici (pravoúhlé souřadnice). Jedná se o kružnici jednotkovou a bod se pohybuje kolem jejího středu. Začíná v poloze 3 hodiny a pohybuje se proti směru pohybu hodinových ručiček.



Přes střed kružnice nakreslíme dvě kolmé osy. Jedna jde přes 3 a 9 hodin a druhá přes 6 a 12 hodin. Kromě obvodu a = 2 PI zavedeme ještě dvě souřadnice, abychom viděli jak daleko je bod vpravo od osy y a jak vysoko nad osou x. Tyto vzdálenosti se nazávají kosinus a sinus. Funkce COS a SIN bude počítač tabulkovat takto:



Všimněte si, že když se bod dostává za osu y, kosinus je záporný a když se bod dostává pod osu x, stává se i sinus záporným. Další vlastnost je, že když se bod dostaně zpět do výchozí pozice, mají sinus a kosinus opět stejné hodnoty.

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

Tangens je definován jako podíl SIN a/COS a. Odpovídající funkce na počítači je TAN. Občas potřebujeme tyto funkce obráceně, tím že klademe hodnotu, která dala SIN, COS a TAN. Tyto funkce se nazývají arcsin (na počítači ASN), arcos (ACS) a arctg (ATN). V diagramu bodu, který se pohybuje podél kruhu se podívejte na poloměr středu kruhu. Vidíte, že vzdálenost, kterou urazil tento bod je cesta úhlu, pohybující se od osy x. Když a = PI/2 je úhel 9Ø, když a = PI je úhel 18Ø a když a = 2PI je úhel 36Ø stupňů. Pamatujte, že ZX Spectrum počítá zásadně v radiánech. Tak PI/2 radiánů je 9Ø stupňů. Ke změně stupňů na radiány dělte 18Ø a násobte PI, ke změně radiánů na stupně dělte PI a násobte 18Ø.

```
rad = úhel ve stupních * PI/18Ø
stupeň = rad * 18Ø/PI
```

KAPITOLA 11 - NÁHODNÁ ČÍSLA

SHRNUTÍ: RANDOMIZE
RND

Tato kapitola pojednává o funkci RND a klíčovém slově RANDOMIZE. Použití se zaměřuje na náhodná čísla, musíte však být opatrní, abyste je nepopletli. Obě jsou na stejném tlačítku (T). RANDOMIZE je označeno RAND. V některých případech je RND jako funkce, provádí kalkulace a dává výsledek. Nepotřebuje žádný argunemt. Pokaždé, když ji použijete, je jejím výsledkem nové náhodné číslo mezi Ø a 1. Někdy může dát hodnotu Ø, ale nikdy ne 1. Zkuste:

```
1Ø PRINT RND
2Ø GO TO 1Ø
```

a vidíte jak se mění odpověď. Můžete změnit algeritmus? Měli byste být schopni. Random znamená, že není vzor. Ve skutečnosti RND není náhodné číslo, protože jde o stálou posloupnost 65536 čísel. Říkáme, že RND je pseudonásobné číslo. Jak jsme již uvedli, dává RND náhodné čísla mezi Ø a 1, ale můžeme snadno získat i čísla v jiném rozsahu. Např. 5* RND je číslo mezi Ø a 5, 1.3+Ø.7*RND je mezi 1.3 a 2. K získání celých čísel použijte INT (pamatujte, že INT zaokrouhluje vždy dolů) jako 1+INT(RND*6), což simuluje házení kostkou. RND*6 je v rozsahu Ø - 6, ale ve skutečnosti nikdy nedostaneme 6, proto +1. Zde je program na kostku:

```
1Ø REM  program na kostku
2Ø CLS
3Ø FOR n=1 TO 2
4Ø PRINT 1+INT (RND*6);"";
5Ø NEXT n
5Ø INPUT aØ:GO TO 2Ø
```

Pokaždé, chcete-li házet kostkou stlačte ENTER.
Příkaz RANDOMIZE je použit, aby odstartoval RND v určitém místě posloupnosti čísel, jak ukazuje následující program:

```
1Ø RANDOMIZE 1
2Ø FOR n=1 TO 5: PRINT RND,:NEXT n
3Ø PRINT : GO TO 1Ø
```

Po každém provedení RANDOMIZE 1, začne sekvence RND číslem Ø,ØØ22735596. Můžete samozřejmě použít i jiná čísla od 1 - 65535 v příkazu RANDOMIZE, pak je začátek sekvence RND v jiných místech. Když máte program s RND a je v něm chyba, kterou nemůžete najít, pak pomůže RANDOMIZE, protože při každém spuštění se program chová stejně. Samotné RANDOMIZE (RANDOMIZE Ø má stejný účinek) je rozdílné, protože skutečně tvoří RND, jak ukazuje další program:

```
1Ø RANDOMIZE
2Ø PRINT RND : GO TO 1Ø
```

Sekvence, kterou dostanete není náhodná, protože RANDOMIZE používá čas od zapnutí počítače. Od té doby je každé RANDOMIZE provedeno stejně a proto i RND jsou stejná. Pro získání náhodných čísel by bylo lépe nahradit GOTO 1Ø GO TO 2Ø.

Poznámka: většina dialektů BASICU užívá RND a RANDOMIZE pro tvorbu náhodných čísel, ale ne všechny je získávají stejným způsobem.

Zde je program k házení mincí a počítá počet hozených hlav a orlů:

```
1Ø LET heads=Ø:LET tails=Ø
2Ø LET coin=INT (RND*2)
3Ø IF coin=Ø THEN LET heads=heads+1
4Ø IF coin=1 THEN LET tails=tails+1
5Ø PRINT heads;",";tails,
6Ø IF tails<>Ø THEN PRINT heads/tails;
7Ø PRINT:GO TO 2Ø
```

Poměr hlav a orlů by měl být přibližně 1, musíte však hrát dostatečně dlouho.

Cvičení:

1. Otestujte toto pravidlo, předpokládejte, že jste vybrali číslo mezi 1 a 872 a typujte RANDOMIZE vaše číslo, pak další hodnota RND bude (75 * ((vaše číslo + 1)- 1)/65536.
2. (pouze pro matematiky)
   Je dáno p - velké číslo a kořen modulu p. Když b_i je zbytek a_i modulo p ($1 \leq b_i \leq p-1$), sekvence $\frac{b_i - 1}{p - 1}$ je cyklická posloupnost p - 1 čísel rozsahu Ø - 1 (vč. 1). Výběrem vhodných konstant mohou tato čísla vypadat jako náhodná.
   65537 ... 2na 16 + 1. Protože skupina nenulových zbytků modulo 65537 má 2 mocninu, zbytek je primitivní kořen. Užitím Gausova zákona kvadratické reciprocity lze dokázat, že 75 je primitivní kořen modulo 65537.

ZX Spectrum používá p = 65537 a a=75 a ukládá $b_i - 1$ v paměti, RND zobrazuje v paměti bi - 1 číslem $b_{i+1} - 1$. RANDOMIZE n ($1 \leq n \leq 65535$) nahrazuje b_i rovno n+1. RND má přibližně rovnoměrné rozložení v rozmezí Ø - 1.

KAPITOLA 12 - POLE

SHRNUTÍ: Pole (způsob, kterým ZX Spectrum pracuje s řetězovými poli není standartní). DIM...

Předpokládejte, že máte seznam čísel, např. známky 1Ø lidí ve třídě. Aby se uchovaly v počítači, mohli byste každé osobě stanovit jednoduchou proměnnou, ale shledali byste to jako nevhodné. Můžete se rozhodnout volit proměnné BLOGGS 1 , BLOGGS 2 , ale pro 1Ø čísel by to bylo dlouhé a nudné typování. Mnohem snažší je typovat takto:

```
5 REM tento program nefunguje
1Ø FOR n = 1 TO 1Ø
2Ø READ bloggs n
3Ø NEXT n
4Ø DATA 1Ø,2,5,19,16,3,11,1,Ø,6
```

Toto je mechanismus, kterým můžete aplikovat tuto myšlenku užitím pole. Pole je řada proměnných, jejich elementů, všechny mají stejné jméno a liší se pouze číslem psaným v závorce za jménem. V našem případě by jméno mohlo být b. (podobně jako řídicí proměnná FOR-NEXT cyklu, jméno pole musí být jednoduché

písmeno), tedy deset proměnných by mohlo být b(1), b(2), ... b (1Ø). Položky pole se nazývají indexové proměnné, stojí v protikladu jednotlivých proměnných, které již znáte. Před použitím pole musíte pro ně rezervovat v počítači místo, použitím DIM (dimense) příkazu.

```
DIM b(1Ø)
```

deklaruje pole zvané b s mezí 1Ø(je deset proměnných b(1) ... b(1Ø)) a tyto hodnoty jsou inicializovány na Ø. To taky ruší jakékoliv pole, zvané b, které bylo dříve. (ale ne jednoduché proměnné. Pole a jednoduché proměnné se stejným jménem mohou existovat současně vedle sebe, nemělo by dojít ke zmatku, protože pole má vždy ještě index.) Indexem může být numerické vyjádření, můžete taky psát:

```
1Ø FOR n = 1 TO 1Ø
2Ø READ b(n)
3Ø NEXT n
4Ø DATA 1Ø,2,5,19,16,3,11,1,Ø,6
```

Můžete rovněž deklarovat vícerozměrné pole. Ve dvourozměrném poli použijete ke specifikaci pole dvě čísla, podobně jako v matici řádek a sloupec, nebo televizní obrazovku. Dále, pokud si představíte řádek a sloupec, že odpovídají tištěné straně, mohli byste dostat další rozměr pro čísla stran. Hovoříme zde o numerických polích, takže prvky by nebyly tištěny jako znaky v knize, ale jako čísla. Když uvažujeme 3 prvky trojrozměrného pole v proměnné v, pak jde o číslo stránky, číslo řádku, číslo sloupce. Např. navrhujeme deklaraci 2rozměrného pole c s rozměry 3 a 6 s použitím příkazu DIM.

```
DIM c(3,6) dostaneme 3*6=18 indexových proměnných
c(1,1),c(1,2),...,c(1,6)
c(2,1),c(2,2),...,c(2,6)
c(3,1),c(3,2),...,c(3,6)
```

Tento princip platí pro jakoukoliv dimenzi. I když můžete mít čísla a pole se stejnými jmény, nemůžete mít se stejnými jmény dvě pole, i když mají rozdílný počet dimenzí.

Existují také řetězová pole. Řětěz a řetězové pole se od sebe liší tím, že pole má pouze délku a přiřazení je provedeno tak, že zbytek délky, který není obsazen je nahražen mezerami. Můžeme mít zvláštní pole jednoduchých znaků (ale s jedním dalším rozměrem). Jméno řetězového pole je 1. písmeno, které následuje $ a řetěz, ani řetězové podle nesmí mít návzájem shodné jméno (na rozdíl od polí numerických). Teď předpokládejme, že chceme pole a$ o pěti řetězcích. Musíte se rozhodnout, jak dlouhé budou tyto řetězce, předpokládejme, že 1Ø znaků postačí. Pak píšeme:

```
DIM a$(5,1Ø)
```

Je deklarováno pole 5*1Ø znaků, ale můžete také uvažovat o řadě znaků jednotlivých:

```
a$(1)=a$(1,1)a$(1,2) ... a$(1,1Ø)
a$(2)=a$(2,1)a$(2,2) ... a$(2,1Ø)
  :    :      :        :
a$(5)=a$(5,1)a$(5,2) ... a$(5,1Ø)
```

Tedy, když dáte stejná čísla indexů (v tomto případě 2), podle dimensí v příkaze DIM, Pak dostaneme jeden znak, ale když nenapíšete druhý index, dostanete řetězec první délky. Tak např. A$(2,7) je znak řetězce A$(2). Mohli bychom vlastně také psát A$(2)(7). Zkuste:

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

Pro poslední index (ten, který můžeme vynechat) lze použít techniku pro výběr části řetězce (slicing):

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

Pamatujte:

V řetězovém poli mají řetězce stejnou pevnou délku. Příkaz DIM má zvláštní číslo (poslední), které určuje délku řetězce. Když napíšete indexovou proměnnou můžete přidat další číslo, nebo lze vybrat jen určené znaky z řetězce z čísel v příkaze DIM. Můžete mít řetězová pole bez rozměru. Typujte:

```
DIM a$(1Ø)
```

a uvidíte, že a$ se chová stejně jako řetězová proměnná s výjimkou, že má

délku 1Ø a během přiřazení je zbytek doplněn mezerami.
Cvičení:
1. Použijete READ a DATA k načtení řetězce, kde mŽ je 12 řetězců a mŽ(n) je jméno n-tého řetězce (pozn. DIM příkaz bude DIM mŽ(12,9). Otestujte tiskem za použití vyklu. Typujte:
   PRINT "nyní je měsíc";mŽ(5);"ing";"kdy mladíci hrají na námluvy"
   Co můžete dělat s těmi mezerami?

KAPITOLA 13 - PODMÍNKY

SHRNUTÍ: AND, OR
NOT

V kapitole 3 jsme viděli formu, jakou má příkaz IF
IF podmínka THEN ...
Podmínky tam byly vztahy (=,<,>,<=,>=,<>), které porovnávají dvě čísla nebo dva řetězce. Můžete také několik vztahů kombinovat pcužitím logických operací AND, OR a NOT. Jedna relace AND a druhá relace je pravdivá, když jsou pravdivé obě dvě, mějme takový příklad:
IF aŽ="yes" AND x>Ø THEN PRINT x
a x je tisknuto pouze tehdy, jsou-li splněny obě podmínky. BASIC je zde spojen těsně s angličtinou, takže se zdá být zbytečné vysvětlovat podrobnosti. Stejně jako v jazyce, můžete spojit více vztahů pomocí AND a celek je pravdivý tehdy, jsou-li pravdivé jednotlivé vztahy. Relace OR jako druhá je pravdivá, když alespoň jedna z těchto dvou relací je pravdivá. (Pamatujte, že je pravdivá i tehdy, jsou-li pravdivé obě - toto není v jazyce vyjádřeno přesně.) NOT relace obrací věci naopak. Záporná relace NOT je pravdivá, když je relace nepravdivá a naopak.
Logické vyjádření může být provedeno s AND, OR, NOT stejně jako číselné výrazy vyjadřujeme pomocí +, -, atd. Pokud je to potřebné užijte závorek. Priority jsou stejné jako u operací +, -, *, /, ↑ . OR má nejnižší prioritu, pak je AND, pak NOT, pak zleva vztahy a obyčejné operace. NOT je funkce s argumentem a výsledkem, ale její priotira je mnohem nižší než ostatních funkcí. Proto její argument nepotřebuje závorky, ani když obsahuje AND, pak OR (nebo obě). NOT a=b znamená totéž jako NOT (a=b) a totéž jako a<>b). <> je negace =, znamená to, že je pravdivá pouze, když rovnost je nepravdivá. Jinak a<>b je totéž jako NOT a=b. Dále NOT a<>b je totéž jako a=b. Přesvědčte se, že >= a <= jsou negace <,> a tak NOT můžete vynechat změněním relací. Dále platí NOT (první log. výraz AND druhý log. výraz) je totéž jako NOT (první log. výraz) OR NOT (druhý log. výraz) a NOT (první log. výraz OR druhý log. výraz) a stejně jako NOT (první log. výraz) AND NOT (druhý log. výraz). Při použití NOT v programu není nutné mezi vztahy používat závorek. Použití NOT v programu je zjednodušující prvek. Další sekce je trochu komplikovaná a můžete ji tedy přeskočit. Zkuste:
PRINT 1=2,1<>2
očekáváte syntaktickou chybu. Ve skutečnosti počítač nezná logické hodnoty, místo nich očekává čísla a s nimi podle několika pravidel pracuje.
(i) =,<,>,<=,>=,<> , dávají numerické výsledky, 1 pro pravdivý výrok Ø pro nepravdivý. Příkaz PRINT tiskne Ø pro "1=2", protože jde o nepravdivý výrok, "1<>2" dává 1, což je pravdivý výrok.
(ii) v IF podmínka THEN ...
Podmínka může být jakékoliv číselné vyjádření. Pokud je jeho hodnota Ø, pak se chová jako při nepravdivosti a všechny ostatní hodnoty (vč. 1) se berou jako pravdivý výrok. Tak výrok IF příkaz ve skutečnosti znamená
IF podmínka <> Ø THEN ...
(iii) AND, OR, NOT jsou operace tvořící číselné hodnoty

x AND y má hodnotu	x, pokud y je pravdivé (ne Ø) Ø, pokud y je nepravdivé
x OR y má hodnotu	1, když je y pravdivé (ne Ø) x, když je y nepravdivé

NOT x má hodnotu Ø, (nepravdivé), když x je pravdivé (ne Ø)
1, (pravdivé), když x je nepravdivé (ne Ø)

(všimněte si, že pravdivé znamená ne Ø, když kontrolujeme získanou hodnotu, ale znamená 1, když produkujeme novou hodnotu.)
Přečtěte si tuto kapitolu znovu a ujistěte se, že tomu všemu rozumíte. Ve výrazech x AND y, x OR y, NOT x, x a y budou mít hodnoty Ø a 1 pro nepravdivé a pravdivé výroky. Výroky fungují podle 1Ø pravidel, 4 pro AND a OR a 2 pro NOT. Zkontrolujte, zda se provádějí tak, jak bylo vysvětleno v této kapitole. Zkuste tento program:

```
1Ø INPUT a
2Ø INPUT b
3Ø PRINT (a AND a>=b)+(b AND a<b)
4Ø GO TO 1Ø
```

pokaždé se tisknou větší ze dvou čísel a, b. Přesvědčete se, že můžete uvažovat

x AND y znamená
x je-li y (pokud výsledek je Ø)
x OR y znamená x pokud není y (v případě, že výsledek je 1).

Výraz používající AND a OR je nazýván podmínkový výraz. Příklad užití OR může být i

```
LET total price = price less tax *(1.15 OR v$="zero rated")
```

Všimněte si, jak AND vede ke sčítání (protože její hodnota je Ø) a OR vede k násobení (její hodnota je 1). Rovněž řetězové hodnoty mohou být v podmínkových výrazech, ale pouze při užití AND.

x$ AND y má hodnotu x$, když y není Ø
"", když y je Ø

Totéž je x$ je-li y (jinak prázdný řetězec). Zkuste program, kde vatupují dva řetězce a ten je dává do aritmetického pořadí:

```
1Ø INPUT "natypujte dva řetězce",a$, b$
2Ø IF a$>b$ THEN LET c$=a$: LET a$=b$: LET b$=c$
3Ø PRINT a$;" ";("<" AND a$<b$)+)"=" AND a$=b$);" ";b$
4Ø GO TO 1Ø
```

Cvičení:
1. BASIC může někdy pracovat od angličtiny odlišně, uvažujte větu "pokud a ≠ b nebo a". Jak byste to napsali v BASICU? Odpověď není
IF A<>B OR C ani IF A<>B OR A<>C

KAPITOLA 14 - ZNAKY

SHRNUTÍ: CODE, CHR$
POKE, PEEK
USR
BIN

Písmena, číslice, interpunkční znaménka apod., které se mohou natypovat v řetězcích se nazývají znaky a tvoří abecedu, nebo systém znaků, které používá ZX Spectrum. Většina těchto znaků jsou jednoduché symboly, ale také symboly nazvané "vybrané", které reprezentují celá slova jako PRINT, STOP, .. Existuje celkem 256 znaků, které mají kod mezi Ø a 255. Seznam je v příloze A. Znaky a kódy se dají měnit funkcemi CODE a CHR$. CODE je aplikován na řetězce, dává kod prvního znaku v řetězci nebo Ø pokud jde o prázdný řetězec. CHR$ je aplikován na čísla a dává znak podle hodnoty čísla. Tento program tiskne všechny znaky:

```
1Ø FOR a=32 TO 255: PRINT CHR$ a;: NEXT a
```

Nahoře vidíte mezeru, 15 symbolů a symbolických znamének, 1Ø číslic, sedm dalších symbolů, velké písmena, šest dalších symbolů, malé písmena a pět dalších symbolů. Všechny symboly (s výjimkou £ a ©) jsou z kódu známého jako ASCII.

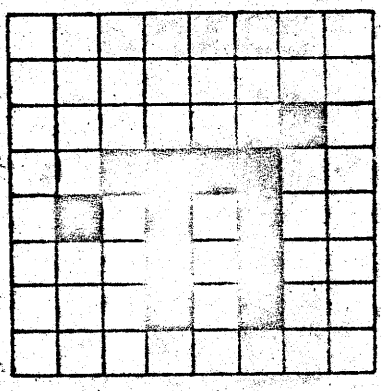
Numerické kody těchto znaků jsou kody, které používá ZX Spectrum. Zbytek znaků není částí ASCII a jsou použity ve Spectru. První mezi nimi je mezera a 15 vzorů čermobílých. První jsou grafické symboly a používají se pro kreslení obrázků. Můžete je vkládat z klávesnice použitím grafického modu. Když zmáčknete GRAPHICS (CAPS SHIFT a 9), kurzor se změní na G. Nyní stlačte

Číslice 1 - 8 a budou se objevovat grafické symboly - buď to jsou symboly na tlačítku a se stiskem SHIFT dávají invertovaný symbol, tj. černé se stává bílým a naopak. Bez ohledu na tlačítka číslice 9 dává zpět L-mod a číslice Ø slouží jako DELETE.
Zde je 16 grafických symbolů:

SYMBOL	KOD	Jak získat	SYMBOL	KOD	Jak získat
	128	G 8		143	G shift a 8
	129	G 1		142	G shift a 1
	130	G 2		141	G shift a 2
	131	G 3		140	G shift a 3
	132	G 4		139	G shift a 4
	133	G 5		138	G shift a 5
	134	G 6		137	G shift a 6
	135	G 7		136	G shift a 7

Po grafických symbolech pak uvidíte kopii abecedy od A do U. Tyto znaky můžete modifikovat sami, ačkoliv když je počítač prvně zapnut jsou k dispozici pouze znaky základní. Uživatelské znaky můžete typovat z klávesnice v grafickém modu a pak užít písmena od A do U. K definici nových znaků se držte tohoto popisu. Definujeme např. písmeno

I) PI. Určete jak má znak vypadat. Každý znak je tvořen maticí 8x8 čtvercových bodů, každý z nich může být barvy papíru nebo inkoustu (podívejte se do úvodní knížky). Měli byste si nakreslit diagram a černými čtverci pro inkoustovou barvu (ink).



Jeden čtverec kolem okraje musíme nechat prázdný, protože ostatní písmena mají rovněž takový okraj (kromě malých písmen s ocásky, jdoucí až ke kraji).

II) Rozhodněte, který uživatelský grafický znak bude zobrazovat PI, řekněme že P a tak ho v grafickém modu stiskněte.

III) Uchovejte nový vzor. Každý uživatelský vzor (znak) má svůj vzor jako 8 čísel pro jednu řadu. Můžete tyto čísla psát jako BIN, za kterým následuje osm Ø nebo 1 (Ø papír, 1 inkoust).
Oněch 8 řad pro písmeno PI bude vypadat následovně:

```
BIN ØØØØØØØØ
BIN ØØØØØØØØ
BIN ØØØØØØ1Ø
BIN Ø11111ØØ
BIN Ø1Ø1Ø1ØØ
BIN ØØØ1Ø1ØØ
BIN ØØØ1Ø1ØØ
BIN ØØØØØØØØ
```

(pokud víte o binárních číslech pak by vám mělo pomoci, že BIN je použit k psaní čísla v binární formě místo obvyklé formy desetinné).

Těch 8 číslic je v paměti na 8 místech, každé má svou adresu. Adresa 1. bytu nebo skupiny 8 čísel je USR"P" (P je to, co chceme změnit) a další jsou USR "P"+1 až do 8, které má adresu USR"P"+7. USR je zde funkce, která mění řetězcový argument na adresu 1. bytu v paměti pro odpovídající uživatelem definovanou grafiku. Řetěz musí být jeden znak, který může být buď uživatelem definovaný grafický symbol nebo odpovídající písmeno (horní nebo spodní). Funkce USR se užívá, když argument je číslo, se kterým budeme pracovat. Pokud tomu ještě nerozumíte, kde o užití programu:

```
1Ø FOR n=Ø TO 7
2Ø INPUT row:POKE USR"P"+n,row
3Ø NEXT n
```

Program se zastaví 8x pro data, kdy můžete typovat 8 BIN čísel, typujte je správně, začněte v horní řadě.

Příkaz POKE uchovává čísla přímo v paměti počítače. Protiklad POKE je PEEK, ten umožňuje prohlížet obsah paměti, ale nemění ho. Tyto příkazy budou přesně probrány v kapitole 24. Po uživatelských znacích přicházejí "tokens". Mohli jste si všimnout, že jsme netiskli prvních 32 znaků s kody Ø až 31. Jsou to kontrolní znaky. Nevytváření nic, co se dá tisknout, ale mají účinek na televizi nebo jsou využity ke kontrole jiných věcí než televize a televizního tisku; aby ukázala, že jim nerozumí. Popsány jsou v příloze A. Televize používá 3 znaky s kody 6, 8 a 13.

CHR$ 8 je jeden z užitečných.

CHR$ 6 tiskne přesně mezery jako např. v příkaze

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

Lepší použití

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

CHR$ 8 je zpět-mezera: posouvá tiskovou pozici o 1 zpět. Zkuste typovat:

```
PRINT "1234"; CHR$ 8;"5"  což tiskne 1235
```

CHR$ 13 je "newline", posouvá tiskovou pozici na počátek dalšího řádku. Televize rovněž užívá kody 16 a 23. Jsou vysvětleny v kapitole 15 a 16. Všechny kontrolní znaky jsou v příloze A. Použitím kodů za znaky můžeme rozšířit abecedu, ukrýt řetězce s libovolnými znaky, ne pouze s písmeny. Místo obvyklých 26 písmen si můžeme abecedu rozšířit na 256 znaků v témž pořadí jak jejich kody. Např. tyto řetězce jsou v pořadí abecedy ZX Spectra (všimněte si, že malá písmena jsou za velkými, tak "a" je za "Z".

```
CHR$ 3+"ZOOLOGICAL GARDENS"
CHR$ 8+"AARDVARK HUNTING"
" AAAARGH!"
"(Parenthetical remark)"
"1ØØ"
"129.95 inc. VAT"
"AASVOGEL"
"Aardvark"
"PRINT"
"Zoo"
"(interpolation)"
"aardvark"
"aasvogel"
"zoo"
"zoology
```

Zde je pravidlo v jakém pořadí budou zobrazeny dva řetězce. Nejprve se porovnají první znaky. Pokud jsou rozdílné, pak jeden má kod menší než druhý a řetězec, ze kterého je znak o nižším kodu je menší. Když jsou první znaky stejné, porovnávají se stejným způsobem znaky další. Řetězce jsou shodné, když jsou si všechny znaky rovny.
Relace =,< ,> ,<=,>=,<> jsou použity pro řetězce i pro čísla
< znamená je před a > znamená přijde potom.

```
"AA man"< "AARDVARK"
" AARDVARK" > "AA man"          jsou pravdivé
```

<= a >= pracuje stejné jako s čísly

```
"The same string" <= "The same string"    je pravdivý výrok, ale
"Stejný řetězec" <   "Stejný řetězec"     je výrok nepravdivý
```

Program, který zpracovává dva vstupní řetězce je následující:

```
10 INPUT "Typujte řetězec:",a$,b$
20 IF a$>b$ THEN LET c$=a$: LET a$=b$: LET b$=c$
30 PRINT a$;" ";
40 IF a$<b$ THEN PRINT "<";:GO TO 60
50 PRINT "="
60 PRINT "  ";b$
70 GO TO 10
```

Všimněte si, že jsme museli zavést c$ v řádce 20, protože pouhá a$ a b$ by nemělo patřičný význam:

```
LET a$=b$: LET b$=a$
```

Tento program pracuje s uživatelskou grafikou a ukazuje šachové figury.

P pro pěšáka
R pro věž
N pro střelce
B pro koně
K pro krále
Q pro královnu

```
 5 LET b=BIN 01111100:LET c=BIN
   00111000: LET d=BIN 00010000
10 FOR n=1 TO 6:READ p$: REM 6 pieces
20 FOR f=0 TO 7:REM   čti do 8 bytů
30 READ a:POKE USR p$+f,a
40 NEXT f
50 NEXT n
100 REM bishop
110 DATA "b",0,d, BIN 00101000, BIN 01000100
120 DATA BIN 01101100,c,b,0
130 REM king
140 DATA "k",0,d,c,d
150 DATA c, BIN 01000100,c,0
160 REM rook
170 DATA "r",0,BIN 01010100,b,c
180 DATA c,b,b,0
190 REM queen
200 DATA "q",0, BIN 01010100, BIN 00101000,d
210 DATA BIN 01101100,b,b,0
220 REM pawn
230 DATA "p",0,0,d,c
240 DATA c,d,b,0
250 REM knight
260 DATA "n",0,d,c,BIN 01111000
270 DATA BIN 00011000,c,b,0
```

Všimněte si, že Ø může být použita místo BIN ØØØØØØØØ. Nechte program proběhnout a sledujte jak se změní grafické symboly (v grafickém režimu).

Cvičení:

1. Představte si, že místo pro 1 symbol je rozděleno na 4 části jako Batenburský koláč. Pak když každá čtvrt může být buď černá nebo bílá existuje 2*2*2*2=16 možností. Najděte je v souboru znaků.

2. Spusťte tento program:
```
1Ø INPUT a
2Ø PRINT CHR$ a;
3Ø GO TO 1Ø
```
Pokud budete experimentovat shledáte, že CHR$ a je zaokrouhlena k nejbližším nižším hodnotám. Pokud není v rozsahu Ø až 255, program se zastaví s ohlášením B integer out of range.
3. Který ze dvou řetězců je menší? "EVIL" - "evil"
4. Jak modifikovat program, aby se grafické symboly definovaly pomocí READ a DATA místo příkazu INPUT.

KAPITOLA 15 - VÍCE O PRINT A INPUT

SHRNUTÍ:
- CLS
- PRINT položky
- Výrazy (numerické nebo řetězové), TAB numerické vyjadřování, AT numerické vyjádření, numerické výrazy
- PRINT operátory ,;
- INPUT položky: proměnné (numerický i řetězový typ)
- LINE řetězové proměnné
- Položky PRINT nezačínající písmenem ("Tokens" nejsou považovány za znaky
- Scrolling
- SCREEN$

Příkaz PRINT se používá dost často, takže jistě už víte jak pracuje a jak se ho užívá. Výrazy, jejichž položky jsou tištěny se nazývají položky PRINTu a jsou odděleny čárkami nebo středníky, které se nazývají separátory PRINT. V tiskových položkách nemusí být nic zastoupeno, je to stejné jako užití dvou čísel za sebou. Je však mnohem více druhů tisku, které říkají ne co, ale kam tisknou. Např. PRINT AT 11, 16;"*" tiskne hvězdu uprostřed obrazovky.

AT řádek,sloupce

Posouvá pozici PRINT na specifikované místo (řádek, sloupec). Řádky se číslují od Ø (horní) po 21, sloupce od Ø (vlevo) po 31.
SCREEN$ je opačná k funkci PRINT AT a řekne jaký znak je v určité pozici na obrazovce. Použitá čísla řádky a sloupce stejně jako PRINT AT, ale jsou uzavřeny v závorkách. Např.

PRINT SCREEN$ (11,16)

Znovu vytiskne hvězdu, která byla tištěna v předešlém příkladě. Znaky se berou normálně jako znaky, mezery se vrací jako mezery. Řádky nakreslené PLOT, DRAW, CIRCLE, uživatelské znaky a grafické symboly se vracejí jako nulový (prázdný) řetězec. Podobně se užívá OVER k vytvoření kompozičního znaku.

Na straně 76 originálního manuálu je uvedeno schema televizní obrazovky s číslováním řádků, sloupců i jednotlivých bodů.

Na poslední dva řádky normálně nemůžeme užít PRINT a PLOT.

TAB column (sloupec)

tiskne mezery k přesunu pozice PRINT do specifikovaného sloupce. Zůstává na stejném řádku. Pamatujte, že počítač redukuje číslo sloupce modulo 32 (dělí 32 a bere zbytek), tak např. TAB 33 znamená totéž jako TAB 1.

PRINT TAB 3Ø;1;TAB 12;"Contents"; AT 3,1;"CHAPTER";TAB 24;"page"

znamená tisk hlavičky obsahu na straně 1 knihy. Zkuste program:

```
1Ø FOR n=Ø TO 2Ø
2Ø PRINT TAB 8*n;n;
3Ø NEXT n
```

program ukazuje, že čísla TAB jsou redukovány modulo 32. Pro elegantnější příklad změňte 8 v řádku 2Ø na 6.

Některé důležité body:

1) Tyto nové body je nejlépe ukončovat středníky. Je možno užít závorky nebo nic na konci řádku, ale to znamená, že na příkazu PRINT se pozice znovu změní a potom je pozici třeba znovu vrátit.
2) Nemůžete tisknout na spodní dva řádky (22 a 23), protože jsou rezervovány pro příkazy, vstupní data, zprávy atd. Odkazy na spodní řádek obvykle znamenají řádek 21.
3) Můžete použít AT, aby PRINT byl proveden tam, kde je již něco vytisknuto starý tisk bude přepsán.

Další příkaz dotýkající se PRINTje CLS. Ten čistí celou obrazovku jako při CLEAR a RUN. Když tisk dosáhne spodní části obrazovky, začínají se řádky posunovat nahoru podobně jako u psacího stroje. Proveďte:

CLS: FOR n=1 TO 22: PRINT n: NEXT n a potom PRINT 99 několikrát.

Pokud počítač něco tiskne, dává pozor, aby se ujistil, že nemaže horní část obrazovky. Vidíte to typujete-li:

CLS: FOR n=1 TO 1ØØ: PRINT n: NEXT n

Když je obrazovka plná, výpočet se zastaví s dotazem: 'scroll?' na konci obrazovky. Nyní máte čas prohlédnout si prvních 22 řádků. Když pak stlačíte y (jako ano), počítač vám dá další obraz čísel. Vlastně jakékoliv tlačítko mimo n, STOP a SPACE má stejný účinek. Tyto přimějí počítač, aby se zastavil se zprávou: D BREAK - CONT repeats.

Příkaz INPUT může provádět víc, než jsme dosud uvedli. Zatím jsme příkaz používali asi následovně:

INPUT "How old are you?",age

v kterém tiskne počítač uvedenou otázku v dolní části obrazovky a vy potom typujete svůj věk.

Ve skutečnosti je příkaz INPUT sestaven z položek stejně jako PRINT a tak 'How old are you' a 'age' jsou obě položky INPUT. Všeobecně jsou položky INPUT a PRINT stejné, ale jsou zde velmi důležité rozdíly. Zaprvé obvyklý vstup INPUT je proměnná, jejíž hodnota musí být vložena - příklad je age. Pravidlo je takové, že když položka INPUT začíná znakem musí to být proměnná, jejíž hodnotu pak vložíme. Zadruhé, zdálo by se že nemůžete tisknout hodnoty proměnných jako část záhlaví, ale provedete to tak, že kolem proměnné dáte závorky. Jakékoliv vyjádření začínající písmenem musí být uzavřeno v závorkách, pokud má být tištěno jako záhlaví. Jakýkoliv druh položky PRINT není prováděn stejně jako INPUT. Zde je příklad ilustrující co se provádí:

```
LET my age=INT (RND * 1ØØ):INPUT ("I am";my age;".");"How old are
you?",your age
```

'My age' je v závorce proto, že se její hodnota tiskne, your age není v závorce, proto musíme typovat její hodnotu. Vše co je v příkaze INPUT se píše v dolní části obrazovky, což se chová nezávisle na horní části obrazovky. Ve všech případech jsou tyto řádky vztaženy k horní řádce spodní půlky, i když se pohlo televizním screen (stane se když typujete hodně dat v INPUT). Abyste poznali jak pracuje AT v příkaze INPUT zkuste provést:

```
1Ø INPUT "To je řádek 1.",a$; AT Ø,Ø;"To je řádek Ø.",a$;AT 2,Ø;
   "To je řádek 2.",a$; AT 1,Ø;"To je stále řádek 1.",a$
```

Nyní zkuste:

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

Dolní část obrazovky se posouvá směrem nahoru , horní část je neporušena až má být napsáno na tentýž řádek v PRINT. Pak se horní část začne rolovat. Další možností v příkazu INPUT, kterou jste dosud nepoznali je typ LINE vstup a jde o jiný způsob vkládání řetězcových proměnných.
Pokud napíšete LINE před jméno řetězové proměnné (při vstupu) např.

```
INPUT LINE a$
```

Pak počítač nedá řetězové uvozovky, jak je pro řetězec obvyklé, ale očekává proměnnou jako kdyby tam uvozovky byly. Tak když napíšete cat jako vstup dat a$ dostane hodnotu cat. Protože řetězové uvozovky nejsou v řetězci, nemůžete je zrušit a typovat tak jiný druh řetězového vyjádření pro vstup dat. Pamatujte, že nemůžete užít LINE pro numerické proměnné. Kontrolní znaky CHR$ 22 a CHR$ 23 mají účinek jak AT a TAB. Jde spíše o kontrolní znaky, proto kdekoliv jsou na televizní obrazovce musí být následovány dvěma dalšími znaky, které nemají jejich obvyklý účinek, počítá se s nimi jako s čísly (jejich kody) ke specifikaci sloupce (pro AT) a pozice (pro TAB). Téměř vždy je snadnější užít AT a TAB v jejich obvyklé formě, než použít kontrolní znaky, ale i ty jsou někdy užitečné. V kontrolním znaku je CHR$ 22 (pro AT). První znak za specifikuje číslo řádku a druhý číslo sloupce, tak

```
PRINT CHR$ 22+CHR$ 1+CHR$ c;        má přesně stejný účinek jako
PRINT AT 1,c;
```

Je tomu proto, že CHR$ 1 a CHR$ c by normálně měly jiný význam (např. pokud c=13), CHR$ 22 před ním to změní. Kontrolní znak pro TAB je CHR$ 23 a dva znaky za tím jsou užity, aby daly číslo mezi Ø a 65535, specifikující číslo v položce TAB.

```
PRINT CHR$ 23+CHR$ a+CHR$b;         má stejný účinek jako
PRINT TAB a+256*b;
```

Je možné užít POKE, aby se počítač nezastavil se zprávou "scroll?"

```
POKE 23692,255
```

po tomto příkazu se bude scrolling provádět 255x bez zastavení s otázkou scroll? Zkuste tento příklad:

```
1Ø FOR n=Ø TO 1ØØØØ
2Ø PRINT n: POKE 23692,255
3Ø NEXT n
```

a pozorujte co se bude dít na obrazovce.

Cvičení:
1. Zkuste program pro děti, který testuje jak umí násobit:

```
1Ø LET m$=""
2Ø LET a=INT (RND*12)+1:LET b=INT (RND*12)+1
3Ø INPUT (m$) "what is";(a);"*";(b);"?";c
1ØØ IF c=a*b THEN LET m$="Right.": GO TO 2Ø
11Ø LET m$="Wrong. Try again.": GO TO 3Ø
```

Pokud jsou děti vnímavé mohly by zjistit, že vlastně nemusí vůbec počítat. Např. když se jich počítač zeptá kolik je 2*3 mohou místo správné odpovědi typovat 2*3. Cesta jak tomu zabránit je číst vstupní řetězec místo čísla. Nahraďte c v řádku 3Ø c$ a v řádku 1ØØ VAL c$ a vložte řádek

```
4Ø IF c$ <> STR$ VAL c$ THEN LET m$="Typujte správně jako číslo.":GO TO 3Ø
```

To je bude zlobit. Za několik dní mohou objevit, že lze typovat v závorce jako STR$ (2*3). Aby se tomu zabránilo je třeba nahradit c$ v řádku 3Ø příkazem LINE c$.

KAPITOLA 16 - BARVY

SHRNUTÍ: INK, PAPER, FLAHS, BRIGHT, INVERSE, OVER, BORDER

Nechte běžet tento program:

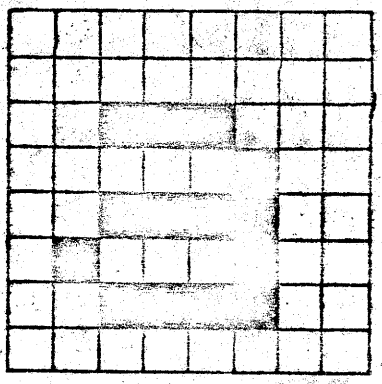
```
1Ø FOR m=Ø TO 1: BRIGHT m
2Ø FOR n=1 TO 1Ø
3Ø FOR c=Ø TO 7
4Ø PAPER c: PRINT "    ";: REM 4 barevné mezery
5Ø NEXT c: NEXT n: NEXT m
6Ø FOR m=Ø TO 1: BRIGHT m: PAPER 7
```

```
 7Ø FOR c=Ø TO 3
 8Ø INK c:PRINT c;"  ";
 9Ø NEXT c: PAPER Ø
1ØØ FOR c=4 TO 7
11Ø INK c: PRINT c;"  ";
12Ø NEXT c: NEXT m
13Ø PAPER 8: INK Ø: BRIGHT Ø
```

Program ukazuje 8 barev (vč. černé a bílé) a dvě úrovně jasu, které ZX Spectrum provádí na barevné televizi. (Pokud je vaše televize černobílá, uvidíte pouze odstíny šedi.) Zde je jejich seznam.

Ø - černá
1 - modrá
2 - červená
3 - purpurová (fialová)
4 - zelená
5 - světle modrá
6 - žlutá
7 - bílá

Na černobílé televizi jsou tyto čísla v stupně šedi. Abyste užívali barev správně, musíte pochopit jak je obraz sestaven. Obraz je rozdělen na 768 (24 řádků a 32 znaků) pozic, kde mohou být znaky tištěny.



Každý znak je tištěn jako 8x8 čtverečků, jak ukazuje obrázek malého a. Mělo by vám to připomenout uživatelskou grafiku z kapitoly 14. Znaková pozice má dvě barvy: ink = barva znaku, barva bodů v našem čtverci, paper = barva pozadí je použita pro bílé body. Na počátku práce mají všechny pozice ink černou barvu a paper barvu bílou a tak se pozice jeví jako černobílé. Pozice znaku má taky jas (normální nebo zvláštní) a možnost poznat , zda znak bliká nebo ne, takže bliká-li provádí se výměna barvy ink a paper. Vše je kodováno v číslech a tak znaková pozice má:

I. 8x8 čtverců Ø a 1 definující tvar znaku a Ø pro paper a 1 pro ink.
II. barvy ink a paper, každá z nich má kod Ø až 7.
III. jas - Ø normální, 1 jasné
IV. blýskání - Ø stálé, 1 blýskavé

Všimněte si, že ink a paper pokrývají celé pole a jak nemůžete mít víc jak dvě barvy v bloku 64 bodů. Totéž platí o jasu a blýskání, platí pro celou pozici znaku a ne pro jednotlivý bod. Čísla barvy, jasu a blýskání se nazývají atributy. Když na obrazovce něco píšete, měníte vzor bodů v této pozici, zároveň tedy měníte atributy této pozice. Nejprve si toho nevšimnete, protože je vše černé na bílém (s normálním jasem, bez blýskání), ale můžete to změnit pomocí příkazu INK, PAPER, BRIGHT a FLASH. Zkuste:

```
PAPER 5
```

a pak pište co vás napadne. Bude se psát na bleděmodrém pozadí. Dále si zkuste podobné příkazy:

```
PAPER číslo mezi Ø a 7
INK číslo mezi Ø a 7
BRIGHT Ø nebo 1
nebo FLASH Ø nebo 1
```

} Ø nic se neděje, 1 pracuje

Tiskne se v barvách podle těchto příkazů. Nyní byste měli být schopni poznat, jak pracoval program na začátku kapitoly (mezera je znak, kde barva ink a

paper je shodná). Je několik dalších čísel, která můžeme použít v těchto příkazech, a která mají obdobný účinek. 8 lze užít ve všech 4 příkazech a znamená "transparentní" ve smyslu, že se zachovají staré atributy. Předpokládejme např.:

```
PAPER 8
```

Žádný znak nebude mít nikdy barvu 8, protože taková barva neexistuje. Při stisku je barva stejná jak byla dříve. INK 8, BRIGHT 8 a FLASH 8 pracují stejným způsobem.
9 lze použít pouze u PAPER a INK a znamená kontrast. Barva, kterou použijete je v kontrastu s druhou užitou barvou, je to bílá, když druhá barva je černá, (modrá, červená, nachová). Má černou barvu, když druhá barva je světlá (zelená, cyan, žlutá, bílá). Zkuste:

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

Inpresionističtěji vypadá obrazovka, probíhá-li program s barevnými pruhy.

```
INK 9: PAPER 8: PRINT AT Ø,Ø;: FOR n=1 TO 1ØØØ: PRINT n;: NEXT n
```

Barva ink je zde vždy v kontrastu se starým paper a to v každé pozici. Barevná televize je založena na kuriozním faktu, že lidskému oku stačí zobrazit tři barvy - modrou, červenou a zelenou. Ostatní barvy jsou směsicí barev (těchto základních). Abyste poznali, jak všech 8 barev hraje dohromady, představte si obdélníky svítící modře, červeně a zeleně. Různé směsice těchto barev pak dávají barvy ostatní, jak to ukazuje program (všimněte si, že ink squares získáte užitím SHIFT a 8 v G-modu).

```
 1Ø BORDER Ø: PAPER Ø: INK 7: CLS
 2Ø FOR a=1 TO 6
 3Ø PRINT TAB 6; INK 1;"          ": REM 18 ink squares
 4Ø NEXT a
 5Ø LET dataline=2ØØ
 6Ø GO SUB 1ØØØ
 7Ø LET dataline=21Ø
 8Ø GO SUB 1ØØØ
 9Ø STOP
2ØØ DATA 2,3,7,5,4
21Ø DATA 2,2,6,4,4
1ØØØ FOR a=1 TO 6
1Ø1Ø RESTORE dataline
1Ø2Ø FOR b=1 TO 5
1Ø3Ø READ c: PRINT INK c;"      ";: REM 6 ink squares
1Ø4Ø NEXT b: PRINT : NEXT a
1Ø5Ø RETURN
```

Existuje funkce, která udá atributy v dané pozici na obrazovce. Jde o celkem komplikovanou funkci a pojednáme o ní na konci této kapitoly. Jsou další dva příkazy INVERSE a OVER , které neřídí atributy, ale vzor, který je tištěn na obrazovce. Používají se čísla Ø pro není a 1 pro je, podobně jako FLASH a BRIGHT, jsou to jediné možnosti. Provedete-li INVERSE 1, pak bodový vzor bude invertní v obvyklé formě - body papíru budou nahrazeny ink a obráceně. Pokud máte černou jako ink a bílý papír (jako po prvním zapnutí), tak po inverzi budou písmena bílá na černém pozadí.
Příkaz OVER 1 vysílá k akci zvláštní druh přepisování. Formálně, když je něco napsáno do znakové pozice, je zcela jedno, co tam bylo dříve, ale nyní nový znak překryje znak starý (podívej se na cvičení 1). Může to být zvlášť užitečné pro psaní kompozičních znaků, podobně jako znaků, na které je kladen důraz, stejně jako se v němčině píše přehlasované o. (Nejdříve NEW) pak typujte:

```
 1Ø OVER 1
 2Ø FOR n=1 TO 32
 3Ø PRINT "o"; CHR$ 8;"""";
 4Ø NEXT n
```

(Všimněte si kontrolního znaku, který vrací zpět mezeru) Existuje další způsob využití INK, PAPER, která bude pravděpodobně užitečnější. Můžete dávat položky v PRINT (následovně;) a pak co se bude provádět, to co by se dělo, kdyby šlo o příkazy, avšak účinek je dočasný, končí příkazem PRINT, který je obsahuje. Tak typujte:

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

pak jenom x bude na žlutém pozadí. INK a ostatní příkazy nemají účinek na barvy v dolní části obrazovky, kde jsou typovány příkazy INPUT. Dolní část obrazovky používá barvu pro paper a kod 9 pro kontrast v barvě INK, bez blikání s normálním jasem. Můžete změnit okraj obrazovky na kteroukoliv z 8 barev (ne 8 a 9) užitím příkazu:

BORDER barva

Když typujete data pomocí INPUT, následuje toto pravidlo pro kontrast ink na barevném papíře, ale barvu můžete změnit užitím INK na PAPER, jako v příkaze PRINT. Účinek trvá do konce příkazu nebo do dalšího příkazu INPUT.
Zkuste:

INPUT FLASH 1; INK 1;"What is your number?";n

Zde je další způsob změny barev užitím kontrolních znaků - podobně jako AT a TAB v kapitole 15.

CHR$ 16	odpovídá	INK
CHR$ 17	-"-	PAPER
CHR$ 18	-"-	FLASH
CHR$ 19	-"-	BRIGHT
CHR$ 20	-"-	INVERSE
CHR$ 21	-"-	OVER

Pak následuje 1. znak, udávající barvu kodu např.:

PRINT CHR$ 16+CHR$ 9; ... má stejný účinek jako
PRINT INK 9; ...

Těchto kontrolních znaků nemusíte užívat, používejte raději příkazy. Je však užitečné pracovat s nimi v programech. Výsledky v různých částech budou mít různou barvu, aby se odlišily od sebe, nebo pěkně vypadaly. Musíte je typovat za číslem řádku, jinak se ztrácejí. Abyste je dostali do programu musíte vstoupit z klávesnice, většinou užitím rozšířeného modu.
Čísla Ø - 7 dávají odpovídající barvu - ink, pokud je současně stlačen CAPS SHIFT, a paper, když ne. . Přesněji, když jste v E modu a stlačíte číslo (řekněme 6 pro žlutou) , pak se vloží dva znaky - prvně CHR$ 17 pro PAPER a CHR$ 6 pro žlutou barvu. Když stlačíte CAPS SHIFT dostáváte kod CHR$ 16, což znamená barvu INK místo CHR$ 17. Protože získáte 2 prvky, vymazání musíte provést tak, že dvakrát stlačíte DELETE. Po prvním stlačení se objeví? Nebojte se a stlačte DELETE znovu.
⇦ a ⇨ se mohou chovat divně, když se přes kontrolní znaky posouvá kurzor.
Stále v rozšířeném modu:

8 dává CHR$ 19 a CHR$ Ø pro normální jas
9 dává CHR$ 19 a CHR$ 1 pro zvláštní jas
CAPS SHIFT a 8 dává CHR$ 18 a CHR$ Ø pro blýskání
CAPS SHIFT a 9 dává CHR$ 18 a CHR$ 1 pro blýskání

Zde jsou dvojice v obyčejném L modu:

CAPS SHIFT a 3 dává CHR$ 2Ø a CHR$ Ø pro normální znaky
CAPS SHIFT a 4 dává CHR$ 2Ø a CHRS 1 pro inverzní znaky

Ke shrnutí je podrobný popis horní řady klávesnice. (Viz strana 87 anglického originálu, nebo tabulka na následující straně.)
Funkce ATTR má formu (řádek, sloupec)
Dva argumenty jsou čísla řádku a sloupce, které byste užili v příkazu AT a výsledkem je číslo ukazující barvy odpovídajícího znaku na obrazov.Tohc můžete využít samostatně nebo jako část jiné funkce. Číslo, které je výsledkem je součet 4 dalších čísel:
128 pokud pozice blýská, Ø když je stálá
64 pokud je pozice jasná, Ø když je normální
8 x je dáno kodem barvy papíru
kod barvy ink.
Např. když je pozice blýskavá s normálním jasem, žlutým papírem a modrým inkoustem. Pak čísla, která musíte sečíst jsou 128,Ø,8x6=48 a 1 dostaneme 177.
Zkuste toto:

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

Cvičení:
Zkuste

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

MÓD SHIFT

MÓD	SHIFT										
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 žlutá	ink bílá	flash ne	flash ano	ink černá
	ŽÁDNÝ	papír modrá	papír červená	papír purpur	papír zelená	papír sv. modrá	papír žlutá	papír bílá	normální jas	extra jas	papír černá
G	EITHER									EXIT GRAPHICS	DELETE
	ŽÁDNÝ									EXIT GRAPHICS	DELETE
K,L or C	CAPS	EDIT	CAPS LOCK	TRUE VIDEO	INVERSE VIDEO					GRAPHICS MOD	DELETE
	SYMBOL	!		#	£	%	&	'	(	)	-
	ŽÁDNÝ	1	2	3	4	5	6	7	8	9	ø

kde / má prožíznout B a má vlevo bílý čtverec. Je to způsob, jak na ZX Spectru přepisovat. Dva papíry nebo dva inkousty dají papír, jeden z nich dává inkoust. Je zajímavé, že pokud přetisknete totéž 2x, dostanete zpět to, co tam bylo. Zkuste nyní natypovat:

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

2. Typujte

```
PAPER Ø: INK Ø
```

Není dobré, že to neplatí v dolní části obrazovky? Nyní typujte:

```
BORDER Ø
```

a vidíte, jak dobře pracuje počítač.

3. Nechte proběhnout tento program:

```
1Ø POKE 22527+RND*7Ø4,RND*127
2Ø GO TO 1Ø
```

Nikdo nepozná jak to pracuje. Mění se barvy na obrazovce, RND zajišťuje náhodnost provádění. Diagonální prvky, které momentálně vidíte jsou dokladem toho, že RND dává pseudonáhodné číslo.

4. Natypujte nebo nahrejte program šachových figurek z kapitoly 14 a pak typujte tento program, který sestaví diagram šachových pozic

```
  5 REM prázdná šachovnice
 1Ø LET bb=1: LET bw=2: REM červená a modrá šachovnice
 15 PAPER bw: INK bb: CLS
 2Ø PLOT 79,128:REM hranice
 3Ø DRAW 65,Ø:DRAW Ø,-65
 4Ø DRAV -65,Ø:DRAV Ø,65
 5Ø PAPER bb
 6Ø REM board
 7Ø FOR n=Ø TO 3: FOR m=Ø TO 3
 8Ø PRINT AT 6+2*n, 11+2*m;" "
 9Ø PRINT AT 7+2*n, 1Ø+2*m;" "
1ØØ NEXT m: NEXT n
11Ø PAPER 8
12Ø LET pw=6: LET pb=5: REM černá a bílá barva figurek
2ØØ DIM b$(8,8): REM pozice figurek
2Ø5 REM initalizace figurek
21Ø LET b$(1)="rnbqkbnr"
22Ø LET b$(2)="pppppppp"
23Ø LET b$(7)="PPPPPPPP"
24Ø LET b$(8)="RNBQKBNR"
3ØØ REM zobraz pole
31Ø FOR n=1 TO 8: FOR m=1 TO 8
32Ø LET bc=CODE b$(n,m): INK pw
325 IF bc=CODE " " THEN GO TO 35Ø: REM space
33Ø IF bc > CODE "Z" THEN INK pb: LET bc=bc-32: REM nižší úroveň černé
34Ø LET bc=bc+79: REM změna na grafiku
35Ø PRINT AT 5+n, 9+m; CHR$ bc
36Ø NEXT m: NEXT n
4ØØ PAPER 7: INK Ø
```

KAPITOLA 17 - GRAFIKA

SHRNUTÍ: PLOT, DRAW, CIRCLE, POINT

V této kapitole uvidíte, jak na ZX Spectru kreslit obrázky. Část obrazovky, kterou můžete použít má 22 řádků 32 sloupců, tj. 22*32=7Ø4 znaků. Jak si pamatujete z kapitoly 16, každý z těchto znaků je vytvořen z 8*8čtvercových bodů a ty se nazývají "pixels"(obrazové elementy). Pixel je specifikován dvěma čísly - souřadnicemi. První souřadnice je X a druhá je Y. Tyto souřadnice se píšou jako pár v závorce (Ø,Ø), (Ø,175), (255,Ø),(255,175) jsou rohovébody kreslicího pole. Příkaz:

PLOT x souřadnice, y souřadnice kreslí bod daný souřadnicemi . Program:

```
1Ø PLOT INT (RND*256), INT (RND*176): INPUT a$: GO TO 1Ø
```

kreslí náhodně bod, stlačíte-li ENTER. Zde je mnohem zajímavnější program. Kreslí grafy funkce SIN (sinusovka) pro hodnoty Ø až 2 PI.

```
1Ø FOR n=Ø TO 255
2Ø PLOT n,88+8Ø*SIN (n/128*PI)
3Ø NEXT n
```

Další program kreslí graf SQR (část paraboly) mezi Ø a 4:

```
1Ø FOR n=Ø TO 255
2Ø PLOT n,8Ø*SQR (n/64)
3Ø NEXT n
```

Všimněte si, že souřadnice bodu se liší od souřadnice v AT. Zjistíte, že diagram z kapitoly 15 je užitečný, když pracujete se souřadnicemi řádku a sloupce. Aby vám počítač při kreslení pomohl, bude přímo tisknout čáry, kružnice a části kruhu s použitím příkazu DRAW a CIRCLE. Příkaz DRAW kreslí přímku ve formě

```
DRAW x, y
```

Startovacím místem přímky je bod, kde byl poslední PLOT, DRAW, CIRCLE, RUN, CLEAR, CLS a NEW to mění na spodní levý roh (Ø,Ø) a končí bodem udanými souřadnicemi. Zkoušejte příkazy PLOT a DRAW např.

```
PLOT Ø,1ØØ: DRAW 8Ø,-35
PLOT 9Ø,15Ø: DRAW 8Ø,-35
```

Všimněte si, že čísla v DRAW mohou být záporná, ačkoliv v příkazu PLOT tomu tak nebylo. Můžete samozřejmě tisknout v barvách, pamatujte však, že barvy pokrývají celou pozici znaku a nemohou být specifikovány jako jednotlivé body. Bod je tištěn barvou INK a celá znaková pozice je dána touže barvou, jak předvádí tento program:

```
1Ø BORDER Ø: PAPER Ø: INK 7: CLS : REM  černá zmizela z obrazovky
2Ø LET x1=Ø: LET y1=Ø: REM počátek přímky
3Ø LET c=1: REM  pro INK barvu - začíná se modrou
4Ø LET x2=INT (RND*256): LET y2=INT (RND*176): REM  dokončení
5Ø DRAW INK c;x2-x1,y2-y1
6Ø LET x1=x2: LET y1=y2: REM  další čára začíná tam, kde minulá končila
7Ø LET c=c+1: IFc=8 THEN LET c=1: REM  nová barva
8Ø GO TO 4Ø
```

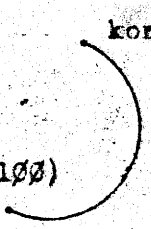
Jak postupuje program, čáry se zdají širší a to proto, že mění barvy. Pamatujte, že můžete použít PAPER, INK, FLASH, BRIGHT, INVERSE, OVER v příkazu PLOT a DRAV, stejně jako v PRINT a INPUT. Další možnost využití DRAW je možnost kreslit část kruhu místo přímek, přidáním dalšího čísla k příkazu, který specifikuje úhel otáčení.

```
DRAW x, y, a
```

x a y udávají polohu koncového bodu (jako dříve) a je počet radiánů udávající otáčení - pokud je kladné otáčí se doleva, pokud je záporné tak doprava. Jiná možnost jak se dívat na a je, že je to část úplné družnice - celá kružnice je 2PI radiánů a pak když a = PI bude nakreslena půlkružnice, pokud a = Ø,5 * PI bude to čtvrt kruhu atd. Předpokládejme, že a=PI. Pak pro libovolné x a y se nakreslí:

```
1Ø PLOT 1ØØ,1ØØ:DRAW 5Ø,5Ø,PI
```

nám nakreslí



Spusťte program několikrát a PI pokaždé nahraďte jiným číslem:

```
-PI, PO/2, 3*PI/2, PI/4, 1,Ø
```

Poslední příkaz této kapitoly je CIRCLE , který kreslí celé kružnice. Specifikujte střed kruhu a jeho poloměr užitím

```
CIRCLE x - souřadnice, y - souřadnice, poloměr
```

Podobně jako PLOT a DRAW, můžete použít různé barvy.

Funkce POINT nám říká, zda bod má barvu ink nebo paper. Má dva argumenty, souřadnice bodu (musí být v závorkách) a výsledek je Ø, když bod má barvu papíru a 1, pokud jde o barvu ink. Zkuste:

```
CLS : PRINT POINT (Ø,Ø): PLOT Ø,Ø: PRINT POINT (Ø,Ø)   pak typujte:
PAPER 7: INK Ø
```

a zkuste jak pracuje INVERSE, OVER, PLOT. Mají vliv jenom na vlastní bod a ne na zbytek pozice znaku. Normální hodnota v příkazu PLOT je Ø. Změňte ji na 1 a podívejme se co se stane:

PLOT INVERSE 1; - kreslí bod, v barvě papíru
PLOT OVER 1; - změní bod z čehokoliv - přehodí se barva ink a paper
PLOT INVERSE1; OVER 1; nechává bod jak byl dříve, ale pamatujte, že mění pozici PLOT a tak toho můžete užít

Jako další příklad užití OVER vyplňte obrazovku černobíle a typujte:

PLOT Ø,Ø: DRAW 1; 255,175

program bude kreslit decentní čáru, v mezerách i v textu. Nyní proveďte znovu totéž a čára zmizí. . Je to výsledek OVER 1. Další příklad:

PLOT Ø,Ø: DRAW 255,175 a smazání je provedeno
PLOT Ø,Ø: DRAW INVERSE 1;255,175 ale takto můžete rovněž smazat část textu

Nyní zkuste:

PLOT Ø,Ø: DRAW OVER 1;25Ø,175 a zkuste kreslit pomocí
DRAW OVER 1;-25Ø,-175

Nepracuje to přesně, protože body, které čára pužívá na cestě zpět nejsou stejné jako body tam. Čáru musíte vymazat přesně ve směru v jakém jste ji kreslili. Jeden ze způsobů jak získat neobvyklé barvy je užitím uživatelské grafiky. Natypujte tento program a spusťte ho:

```
1ØØØ FOR n=Ø TO 6 STEP 2
1Ø1Ø POKE USR"a"+n,BIN Ø1Ø1Ø1Ø1: POKE USR "a"+n+1, BIN 1Ø1Ø1Ø1Ø
1Ø2Ø NEXT n
```

což dává uživatelskou grafiku šachovnicového vzoru. Tisknete-li znak (grafický mod a pak a) s červeným ink na žlutém paper, je výsledná barva oranžová.

Cvičení:

1. Procvičte položky příkazů PAPER, INK, BRIGHT v příkazu PLOT. Ty jsou částí znakové pozice obsahující bod. Normálně je to přes příkaz PLOT:

   PLOT PAPER 8; FLASH 8; BRIGHT 8; ...

   a pouze barva ink je změněna, pokud něco tisknete můžete ji změnit libovolně. Buďte opatrní, užíváte-li barvy s INVERSE 1, ukazuje to barvu papíru, ale mění barvu ink a to možná nečekáte.

2. Pokuste se nakreslit kruhy s užitím SIN a COS. Typujte a spusťte:

   ```
   1Ø FOR n=Ø TO 2*PI STEP PI/18Ø
   2Ø PLOT 1ØØ+8Ø*COS n,87+8Ø*SIN n
   3Ø NEXT n
   4Ø CIRCLE 15Ø,87,8Ø
   ```

   Jak vidíte, příkaz CIRCLE je mnohem rychlejší, ale méně přesný.

3. Zkuste:

   CIRCLE 1ØØ,87,8Ø: DRAW 5Ø,5Ø

   vidíte, že příkaz CIRCLE nechává pozici PLOT v neurčitém místě - někde uprostřed pravého kraje kružnice. K dalšímu kreslení musíte obvykle užít PLOT k definování výchozí pozice.

4. ZDE je program, který kreslí grafy téměř jakékoliv funkce. Prvně se zeptá na číslo n a kreslí graficky hodnoty -n až +n. Pak se zastaví pro funkci, která se vkládá jako řetězec. Měl by to být výraz používající x jako argumen funkce.

   ```
   1Ø PLOT Ø,87: DRAW 255,Ø
   2Ø PLOT 127,Ø: DRAW Ø,175
   3Ø INPUT s,e$
   35 LET t=Ø
   4Ø FOR f=Ø TO 255
   5Ø LET x=(f-128)*s/128: LET y=VAL e$
   6Ø IF ABS y  87 THEN LET t=Ø GO TO 1ØØ
   7Ø IF NOT t THEN PLOT f,y+88: LET t=1: GO TO 1ØØ
   8Ø DRAW 1,y-old y
   1ØØ LET old y=INT (y+.5)
   11Ø NEXT f
   ```

   Nechte program běžet a jako příklad natypujte v řádce 1Ø číslo n a 1Ø*TAN x pro funkci. Kreslí to graficky tangens v rozsahu -1Ø až +1Ø.

KAPITOLA 18 - POHYB

SHRNUTÍ: PAUSE, INKEY$, PEEK

Dost často budete chtít, aby program běžel po stanovenou dobu a pak bude příkaz PAUSE užitečný.

PAUSE n

zastaví počítání a zobrazí výpočet po dobu n cyklů televize (50 cyklů/sec v Evropě a 60 v Americe) a může být do 65535, což dává něco pod 22 minut. n=Ø znamená trvalá PAUSE. PAUSE muže být přerušena stlačením tlačítka (pamatuj, že CAPS SHIFT a mezera program přeruší). Tento program ukazuje vteřinovou ručičku hodin:

```
1Ø REM první uděláme ciferník
2Ø FOR n=1 TO 12
3Ø PRINT AT 1Ø-1Ø*COS (n/6*PI),16+1Ø*SIN (n/6*PI);n
4Ø NEXT n
5Ø REM  nyní nastartujeme hodiny
6Ø FOR t=Ø TO 2ØØØØØ: REM t je čas ve vteřinách
7Ø LET a=t/3Ø*PI : REM  a je úhel vteřinové ručičky v radiánech
8Ø LET sx=8Ø*SIN a: LET sy=8Ø*COS a
2ØØ PLOT 128,88: DRAW OVER 1;sx,sy: REM kreslí vteřinovou ručičku
21Ø PAUSE 42
22Ø PLOT 128,88: DRAW OVER 1;sx,sy: REM maže vteřinovou ručičku
4ØØ NEXT t
```

Hodiny půjdou asi 55,5 hodin, což je dáno řádkem 6Ø, ale můžete to snadno změnit. Všimněte si, jak je čas na řádku 21Ø kontrolován. Mohli byste očekávat PAUSE 5Ø, ale výpočet také nějaký čas zabere. Údaj v PAUSE budete muset zřejmě opravit podle chyby, kterou hodiny vykazují, porovnávají-li se s reálným časem. (Hodiny jdou s přesností asi 2 %, což, což na den může činit odchylku až 1 hodinu.). Zde je přesnější způsob změření času. Užívá obsah určité části paměti. Uchované data se vybírají pomocí PEEK. V kapitole 25 bude podrobné vysvětlení: Vyjádření užití je

(65536*PEEK 23674+256*PEEK 23673+PEEK 23672)/5Ø

Dává to počet vteřin od zapnutí počítače (po 3 dny a 21 hodin, pak se mění na Ø a počítá dále). Zde je upravený hodinový program, který toho využívá:

```
1Ø REM nejdříve uděláme číselník
2Ø FOR n=1 TO 12
3Ø PRINT AT 1Ø-1Ø*COS (n/6*PI),16+1Ø*SIN (n/6*PI);n
4Ø NEXT n
5Ø DEF FN t()=INT((65536*PEEK 23674+256*PEEK 23673+PEEK 23672)/5Ø):REM
                počet vteřin od počátku
1ØØ REM nyní nastartujeme hodiny
11Ø LET t1=FN t()
12Ø LET a=t1/3Ø*PI : REM  a je úhel vteřinové ručičky v radiánech
13Ø LET sx=72*SIN a: LET sy=72*COS a
14Ø PLOT 131,91: DRAW OVER 1; sx,sy: REM  nakreslí ručičku
2ØØ LET t=FN t()
21Ø IF t<=t1THEN GO TO 2ØØ: REM  počkej, až je čas na další ručku
22Ø PLOT 131,91: DRAW OVER 1;sx,sy: REM  maže starou ručičku
23Ø LET t1=t: GO TO 12Ø
```

Vnitřní hodiny používají tuto metodu mají přesnost asi Ø,Ø1 %. Je to asi 1Ø vteřin za den, hodiny však dočasně zastaví použijete-li příkazu BEEP nebo operace s magnetofonem, tiskárnou nebo dalším zvláštním vybavením. Tyto operace znamenají ztrátu času. Čísla PEEK 23674, PEEK 23673 a PEEK 23672 jsou drženy v paměti počítače a jsou užity pro počítání v 5Ø-nách vteřiny. Každé je v rozmezí od Ø do 255, po 255 se vrací na Ø. Nejčastěji se mění číslo v PEEK 23672. Vždy za 1/5Ø vteřiny se zvětší o 1. Když je 255 tak se změní na Ø a ve stejném času se mění číslo v PEEK 23673 a Ø na 1. Když uplyne 256/5Øs, PEEK 23673 má hodnotu 255 a mění se na Ø. Zároveň číslo v PEEK 23674 se mění z Ø na 1. To by mělo dostatečně vysvětlit jak čas pracuje. Nyní pečlivě uvažujte. Předpokládejme naše tři čísla: Ø (pro PEEK 23674), 255 (pro PEEK 23673) a 255 (pro PEEK 23672). To znamená, že je 21 minut po zapnutí počítače-

-náš výraz je:

(65536*Ø+256*255+255)/5Ø=131Ø.7

Ale je zde nebezpečí. Další čas je 1/5Ø s a čísla se mění na 1,Ø a Ø. To se stane vždy, když jste v polovině provedení výrazu, počítač by měl uvažovat PEEK 23674 jako Ø, ale pak mění další dva čísla na Ø, předtím než vybere. Odpověď by pak mohla být

(65536*Ø+256*Ø+Ø)/5Ø=Ø což je pochopitelně chybné

Jednoduché pravidlo umožní vyhnout se tomuto problému. Stačí počítat $2x^2$ vzít další výsledek. Podíváte-lise pečlivě na program,uvidíte, že to dělá implicitně. Zde je tisk pro uvedenou aplikaci. Definujte

```
1Ø DEF FN m(x,y)=(x+x+ABS(x-y))/2: REM  větší z  x a y
2Ø DEF FN u()=(65536*PEEK 23674 + 256*PEEK 23673+PEEK 23672)/5Ø:REM
                čas může být chybný
3Ø DEF FN t()=FN m(FNu(),FN u()): REM    správný čas
```

Tři počáteční čísla můžete vyměnit, aby dala reálný čas a ne čas od doby zapnutí počítače. Např. nastavíme čas na 10 dopoledne - provedeme takto:

1Ø*6Ø*6Ø*5Ø=18ØØØØØ=65536*27+256*119+64

K nastavení tří čísel na 27, 119, 64 provedete:

```
POKE 23674,27: POKE 23673,119: POKE 23672,64
```

V zemích s frekvencí sítě 6Ø Hz musíme počítat s 6Ø.

Funkce INKEY$ (nemá argument) čte klávesnici. Pokud stlačíte 1 tlačítko (nebo SHIFT a jedno tlačítko), pak výsledkem je znak, klíč je v L-modu, jinak je výsledkem prázdný řetězec. Tento program pracuje jako psací stroj.

```
1Ø IF INKEY$ <> ""THEN GO TO 1Ø
2Ø IF INKEY$= "" THEN GO TO 2Ø
3Ø PRINT INKEY$;
4Ø GO TO 1Ø
```

Řádek 1Ø čeká, až dáte prst pryč z klávesnice a řádek 2Ø čeká až stlačíte nové tlačítko. Pamatujte, že na rozdíl od INPUT, INKEY$ na vás nečeká. Netypujte ENTER, ale pokud zase netypujete nic, tak ztrácíte šanci.

Cvičení:

1. Co se stane vypustíte-li řádek 1Ø v minulém programu?
2. Další způsob užití INKEY$ je se spojením s PAUSE. Psací stroj přerušovaný.

```
1Ø PAUSE Ø
2Ø PRINT INKEY$;
3Ø GO TO 1Ø
```

   Po provedení příkladu - proč pauza nekončí stlačením tlačítka a kdy začíná?
3. Natypujte program vteřinové ručičky, aby ukazovala minuty a hodiny vždy po minutě. Pokud se cítíte ambiciozní, proveďte to tak, aby každou 1/4 hod. zazněla nějaká legrace - můžete prodádovat melodii pomocí BEEP (např. zvonění BIG BEN - ukazuje to další kapitola.
4. (Pro sadisty) Zkuste:

```
1Ø IF INKEY$="" THEN GO TO 1Ø
2Ø PRINT AT 11,14;"OUCH!"
3Ø IF INKEY$ <> "" THEN GO TO 3Ø
4Ø PRINT AT 11,14;" "
5Ø GO TO 1Ø
```

KAPITOLA 19 - BEEP (PÍPÁNÍ)

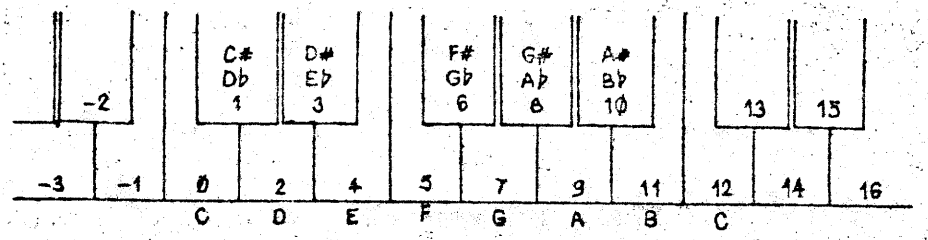
SHRNUTÍ: BEEP

Jak jste již slyšeli, má ZX Spectrum zabudovaný reproduktor. Mohli jste si o tom přečíst v úvodní knize. Reproduktor zní, použijeme-li příkaz BEEP.

```
BEEP trvání, ladění
```

kde jako obvykle trvání a ladění jsou numerické výrazy. Trvání je ve vteřinách, ladění v půltónech nad středním C. Pro noty nižší se užívají záporná čísla.

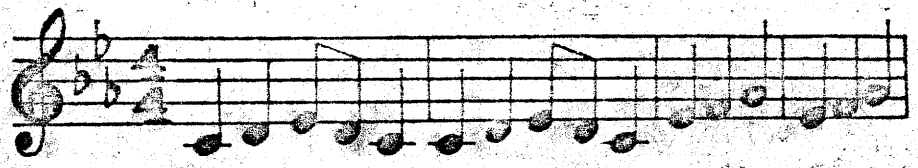
Zde je diagram, ukazující hodnoty všech not jedné oktávy na piánu.



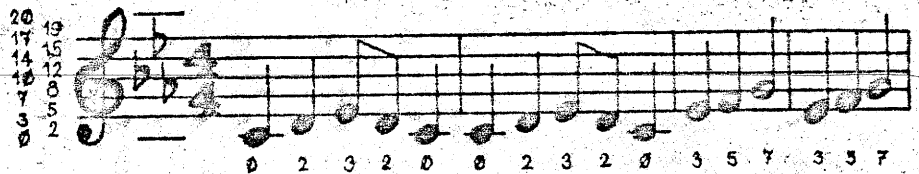
K získání vyšších nebo nižších not musíme přidat nebo odečíst 12 pro každou oktávu nahoru i dolů. Pokud máte při programování vedle sebe piáno, bude pravděpodobně vše, co potřebujete k práci s laděním tento diagram. Pokud je Vám psaná hudba cizí, radíme vám nakreslit si diagram s hodnotami v každé lince a mezeře v určitém klíči. Např. typujte:

```
1Ø PRINT "Frere Gustav"
2Ø BEEP 1,Ø: BEEP 1,2: BEEP .5,3: BEEP .5,2: BEEP 1,Ø
3Ø BEEP 1,Ø: BEEP 1,2: BEEP .5,3: BEEP .5,2: BEEP 1,Ø
4Ø BEEP 1,3: BEEP 1,5: BEEP 2,7
5Ø BEEP 1,3: BEEP 1,5: BEEP 2,7
6Ø BEEP .75,7: BEEP .25,8: BEEP .5,5: BEEP .5,3: BEEP .5,2: BEEP 1,Ø
7Ø BEEP .75,7: BEEP .25,8: BEEP .5,7: BEEP .5,5: BEEP .5,3: BEEP .5,2:
   BEEP 1,Ø
8Ø BEEP 1,Ø: BEEP 1,-5: BEEP 2,Ø
9Ø BEEP 1,Ø: BEEP 1,-5: BEEP 2,Ø
```

Když program spustíte, měli byste slyšet pohřební pochod z Mahlerovy první symfonie, kousek, kde skřítci pochovávají muže z americké kavalérie. Předpokládejte, že vaše melodie je psaná v klíči C mol, podobně jako Mahlerova. Počátek vypadá takto:



a můžete psát hodnoty not takhle:



Pro názornost jsme noty rozepsali do dvou řádků. Všimněte si, že snížené E nemá účinek pouze v lince, kde je psáno, tam snižuje 16 na 15, ale má stejný účinek v dolní řádce, kde mění 4 na 3. Též by mělo být zcela snadné nalézt hodnotu jakékoliv noty. Pokud chcete změnit stupnici, je nejlépe vložit hodnotu "klíč" před každou hodnotu, tak tedy:

```
2Ø BEEP 1,key+Ø: BEEP 1,key+2: BEEP .5,key+3: BEEP .5,key+2: BEEP 1,key+Ø
```

Před spuštěním programu musíte dát vhodnou hodnotu pro "klíč". O pro C mol, 12 pro C mol zvednuté o jednu oktávu, atd. Můžete přinutit počítač, aby zněl jako jiný nástroj, když "klíč" je ve formě zlomku. Rovněž musíte pracovat na délce not. Protože se zde jednalo o pomalou skladbu, volili jsme 1 vteřinu pro čtvrtovou notu, a založili na tom pravidlo, že např. osminová nota zní 1/2 vteřiny, atd. Mnohem pružnější je užít proměnnou "crotchet" tak, aby měla délku čtvrteční noty a pak specifikovat její trvání. Tak řádek 2Ø by nyní vypadal:

```
2Ø BEEP crotchet,key+Ø: BEEP corotchet,key+2: BEEP crotchet/2,key+3:
   BEEP crotchet/2,key+2: BEEP crotchet,key+Ø
```

Asi budete chtít použít kratší jména pro crotchet a key. Nastavením crotchet na různé hodnoty můžete měnit snadno rychlost hraní. Uvědomte si, že v počítači je pouze jeden reproduktor a tak můžete hrát pouze jednu notu, jste omezeni na melodie neharmonické! Pokud vám to nestačí, tak si k tomu zpívejte. Zkuste programovat svoje melodie, začněte třeba zcela jednoduchou, jako jsou "Tři slepé myši". Pokud nemáte piáno a neznáte noty, pořiďte si jednoduchý nástroj jako je píšťala (zobcová) a tvořte melodie. Počítač vytvoří každý tón jako tento nástroj. Typujte:

```
FOR n=Ø TO 1ØØØ: BEEP .5,n: NEXT n
```

Bude to hrát noty až nejvýše a pak se počítač zastaví se zprávou: B integer out of range (číslo je mimo rozsah). Zkuste totéž dolů pro nízké tóny. Nejnižší noty budou znít jako klepání, ve skutečnosti se vyšší tóny tvoří stejně, ale rychleji, takže to klepání neslyšíte. Pouze uprostřed jsou tóny vhodné pro hudbu. Nízké tóny slyšíme jako klepání, vyšší jsou tvrdé a slabé; typujte tento program:

```
1Ø BEEP .5,Ø: BEEP .5,2: BEEP .5,4: BEEP .5,5: BEEP .5,7: BEEP .5,9:
   BEEP .5,11: BEEP .5,12: STOP
```

Příklad hraje stupnici C dur, která používá všechny bílé klávesy na pianu od středního C po následné C. Stupnice zní stejně jako na pianu a nazývá se temperovaná, protože interval půltónu je stejný po celou dobu. Houslista by hrál stupnici odlišně, tak, že by tóny zněly pro ucho příjemněji. Můžete to udělat tím, že jemně sunete prsty po struně, což klavíristu nemůže. Základní stupnice, kterou hraje houslista je asi následující:

```
2Ø BEEP .5,Ø: BEEP .5,2.Ø39: BEEP .5,3.86: BEEP .5,4.93: BEEP .5,7.Ø2:
   BEEP .5,8.84: BEEP .5,1Ø.88: BEEP .5,12: STOP
```

Možná, že nebudete schopni poznat rozdíl mezi stupnicemi. První rozdíl je v tom, že třetí tón je nižší než v přirozené ladící stupnici. Pokud jste specialista, můžete v této tónině programovat. Nevýhodou je, že když to pracuje dobře v tónině C, tak v některých stupnicích to lze udělat jen ztěží. V temperované stupnici to však můžeme použít v jakémkoliv klíči. V počítači je zde problém, ale můžete užít trik s proměnným klíčem "key". Některá hudba, zvláště indiánská, užívá intervaly menší, než je část tónu. Můžete to programovat s užitím BEEP bez problémů, např. čtvrtton nad hodnotou C má hodnotu .5. Můžete zařídit, aby klávesnice místo cvakání pípala:

```
POKE 236Ø9,255
```

Druhé číslo určuje délku "beepu" (zkuste hodnoty mezi Ø a 255). Když je Ø, pak příkaz zní jako krátký cvak. Pokud chcete slyšet kvalitnější zvuky, musíte použít jiný než vestavěný reproduktor. Signál je přítomen v zásuvce MIC a EAR. Vyšší úroveň je v EAR, ale jinak je signál stejně kvalitní. Můžete zkusit připojit ke Spectru sluchátka; vypíná se reproduktor. Další možnost je použít zesilovač s repros stavou a s výsledným zvukem budete jistě spokojeni. Tóny můžete samozřejmě nahrávat na magnetofon a pak společně přehrávat se Spectrem - dvoujhlasná melodie. Musíte vyzkoušet, na kterém výstupu bude lepší signál, abyste byli spokojeni - experimentujte!

Cvičení:

1. Přepište Mahlerův program s použitím smyčky FOR - NEXT k opakování. Zkuste naprogramovat celou Mahlerovu první symfonii, nejen pohřební pochod.

KAPITOLA 20 - NAHRÁVÁNÍ PAMĚTI

SHRNUTÍ: LOAD, SAVE, VERIFY, MERGE

Základní metody užití SAVE, LOAD, VERIFY byly popsány v úvodní knize. Tuto kapitolu byste měli číst až pak. Viděli jsme, že LOAD ruší v počítači starý program a proměnné. Zde je další příkaz MERGE, který to nedělá, ruší pouze řádky se stejným číslem a stejně označené proměnné - zůstává hodnota uvedená v MERGE. Natypujte program kostky z kapitoly 11 a uchovejte ho na pásce jako "dice". Nyní natypujte:

```
 1 PRINT 1
 2 PRINT 2
1Ø PRINT 1Ø
2Ø LET x=2Ø
```

a pak pokračujte jako při ověřování, ale nahraďte VERIFY "dice" MERGE "dice". Pokud si pak prolistujete program, vidíte, že řádky 1 a 2 přibyly, ale řádky 1Ø a 2Ø byly nahrazeny novými a rovněž x (zkuste PRINT x). Nyní uvidíte 4 prosté formy příkazů použitých pro magnetofon.

SAVE - nahrává program a proměnné na pásek
VERIFY - kontroluje program a proměnné na pásku proti těm, které jsou v počítači
LOAD - čistí počítač (starý program a proměnné) a nahrazuje ho novým, který čte z pásku
MERGE - je podobné LOAD s výjimkou toho, že neruší starý program a proměnné, pokud jsou řádky jinak očíslovány a proměnné mají jiné označení.

V každém z těchto příkazů je klíčové slovo následováno řetězcem. Pro SAVE je tam jméno programu na pásku, zatímco pro další 3 říká počítači, který program vybrat. Během hledání se tisknou jména programů na které se narazilo. Identifikace je dvojitá.
Pro VERIFY, LOAD a MERGE můžete použít prázdný řetězec jako jméno, které hledáte. Pak se počítač nezajímá o jméno, ale bere první program, na který narazí. Možná varianta příkazu SAVE je tato:

SAVE řetězec, LINE číslo

Program je uchován tak, že při zpětném čtení pomocí LOAD se automaticky skáče na řádek s udaným číslem, odkud začíná program pracovat. Zatím jediným druhem informací, které jsme uchovávali na pásku byly programy společně s jejich proměnnými. Zde jsou další, zvaná pole a bytes.
S poli se zachází trochu odlišněji. Pole můžete uchovat za použití DATA v příkazu SAVE:

SAVE řetězec, DATA jméno pole ()

Řetězec je jméno o uchování informace na pásce a pracuje stejně jako když uchováváme program nebo byte. Jméno pole specifikuje pole, které chcete uchovat, jde o jedno písmeno nebo písmeno a %. Pamatujte na závorky vzadu. Možná si myslíte, že jsou logicky zbytečné, ale počítači to ulehčuje práci. Musí vám být jasný rozdíl mezi řetězcem a jménem pole. Pokud např. provedete

SAVE "Bloggs" DATA b()

pak příkaz SAVE bere pole b a ukládá ho na pásek pod jménem "Bloggs". Když natypujete

VERIFY "Bloggs" DATA b()

počítač bude hledat číselné pole uložené na pásku pod jménem "Bloggs" (když počítač nalezne, oznámí: Number array: Bloggs) a kontroluje ho zpětně proti poli b v počítači.

LOAD "Bloggs" DATA b()

hledá pole na pásce a pak, pokud je pro něj v počítači místo, zruší v počítači pole b, pokud existuje a nahraje nové pole b. MERGE nemůžete užít při uchování polí. Můžete uchovat znak (řetězec)-ová pole v téže odobě. Když počítač hledá na pásce a najde jedn z tohoto pole napíše: CHaracter array a jméno. Když nahráváte do paměti znakové pole, neruší to pouze znakové pole s tímto jménem, ale taky řetězec s tímto jménem.
Uchování byte je použité pro informaci bez ohledu na další použití - mohl by to být televizní obraz, uživatelem definovaná grafika, ... Je to provedeno s užitím slova CODE jako v

SAVE "picture" CODE 16384,6912

Jednotkou uchování paměti je byte (číslo mezi Ø a 255), každý byte má svou adresu (číslo mezi Ø a65535). První číslo po CODE je adresa prvého byte, který se má nahrát na pásku a druhé je počet byte, které se nahrají . V našem případě je 16384 adresa prvého byte v poli 'display' (které obsahuje televizní obraz) a 6412 je počet byte v něm obsažených. Tak uchováme kopii televizní obrazovky - zkuste to. Jméno "picture" je jméno, podobně jako u programu. K nahrání zpět:

LOAD "picture" CODE můžete dát čísla po CODE ve formě:
LOAD jméno CODE začátek, délka

Zde je délka pouze jako měřítko, když počítač zjistí , že na pásku má správné jméno, odmítne ho, pokud je větší než specifikace - jinak by přepsal něco, co přepsat v úmyslu neměl. Dává chybové hlášení: R Tape loading error.

Pokud se délka vynechá, počítač bude číst byte, ať jsou dlouhé jakkoliv. Začátek ukazuje adresu, kam se má uložit první byte, může být rozdílný od adresy v příkazu SAVE. Pokud jsou adresy stejné, může se vynechat. CODE 16384, 6912 je tak užitečný, že ho můžete použít v jednoduchší formě - nahradit ho SCREEN$ - např.:

SAVE "picture"SCREEN$
LOAD "picture"SCREEN$

Zde je řídký příklad, kdy VERIFY nebude pracovat, VERIFY napíše jméno, které najde na pásce a tak při verifikaci, kdy je obraz změněn, dochází k verifikační chybě. Ve všech jiných případech byste měli užít příkazu VERIFY, pouužijete-li SAVE. Dále je kompletní seznam čtyř příkazů užitých v této kapitole. Jméno je jakékoliv vyjádření, odpovídá jménu, pod kterým je informace uchovávána na pásce. Mělo by se skládat z ASCII znaků, avšak jich může být maximálně 10. Jsou 4 druhy informací, které mohou být uchovány na pásku: program a proměnné dohromady, číselná pole, řetězcová pole, přímo byte. Když VERIFY, LOAD a MERGE hledají informace na pásce s uvedeným jménem, pak se na obrazovce tisknou jména informací, které našly. Zpráva je ukázána jako: program, number array , character array , bytes. Pokud jméno byl prázdný řetězec, vezme první část informací bez ohledu na její jméno.

SAVE uchovává informaci na pásku pod daným jménem. Objeví se chyba F, když jméno je prázdné nebo má 11 a více znaků. SAVE vždy vyvolá zprávu: Start tape (začni nahrávat), then press any key (pak stlač jakékoliv tlačítko), a čeká až je tlačítko stlačeno:

1. PROGRAM A PROMĚNNÉ
   SAVE jméno LINE číslo řádku
   uchovává program tak, že LOAD automaticky následuje GO TO číslo řádku.
2. BYTES
   SAVE jméno CODE začátek, délka
   chrání délku byte, začíná na uvedené adrese
   SAVE jméno SCREEN$ je vlastně
   SAVE jméno CODE 16384,6912 a uchovává televizní obraz.
3. POLE
   SAVE jméno DATA znak () nebo
   SAVE jméno DATA znak $ ()
   uchovává pole jehož jméno je písmeno nebo písmeno a $ (nemusí mít vztah ke jménu).

VERIFY - kontrolují informaci na pásce proti informaci v počítači. Chyba dává hlášení R Tape loading error.

1. PROGRAM A PROMĚNNÉ
   VERIFY a jméno
2. BYTES
   VERIFY jméno CODE začátek, délka
   pokud je délka na pásce delší než je uvedeno dostáváme chybu R.
   Jinak kontroluje byte v paměti počítače od zadání adresy.
   VERIFY jméno CODE začátek
   kontroluje jména byte na pásce oproti těm v paměti a začíná od uvedené adresy.
   VERIFY jméno CODE
   kontroluje jména bytes na pásce oproti těm v paměti, začíná na adrese, kde byl uchován první byte.
   VERIFY jméno SCREEN$ je totéž jako
   VERIFY jméno CODE 16384,6912 a bude jistě chybné.
3. POLE
   VERIFY jméno DATA písmeno () nebo
   VERIFY jméno DATA písmeno $ ()
   kontroluje pole na pásce s polem v paměti.

LOAD - nahraje z pásku novou informaci a ruší starou.

1. PROGRAM A PROMĚNNÉ
   LOAD jméno
   ruší starý program a proměnné, nahraje nový program a proměnné z kazety. Pokud byl program uchován jako SAVE jméno LINE, provede se automaticky skok.

Chyba: 4 Out of memory (chyba 4, málo paměti) se objeví, když není místo pro nový program a proměnné. V tomto případě starý program a proměnné nejsou zrušeny.

2. BYTES
   LOAD jméno CODE počátek, délka
   pokud je bytes na pásce více než je specifikováno, hlásí se chyba R. Jinak se provede načtení do paměti a přepíše se vše jak bylo uvedeno dříve.
   LOAD jméno CODE počátek
   nahraje bytes z pásku do počítače , začíná na počáteční adrese a přepisuje vše, co tam bylo dříve.
   LOAD jméno CODE
   nahraje bytes z pásku do počítače podle toho, jak byl specifikován 1. byte při SAVE a přepisuje staré bytes v paměti počítače.
3. POLE
   LOAD jméno DATA písmeno () nebo
   LOAD jméno DATA písmeno $ ()
   ruší jakékoliv pole nazvané písmenem nebo písmenem a $ a vytváří nové pole z pásku. Chyba 4 Out of memory (chyba 4 - málo paměti) se objeví, pokud není místo pro nové pole. Staré pole pak není zrušeno.

MERGE - nahrává novou informaci z pásku bez zrušení stané, pokud nejsou použity stejné čísla pro řádky a stejné znaky pro proměnné.

1. PROGRAM A PROMĚNNÉ
   MERGE jméno
   spojuje program z pásku s programem v počítači, připisuje řádky programu nebo proměnné do starého programu. Error 4 Out of memory se objeví, pokud není dostatek místa pro starý a nový program dohromady.
2. BYTES - není možné
3. POLE - není možné

Cvičení:

1. Vytvořte kazetu, kde je 1 program, který po načtení do paměti tiskne seznam dalších programů na kazetě, vyberte program a počítač ho nahraje.
2. Vezměte šachovou grafiku z kapitoly 14 a typujte NEW - nesmaže se, avšak smaže se při vypnutí počítače. Pokud chcete gragiku uchovat musíte ji uložit na pásek pomocí SAVE a CODE. Nejsnadnější způsob jak uchránit všech 21 uživatelsky definovaných znaků je:
   SAVE "chess" CODE USR "a",21*8 a následuje
   VERIFY "chess" CODE
   To je způsob bytového uchování, který byl užit pro záznam obrazu. Adresa prvního byte je USR "a", adresa prvního z osmi bytes, které určuje vzor uživatelské grafiky a číslo je 21*8 - tedy pro každý z 21 znaků to je 8 byte. K zpětnému nahrání užijte:
   LOAD "chess" CODE
   Avšak pokud nahráváte zpět do Spectra s rozdílnou pamětí nebo když jste posunuli uživatelskou grafiku do jiného místa (je to možné několika způsoby), buďte více opatrní a užijte raději
   LOAD "chess" CODE USR "a"
   USR zajistí, že grafika bude nahrána na správnou adresu.

KAPITOLA 21 - TISKÁRNA ZX

SHRNUTÍ: LPRINT, LLIST, COPY

Poznámka: Žádný z těchto příkazů není standartní BASIC, ačkoliv je užit i jinými počítači.

Pokud máte tiskárnu ZX, budete mít jistě i její uživatelskou knížku. Tato kapitola zahrnuje příkazy BASICU potřebné k její práci. První dva LPRINT a LLIST jsou obdobné PRINT a LIST s výjimkou, že místo televize užívají tiskárnu. (L znamená listovnická událost). Při zavedení BASICU se užívalo elektrických psacích strojů místo televize a print skutečně znamenalo tiskněte. Pokud chcete užít rychlou tiskárnu počítače, znamená příkaz LPRINT tisk na řádkové tiskárně. Zkuste tento program:

```
1Ø LPRINT "tento program"
2Ø LLIST
3Ø LPRINT "tiskne řadu znaků"
4Ø FOR n=32 TO 255
5Ø LPRINT CHRØ n;
6Ø NEXT n
```

Užitím příkazu COPY tiskne kopii televizní obrazovky. Např. tiskněte LIST k získání listingu na obrazovce a typujte:

```
COPY
```

Všimněte si, že COPY nepracuje s automatickým listingem, protože je při provedení příkazu vymazán. Musíte buď prvně užít LIST, nebo užít LLIST a zapomenout na COPY. Tiskárnu můžete vždy zastavit tlačítkem BREAK (CAPS SHIFT a SPACE). Pokud provádíte příkazy bez zapnuté tiskárny, měl by se ztratit celý výstup a pak se pokračuje na dalším příkaze.

```
1Ø FOR n=31 TO Ø STEP -1
2Ø PRINT AT 31-n,n;CHRØ(CODE"Ø"+n);
3Ø NEXT n
```

uvidíte vzor znaků, tisknoucích se z horního pravého rohu diagonálně až do konce obrazovky, kde se počítač zeptá scroll? (nepoužívá ne)
Nyní změňte PRINT v řádku 2Ø na LPRINT. Teď tam bude scroll? - což tiskárna a vzor bude pokračovat znaky od F po O. Změňte nyní TABm na AT 31-n,n stále u LPRINT. Dostanete symboly v řadě. Příčinou je, že výstup není tištěn přímo, ale shromažďuje se ve vyrovnávací paměti, až počítač vyšle výsledek. Tištění je:

I když je vyrovnávací paměť plná
II když čísla, apostrofy nebo TAB vyžadují nový řádek
III po příkazu LPRINT, který nekončí středníkem nebo čárkou
IV na konci programu, když něco bylo nevypsáno.

III vysvětluje jak program pracuje s TAB. Pe AT je ignorováno číslo řádku a pozice LPRINT je změněna na číslo sloupce (podobně jak u PRINT, ale pro tiskárnu místo obrazovky). Položka v AT nemůže způsobit, aby byl řádek vyslán na tiskárnu.

Cvičení:

1. Proveďte tištěný graf funkce SIN podle programu v kapitole 17 a pak užijte COPY.

KAPITOLA 22 - DALŠÍ VYBAVENÍ

SHRNUTÍ:

Existují další vybavení, která budou pro ZX Spectrum dostupné. ZX Microdrive je zařízení s velkou rychlostí záznamu a je mnohem pružnější než kazetový přehrávač. Umí operovat jen se SAVE, VERIFY, LOAD, MERGE, ale taky s PRINT, LIST, INPUT, INKEYŽ. Je vytvořena síť spojení, takže můžete používat jeden Microdrive a více počítačů ZX Spectrum. Interface RS232 je standartní spojení, které dovolí spojení Spectra s jinými počítači a tiskárnami než používá Spectrum. Budou se používat další klíčová slova z klávesnice jako OPEN , CLOSE , MOVE, ERASE, CAT a FORMAT. Tyto příkazy zatím nepracují.

KAPITOLA 23 - IN a OUT

SHRNUTÍ: OUT
IN

Procesor umí číst (u paměti RAM) a psát do paměti za pomoci PEEK a POKE. Procesor se nezajímá, zda paměť je ROM, RAM, nebo zda paměť není, ví pouze, že je 65536 adres paměti a může číst z každé paměti (i když je to nesmysl) a psát 1 byte do 1. adresovatelného místi (i když se to může ztratit). Je tam 65536 míst, kterým říkáme místa I/O (jsou to stanoviště pro vstup a výstup). Tyto jsou užity procesorem pro komunikaci s klávesnicí nebo tiskárnou

a mohou být kontrolovány z BASICU užitím funkce IN a příkazu OUT.
IN je funkce podobná PEEK
IN adresa
má jeden argument(adresa)a výsledek je byte čtený z pozice. OUT je příkaz podobný POKE.
OUT adresa, hodnota
píše danou hodnotu na místo určené adresou. Jak je adresa chápána záleží na počítači, docela často bude mnoho různých adres znamenat totéž. U Spectra je významné si všimnout adresy, která je psána binárně, protože jednotlivé bity vedou k tomu, aby pracovaly nezávisle. Existuje 16 bitů, které nazveme A (jako adresa)-A15, A14, A13, A12 ..... A2, A1, AØ
Zde AØ je první bit, A1 je druhý bit, atd. Bity AØ až A4 jsou nejdůležitější. Normálně jsou 1, ale pokud některý z nich je Ø,říká to počítači, aby udělal něco specifického. Počítač nemůže dělat současně víc,jak jednu činnost a tak ne více,jak jeden z těchto pěti bitů může být Ø. Bity A5 a A7 jsou ignorovány, ale pokud se vyznáte, můžete je použít! Nejvhodnější adresy k užití jsou ty, které jsou o 1 menší než násobky 32, žeAØ....A4 jsou 1. Bity A8, A9 jsou někdy užity pro zvláštní informace. Přečtený byte má 8 bitů a ty se často označují jako D7, D6, ...D1, DØ. Zde je řada vstupních adres, které čtou klávesnici a taky EAR zásuvku.
Klávesnice je rozdělena na 8 polovin řad po 5 tlačítkách.

IN 65278	čte polovinu řady od	CAPS SHIFT	do V
IN 65Ø22	-"-	A	" G
IN 64510	-"-	Q	" T
IN 63486	-"-	1	" 5
IN 61438	-"-	Ø	" 6
IN 57342	-"-	P	" 7
IN 49150	-"-	ENTER	" H
IN 32766	-"-	SPACE	" B

(Tyto adresy jsou 254+256x(255-2↑ n), kde n=Ø až 7).
V čteném bitu znamenají bity DØ až D4 pět klíčů (tlačítek). V dané polovině řady - DØ pro vnější klíč, D4 pro nejbližší klíč u středu. Bit je Ø, když je klíč stlačen a 1, když stlačen není. D6 je hodnota zásuvky EAR. Adresa 254 je pro reproduktor (D4) a mikrofonní zásuvku (D3) a taky nastavuje hranici barvy (D2, D1, DØ). Adresa 251 spouští tiskárnu, jak čtení tak psaní, čtení hledá, zda je tiskárna připravena a psaní vysílá tiskové body. Adresy 254, 247, 239 jsou užity pro zvláštní vybavení o které pojednávala kapitola 22. Zkuste program:

```
1Ø FOR n=Ø TO 7: REM poloviční řada - číslo
2Ø LET a=254+256x(255-2↑n)
3Ø PRINT AT Ø,Ø; IN a: GO TO 3Ø
```

a hrajte si. Až budete unuděni,stlačte BREAK a pak typujte NEXTa.

KAPITOLA 24 - PAMĚŤ

SHRNUTÍ: CLEAR

Uvnitř počítače je věc uložena v bytes, tj. číslech mezi Ø a 255. Možná si myslíte, že jste uložili cenu nebo adresu svých zásob, ale to vše bylo změněno na bytes a jen s nimi umí počítač zacházet.Každé místo, kde může být byte uložen,má adresu, což je číslo mezi Ø a FFFFh (adresa je 2 x bytová)-můžete si představit paměť jako řadu číselných schránek s nichž každá může obsahovat 1 byte. Avšak nejsou všechna místa stejná. V 16K RAM počítači chybí místa od 8ØØØ do FFFFh. Adresy od 4ØØØh do 7FFFh jsou místa s přímým přístupem, což znamená, že se dají číst i přepisovat. Od Ø do 3FFFh jsou ROM místa, ty se mohou pouze číst a jejich obsah je dán při výrobě počítače.

ROM	RAM	není užito	
Ø	4ØØØh = 16384	8ØØØh = 32768	FFFFh =65535

K získání obsahu užijeme funkce PEEK, jejímžargumentem je adresa místa a výsledkem je obsah. Tento program tiskne prvních 21 bytes z ROM i s jejich adresami:

```
1Ø PRINT "Address"; TAB 8; "Byte"
2Ø FOR a=Ø TO 2Ø
3Ø PRINT a; TAB 8; PEEK a
4Ø NEXT a
```

Všechny tyto bytes budou pro vás bez významu, ale procesor jim rozumí a ví, co se s nimi dělá. Ke změně obsahu schránky (jen RAM) užíváme příkazu POKE. Má formu:

POKE adresa, nový obsah

kde adresa a nový obsah jsou numerické výrazy. Např. napíšeme:

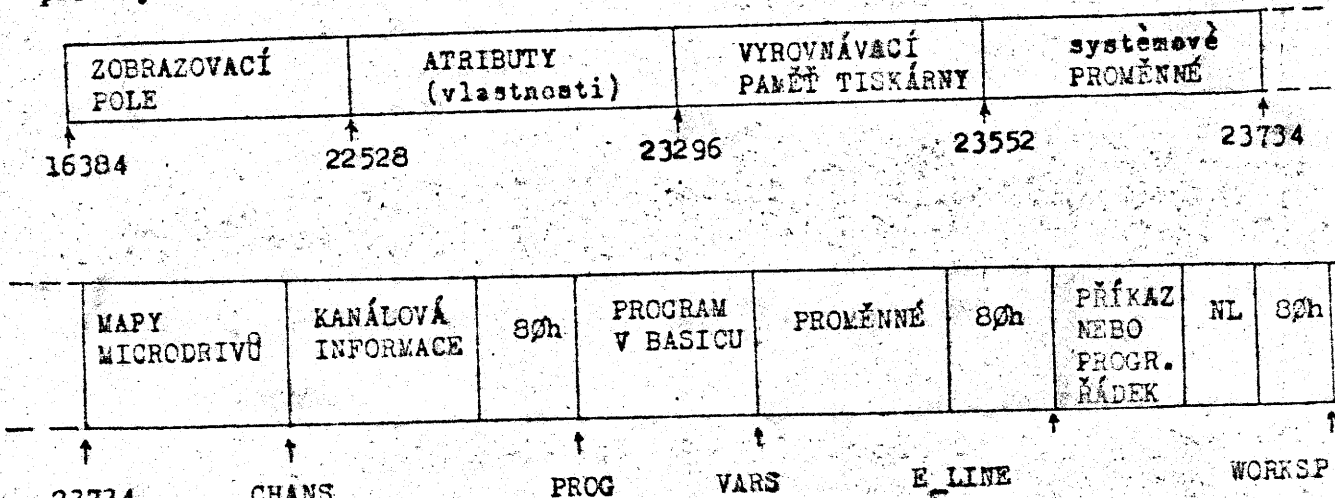
POKE 31ØØØ,57

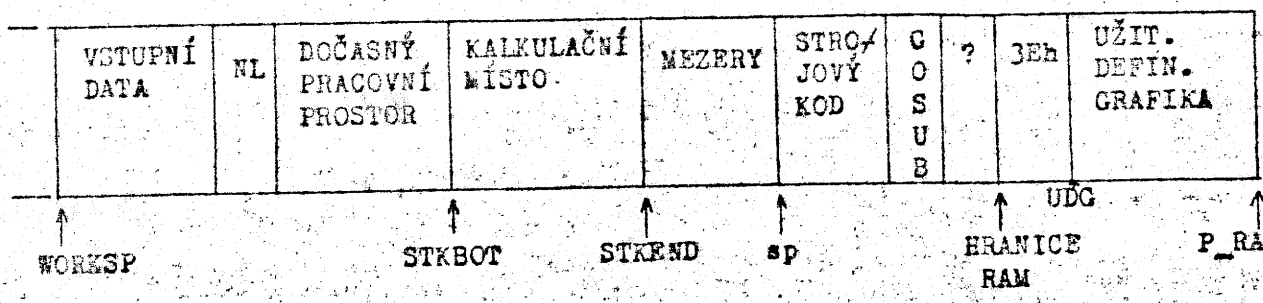
dává na adrese 31ØØØ novou hodnotu 57. Typujte:

PRINT PEEK 31ØØØ

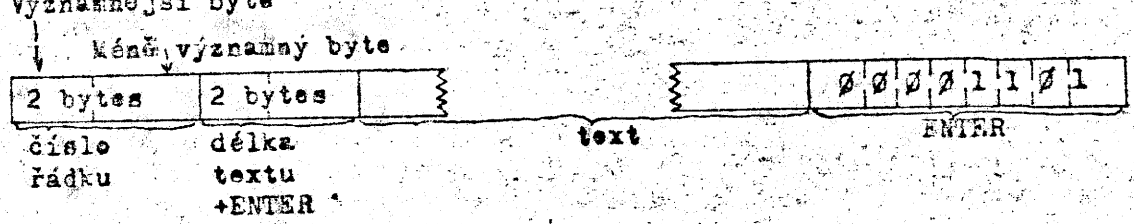
a dostanete 57. (Zkuste POKE pro jiné hodnoty, k ověření, že zde není podvod.) Nová hodnota musí být mezi -255 a 255, při záporném výrazu je připočteno 256. Schopnosti příkazu POKE vám dávají velké možnosti, jak využít počítač, pokud víte, jak na to, ale také velká ničící možnost. Je velmi snadné pomocí POKE vložit na adresu chybnou hodnotu a programy, které jste typovali hodiny, jsou k ničemu. Naštěstí nemůžete způsobit žádnou škodu na počítači. Nyní, pokud vás to zajímá, se podíváme detailněji na použití RAM, pokud vás to nezajímá, neobtěžujte se číst. Paměť je rozdělena na různé oblasti (ukázáno na velkém diagramu), aby uchovala různé invormace. Oblast je dost velká a když vložíte např. číslo řádku programu, prázdný prosto se posune. Totéž se děje při rušení informace, ale opačným směrem. Pole displeje uchovává televizní obraz. Vypadá zvláštně a tak ho asi nebude užití POKE měnit. Každý znak na obrazovce má 8x8 čtvercových bodů a každý bod může být buď Ø nebo 1 (pro papír a inkoust) a užitím binárního vyjádření můžete vzor ukládat jako 8 byte; pro každou řadu 1 byte. Avšak těchto 8 byte není uloženo dohromady. Odpovídající řady 32 znaků jsou uloženy jako jeden pohled, kvůli tom, že elektronový paprsek v televizi běhá zleva doprava. Z toho, že obraz má 24 řádků po 8 pohledech byste očekávali 172 pohledů za sebou, ale chybili byste. První je horní pohled řádky Ø a 7, pak další pohledy a pak je totéž pro řádky 8 až 15 a pak totéž pro řádky 16 až 23. Závěrem toho je, že když použijete PEEK a POKE pro obrazovku, musíte užít SCREEN$ a PRINT AT nebo PLOT a POINT.

Atributy jsou barvy, a tak pro každou tiskovou znakovou pozici je použit formát ATTR. Tím je uchován řádek za řádkem v očekávaném pořadí. Vyrovnávací paměť tiskárny uchovává znaky určené pro tiskárnu. Systémové proměnné obsahují různé informace, říkající v jakém sledu (stadiu) je počítač. Jsou popsány v další kapitole, poznamenejme nyní, že jsou proměnné obsahující hranice mezi různými poli v paměti (nazvané CHANS, PROG, VARS, E_LINE). To nejsou příkazy BASICU a počítač nebude jejich názvy znát.



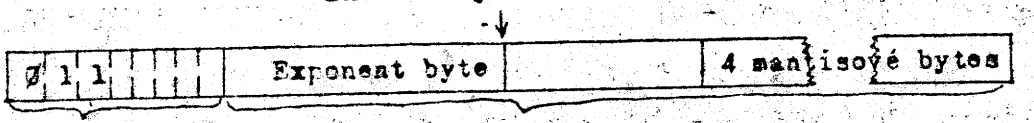


Mapy microdrivů se užívají jen s nimi. Normálně tam nic není. Kanálové informace obsahují informace o vstupních a výstupních zařízeních, klávesnici (s dolní částí obrazovky), horní části obrazovky a tiskárně. Každý řádek programu má v BASICU formu:

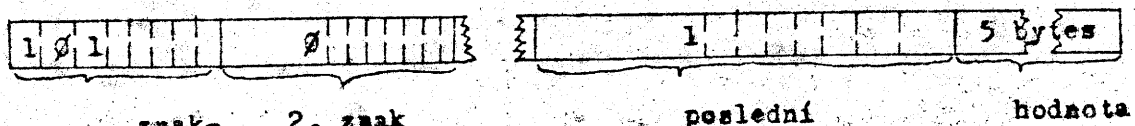


Všimněte si, že na rozdíl od ostatních případů binárních čísel u Z8Ø, je číslo řádku uchováno první s významným bytem, tzn., že je v pořádku, jak jste je napsali. Numerická konstanta v programu je sledována její binární hodnotou, používá znak CHR$ následovaný 5 byte pro samotné číslo. Proměnné mají různé formáty a to podle jejich povah. Znaky ve jméně by se mohly představit takto:

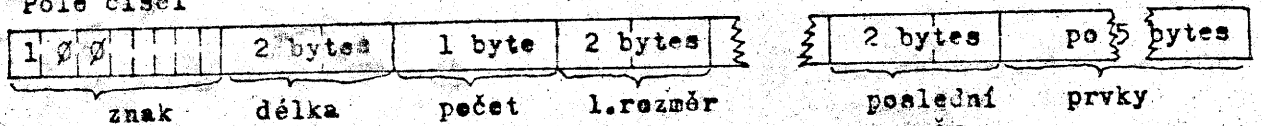
Číslo, jehož jméno je jedno písmeno



Číslo, jehož jméno je delší než 1 písmeno

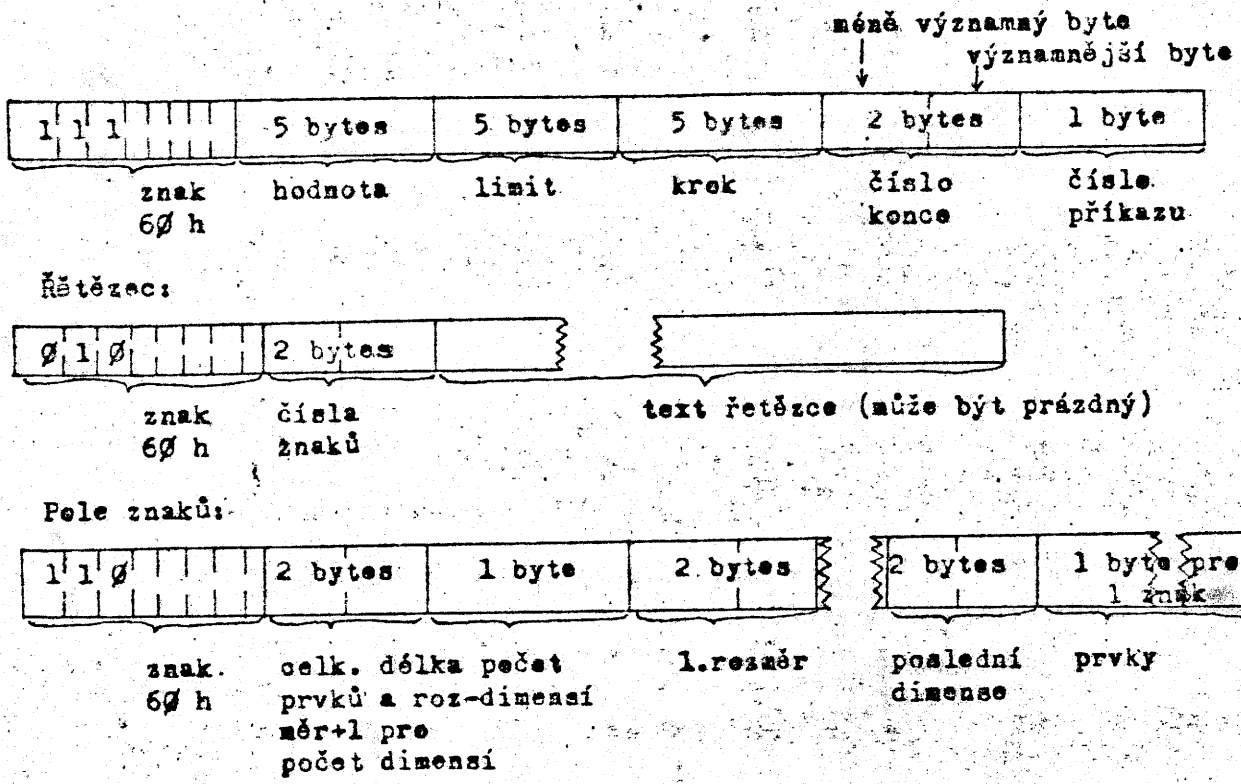


Pole čísel



Pořadí prvků je:
prvně prvky, pro které je první meze 1
dále prvky, pro které je první meze 2
dále prvky, pro které je první meze 3, atd.
pro všechny další možné hodnoty
Prvky se stejnou první mezí jsou sečteny (seřazeny) podle dalších mezí. Jako příklad prvky b, pole 3x6 v kapitole 12 jsou uchovány v pořadí:
b(1,1) b(1,2) b(1,3) b(1,4) b(1,5) b(1,6) b(2,1) b(2,2) ...b(2,6) b(3,1) ...b(3,6)

Řídicí proměnná smyček FOR - NEXT.



Část systému BASIC je kalkulátor, který s aritmetikou a čísly provádí operace. Volná část obsahuje mezery dosud neužité. Strojová oblast je užita procesorem Z 8Ø a uchovává návratové adresy. O oblasti GOSUB byle pojednáme v kapitole 5, byte v RAMTOP má nejvyšší adresu použitou systémem BASIC. Dokonce NEW, který čistí RAM pracuje tak dlouhe, že nemění uživatelem definovanou grafiku. Adresu RAMTOP můžete změnit udáním čísla za příkazem CLEAR.

I - vymaže všechny proměnné
II - vymaže display file (obdobně jako CLS)
III - přesune pozici PLOT do dolního levého rohu obrazovky
IV - vykoná RESTORE
V - vymaže zásobník GOSUB a vloží nový RAMTOP

je důležité, že leží mezi oblastí kalkulátoru a fyzickým koncem RAM. Jinak leží mimo RAMTOP. Užijeme-li CLEAR tímto způsobem, můžeme buď posunout RAMTOP, aby bylo více místa pro BASIC, přepsáním uživatelských grafických symbolů nebo posuneme RAMTOP dolů, aby nebyla horní část paměti mazána příkazem NEW. Natypujte NEW a pak CLEAR 238ØØ k získání představy co se stane s počítačem, když se zastaví. První, čeho si všimnete, když nepřijímá a bzučí. Znamená to, že je blokovaný a musíte ho tedy uvolnit. Uvádíme zde dvě obdobné chybové hlášení s téměř stejným významem. 4 Memory full (plná paměť) a G No room for (není místo pro řádek). Bzučení se rovněž objeví, typujete-li řádek delší než 23 řádků - vaše typování však jak byste si mohli myslet není ignorováno, ale počítač vás upozorňuje, abyste více už netypovali. Délku bzučení můžete ovlivňovat a to vkládáním čísel mezi Ø až 255 na adresu 236Ø8. Obvykle je tam hodnota 64. Jakékoliv číslo (mimo Ø) může být psáno jako

$\pm m \times 2^e$ (kde ± je znaménko, m je mantisa a leží mezi 1/2 a 1 -nesmí být 1, e je exponent a je to celé číslo - může být i záporné).

Předpokládejte, že píšete v binární formě. Protože je to zlomek, bude mít desetinnou tečku a binární zlomek v binární formě vypadá takto:

polovina se píše .1
čtvrt se píše .Ø1
tři čtvrtiny jsou .11
desetina se píše .ØØØ11ØØ11ØØ11ØØ11ØØ11...atd. Protože naše číslo je menší číslo než 1, nebudou tedy před desetinnou tečkou žádné bity a protože je to nejméně 1/2, je bit ihned po binární tečce 1.

K uložení čísla v počítači používáme 5 bytes následovně:

I - napíše se v druhém byte 8 bitů mantisy (víme, že v prvním je 1), dalších 8 bitů do třetího byte, 8 bitů do čtvrtého byte, 8 bitů do pátého byte.

II - nahradí se první bit v druhém byte - který je 1 - znamená Ø pro plus, 1 pro mínus.
III-napíše se exponent+128 v prvém byte. Např. předpokládejme naše číslo 1/1Ø. $1/1Ø=4/5*2^{-3}$
tak mentisa je binárně .11ØØ11ØØ11ØØ11ØØ11ØØ11ØØ11ØØ11ØØ (když je 33. bit 1, musíme zaokrouhlit 32 z Ø na 1) a exponent e=-3. Aplikace našich tří pravidel bude vypadat takto:

Ø111 11Ø1	Ø1ØØ 11ØØ	11ØØ 11ØØ	11ØØ 11ØØ	11ØØ 11Ø1
-3+128	Ø zde značí +	mentisa 4/5 s výjimkou 1. bitu		

Existuje způsob jak uchovat celá čísla mezi -65535 a +65535.
I - první byte je Ø
II - druhý byte je Ø pro kladné číslo, FFh pro záporné
III-třetí a čtvrtý bytex jsou méně a více významné bytes čísla
IV - páté číslo je Ø

KAPITOLA 25 - SYSTÉMOVÉ PROMĚNNÉ

Bytes v paměti od adresy 23552 do 23733 jsou řadou pro specifické užití systému. Pomocí PEEK POUK je s ním možno pracovat. Uvádíme je dále v seznamu s užitím. Jsou nazývány jako systémové proměnné, mají jména, ale neplette si je se jmény jazyka BASIC, počítač je nezná a je to pouze mnemotechnická pomůcka pro lidi. Údaj v prvním sloupci má následující význam:
X proměnná se nesmí zpravovávat POKE, může dojít k havárii systému
N práce s POKE nebude mít žádný účinek
Číslo ve sloupci 1 je počet bytes proměnné. Jsou-li dva bytes, pak první je méně významný byte. Pro změnu hodnoty dvou bytes pomocí POKE můžete použít:

```
POKE n,v-256*INT (v/256)
POKE n+1,INT (v/256)
```

Pek pro hodnotu se může spát

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

POZNÁMKY	ADRESY	JMÉNO	OBSAH
N8	23552	KSTATE	užito pro čtení klávesnice
N1	23560	LAST K	ukládá nově stlačené tlačítko
1	23561	REPDEL	čas (1/50. nebo 1/60. sec). Pro opakování musí být tlačítko drženo. Končí u 35, je možno měnit.
1	23562	REPPER	zpoždění (1/50. nebo 1/60. sec), inicializov.
N2	23563	DEFADD	adresa argumentů uživatelem definované funkce. Byte 1 pokud je funkce, jinak Ø
N1	23565	K DATA	ukládá 2 byte barvy - vstup z klávesnice
N2	23566	TVDATA	ukládá byte barvy, řídí AT a TAB pro televizi
X38	23568	STRMS	kanálové adresy, připojení k řetězci
2	23606	CHARS	o 256 menší než adresa znakového pole (začíná mezerami a pak je symbol copyright). Je obvykle v ROM, možno definovat vlastní v RAM
1	23608	RASP	délka varovného bzučení
1	23609	PIP	délka klepání klávesnice
1	23610	ERR NR	o 1 menší než kod zprávy. Začíná v 255 (pro -1) PEEK 23610 dává 255
X1	23611	FLAGS	různé znaky pro kontrolu systému BASIC
X1	23612	TV FLAG	znak pro televizi
X2	23613	ERR SP	adresa položky v strojovém řetězci, návrat
N2	23615	LIST SP	adresa návratu z automatického listingu
N1	23617	MODE	specifikuje K, L, C, E, G kurzer
2	23618	NEWPPC	řádek, na kterém se má skončit

POZNÁMKY	ADRESY	JMÉNO	OBSAH
1	2362Ø	NSPPC	číslo příkazu na řádku, na který se bude skákat. Pole první NEWPPC a pak NSPPC donutí ke skoku na specifikovaný příkaz v řádku
2	23621	PPC	číslo řádku právě prováděného příkazu
1	23623	SUBPPC	číslo příkazu v řádku, který se provádí
1	23624	BORDCR	barva hranice x8, body obsahující atributy normálně užité pro dolní část obrazovky
2	23625	E PPC	číslo běžného řádku (s programovým kurzorem)
X2	23627	VARS	adresa proměnných
N2	23629	DEST	adresa proměnné v přiřazení
X2	23631	CHANS	adresa kanálových dat
X2	23633	CURCHL	adresa informací užitých pro vstup a výstup
X2	23635	PROG	adresa programu v BASICU
X2	23637	NXTLIN	adresa dalšího řádku v programu
X2	23639	DATADD	adresa ukončení minulé položky DATA
X2	23641	E LINE	adresa typované položky
2	23643	K CUR	adresa kurzoru
X2	23645	CH ADD	adresa dalšího interpretovaného znaku, znak argumentu PEEK nebo NEWLINE na konci POKE
2	23647	X PTR	adresa znaku po otazníku
X2	23649	WORKSP	adresa dočasné pracovní oblasti
X2	23651	STKBOT	adresa počátku kalkulační oblasti
X2	23653	STKEND	adresa počátku volného prostoru
N1	23655	BREG	kalkulační b registr
N2	23656	MEM	adresa oblasti užité pro kalkulační paměti obvykle MEMBOT, ale ne vždy
1	23658	FLAGS2	více příznaků
X1	23659	DF SZ	počet řádků (vč. volných) v dolní části obrazovky
2	2366Ø	S TOP	číslo vrcholu programové řádky v automatickém listingu
2	23662	OLDPPC	číslo řádku, kde skáče CONTINUE
1	23664	OSPCC	číslo příkazu na řádku, kde skáče CONTINUE
N1	23665	FLAGX	různé příznaky
N2	23666	STRLEN	délka řetězce pro užití přiřazení
N2	23668	T ADDR	adresa další položky v symbolické tabulce
2	2367Ø	SEED	užito pro RND. proměnná v příkazu RANDOMIZE
3	23672	FRAMES	3 byte (1. je méně významný), čítač řádu přírustek po 20ms. Viz kapitola 18
2	23675	UDG	adresa 1. uživatelské grafiky, můžete změnit pro úsporu místa zúžením tohoto pole
1	23677	COORDS	x-souřadnice posledního kresleného bodu
1	23678		y-souřadnice posledního kresleného bodu
1	23679	P PONS	33-počet tištěných pozic
1	2368Ø	PR CC	méně významný byte adresy další pozice pro LPRINT (ve vyrovnávací paměti pro tisk)
1	23681		není užito
2	23682	ECHO E	33-počet sloupců a 24-počet řádků (dole) vstup. buffer
2	23684	DF CC	adresa zobrazovacího pole - pozice PRINT
2	23686	DFCCL	jako DF CC pro dolní část obrazovky
X1	23688	S POSN	33-číslo sloupce pro PRINT
X1	23689		24-číslo řádku pro pozici PRINT
X2	2369Ø	SPOSNL	jako 3 POSN pro dolní část
1	23692	SCR CT	počítá scroll, je vždy o 1 víc než počet, který bude proveden před zastavením scroll? Pokud budete provádět POKE s číslem větším jak 1 (např. 255), obraz se bude scrollovat bez dotazu
1	23693	ATTR P	stálé běžné barvy (jako řada barevných příkazů)
1	23694	MASK P	užito pro transparentní barvy. Každý bit, který je 1, ukazuje, že odpovídající atributový bit zaměňte z ATTR P, ale z toho co je na obrazovce

POZNÁMKY	ADRESY	JMÉNO	OBSAH
N1	23695	ATTR T	dočasné běžné barvy (sada barevných položek)
N1	23696	MASK T	jako MASK P, ale dočasně
1	23697	P FLAG	více významů
N30	23698	MEMBOT	kalkulační paměťová oblast užita k uchování čísla, které nemohou být uloženy v kalkulační oblasti
2	23728		není užito
2	23730	RAMTOP	adresa posledního bytes oblasti BASICU
2	23732	P-RAMT	adresa posledního bytesfyzické RAM

Tento program vám ukazuje prvních 22 bytes z proměnné oblasti:

```
10 FOR n=0 TO 21
20 PRINT PEEK (PEEK 23627+256*PEEK 23628+n)
30 NEXT n
```

Zkuste si nalézt kontrolní proměnnou z předešlé tabulky. Nyní změňte řádek 20 na:

```
20 PRINT PEEK (23755+n)
```

Zobrazí se prvních 22 bytes programové oblasti. Najděte tam program.

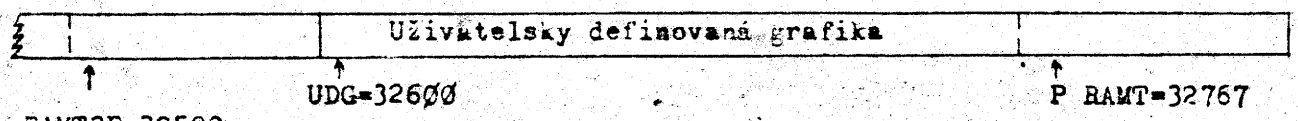
KAPITOLA 26 - POUŽITÍ STROJOVÉHO KÓDU

SHRNUTÍ: USR s numerickým argumentem

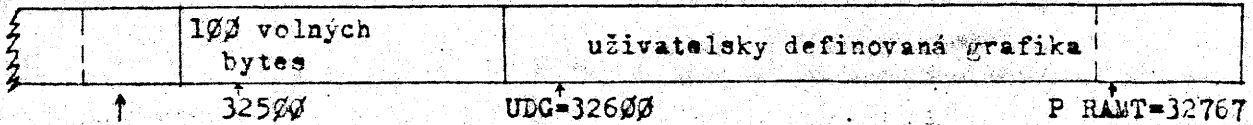
Tato kapitola je napsána pro ty, kteří rozumí strojovému kodu Z 80, souboru instrukcí, kterému rozumí procesorový čip Z80. Pokud máte zájem, můžete o tom sehnat sou stu knih. Bude-li tam zmínka i o ZX SPECTRU, je to výborné. Tyto programy jsou psány v assembleru, který s trochou praxe není složité pochopit (podívejte se na seznam instrukcí v příloze A). Ke zpuštění těchto programů potřebujete instrukce vkládat sekvencí bytes v tzv. strojovém kodu. Tento překlad provádí obvykle počítač za použití assembleru. Ve SPECTRU není žádný assemblor, ale je možné zakoupit ho nahraný na kazetě. Pokud program není příliš dlouhý, můžete překlad provést sami. Zkuste tento příklad:

```
ld bc, 99
ret
```

který uloží do registru bc 99. Vše se přeloží na 4 strojové kody 1, 99, 0 (pro ld bc,99) a 201 (pro ret). Podívejte se do přílohy A. Pokud máte program napsaný ve strojovém kodu je dalším úkolem dostat ho do počítače (assembler to dělá automaticky). Musíte se rozhodnout, kam ho v paměti uložit a za tím účelem si pro něj rezervovat v paměti dostatek místa. Předpokládejte, že používáme 16K SPECTRUM. Horní kod paměti vypadá následovně:



Pokud typujete CLEAR 32499, zajistíte si prostor 100 bytes (počátek je na adrese 32500)



K uložení programu ve strojovém kodu užijete následující program:

```
10 LET a=32500
20 READ n: POKE a,n
30 LET a=a+1: GO TO 20
40 DATA 1,99,0,201
```

(program se zastaví se zprávou E Out of DATA, když se zaplní 4 bytes, které jste specifikovali).Ke spuštění programu ve strojovém kódu užijeme funkce USR, ale nyní s numerickým argumentem(počáteční adresou).Výsledek programu je hodnota bc.

```
PRINT USR 32500
```

dostanete odpověď99.Návrat ze strojového kódu do BASICu provádí instrukce Z80-ret. Registry iy a i byste neměli pro své programy užívat. Náš program snadno uchováme:

```
SAVE "jméno" CODE 32500,4
```

při následovném LOAD je tento program automaticky rozběhnut. Toho lze užít pro rozběhnutí programů v BASICu ihned po jeho natažení z magnetofonu do počítače a to automaticky.

```
10 LOAD ""CODE 32500,4
20 PRINT USR 32500
```

Proveďte nejdříve:

```
SAVE "nějaké jméno" LINE
```

a pak

```
SAVE "xxxx" CODE 32500,4
LOAD "nějaké jméno"
```

bude automaticky natahovat a rozbíhat program v BASICu a BASIC program bude natahovat a rozbíhat strojový kód.

PŘÍLOHA A
Soustava znaků

Na str.135 originálního manuálu je uvedea přehled všech znaků a jejich hexadecimální kódy.

kód	znak	hex	Z80 assembler	spolu s CB	spolu s ED

Pokračování viz příloha A.

PŘÍLOHA B
Zprávy
Objeví se v dolní části obrazovky, když počítač zablokuje běh BASICU, vysvětlují proč se běh zastavil, zda přirozeně nebo byla chyba. Zpráva obsahuje číslo nebo písmeno a pak stručné vysvětlení,co se stalo a na kterém řádku to bylo. Sděluje se i číslo příkazu na řádce. To, jak se chová CONTINUE,záleží na zprávách. Normálně se jde na řádek a příkaz specifikovaný v poslední zprávě, ale existují i výjimky se zprávami 0,9 a D (viz příloha C). Zde je tabulka obsahující všechny zprávy. Dovíme se i za jakých okolností se může zpráva objevit s odvoláním se na přílohu C. Např. chyba A Invalid argument (neplatný argument) se může objevit s SQR, IN, ACS, ASN a odkáže Vás na přílohu C, kde se dovíte jaký je platný argument.

Kód	Význam	Situace
0	OK úspěšný konec programu nebo skok na číslo řádku. Zpráva nemění řádek a příkaz pro CONTINUE.	žádná
1	NEX bez FOR neexistuje kontrolní proměnná (nebyla udána v příkazu FOR), ale je proměnná s tímtéž jménem.	NEXT
2	Proměnná nenalezena pro jednoduchou proměnnou se to stane, když je proměnná vyvolána dříve, než jí byla příkazem LET přiřazena nějaká určitá hodnota nebo čtena v READ nebo INPUT nebo natažena z magnetofonu nebo definována v příkazu FOR.	žádná

U dimenzí se to stane, když je proměnná užita dřív než dostala rozměr v příkazu DIM nebo byla natažena z magnetofonu.

indexová proměnná řetězce

3 Chybný index
index pro pole leží mimo pole nebo je napsán jako chybné číslo. Chyba se objeví při záporném indexu nebo když je větší než 65535, pak bude chyba typu B.

4 Nedostatek paměti
není dostatek místa v počítači pro to, co se bude provádět. Pokud je skutečně málo místa můžete zkusit užít DELETE a zrušit řádek programu nebo i dva (pak je znovu napíšete), aby bylo místo pro manipulaci s příkazem CLEAR.

LET, INPUT, FOR, DIM, GO SUB, LOAD, MERGE občas během provádění výrazů

5 Mimo obrazovku
vstupní příkaz se pokusil zapsat víc jak 23 řádků v dolní polovině obrazovky. Také se objeví při PRINT AT 22, ...

INPUT, PRINT AT

6 Číslo je příliš velké
výpočet vedl k číslu většímu než 10^{38}

aritmetika

7 RETURN mimo GO SUB
existuje více příkazů RETURN než GO SUB

RETURN

8 Konec pole

microdrive, operace

9 Příkaz STOP
následné CONTINUE nebude STOP opakovat, ale bude pokračovat na dalším řádku.

STOP

A Neplatný argument
nevhodný argument funkce

SQR, LN, ASN, ACS, USR (s řetězcovým argumentem)

B Celé číslo mimo rozsah
když se požaduje celé číslo, floating argument je zaokrouhlen na nejbližší celé číslo Pokud je číslo mimovhodnou mez, objeví se chyba B.

RUN, RANDOMIZE, POKE, DIM, GO TO, GO SUB, LIST, LLIST, PAUSE, PLOT, CHR$, PEEK, USR (numeric. argumentem)

C Nesmysl v BASICU
Text v řetězci nemá formu platného výrazu.

VAL, VAL$

D BREAK - CONT opakuje
během nějaké rozsáhlé operace byl stlačen BREAK. Chování CONTINUE je pak normální v tom, že opakuje tento příkaz. Srovnej se zprávou L.

LOAD, SAVE, VERIFY, MERGE, LPRINT, LLIST, COPY taky zeptá-liše počítač scroll? a vy typujete N, SPACE nebo STOP

E Mimo DATA
čte za koncem seznamu DATA

READ

F Chybné jména
SAVE má prázdné uvozovky nebo jméno je delší jak 10 znaků.

SAVE

G Není místo pro řádek
v paměti není místo k přijetí dalšího řádku

vstoupení řádku do programu
INPUT

H STOP v INPUT
Některá vstupní data začala se STOP, nebo byl stlačen INPUR LINE. Podobně jako při zprávě 9, po zprávě H bude CONTINUE pracovat normálně, opakováním příkazu INPUT

I FOR bez NEX
existuje FOR cyklus bez NEXT

FOR

J Chybné zařízení I/O

microdrive, operace

K Chybná barva
cpecifikované číslo není vhodné

INK, PAPER, BORDER, FLASH, BRIGHT, INVERSE, OVER také po odpovídajících kontrolních znacích

L	Zastavení programu byl stlačen BREAK, to je vždy kontrolováno mezi dvěma příkazy. Řádek a příkaz ve zprávě odpovídají před BREAK, ale CONTINUE jde za (umožňuje dělat skoky) a tak se příkazy neopakují.	žádné
M	RAMTOP nevhodné číslo specifikované pro RAMTOP je buď příliš velké, nebo malé.	CLEAR, možno v RUN
N	Ztracený příkaz skok na příkaz, který neexistuje	RETURN, NEXT, CONTINUE
O	Chybný tok dat	microdrive, operace
P	FN bez DEF uživatelem definovaná funkce.	FN
Q	Chyba v parametru špatné číslo argumentu nebo jeden z nich je špatně typován (řetězec místo čísla a opač.)	FN
R	Chyba v nahrávání data na pásce nalezena, ale z různých důvodů nemohla být načtena nebo ověřena.	VERIFY, LOAD, MERGE

PŘÍLOHA C
Popis ZX SPECTRA pro praktiky

První část této přílohy opakuje některé části úvodní knihy, týkající se klávesnice a obrazovky.
Klávesnice:
ZX Spectrum neobsahuje pouze prosté symboly (písmena a čísla), ale taky složené symboly (klíčová slova, funkční jména). Tyto se typují přímo z klávesnice. K získání toho všeho je tlačítkům určeno 5 nebo i více významů, čehož se dosahuje pomocí tlačítek shift a různými mody počítače. Mod je ukazován kurzorem, což je znak ukazující, kam bude vložen další znak na obrazovce.
Mod K (pro klíčová slova) je nastaven, když program očekává příkaz nebo řádekprogramu (dříve než vstup dat)a ze své pozice na řádku ví, že by měl očekávat číslo řádku nebo klíčové slovo. Tento stav je na počátku řádku nebo po THEN nebo dvojtečce(s výjimkou řetězce). Pokud není stlačen SHIFT bude další tlačítko interpretováno buď jako klíčové slovo (psané na tlačítku) nebo jako číslo.
Mod L (pro písmena) se normálně objeví ve všech ostatních případech. Pokud není SHIFT bude tlačítko interpretováno jako hlavní symbol v horní části tlačítka.
V K i L modu jsou tlačítka s symbol SHIFT interpretovány červeným znakem na tlačítku a CAPS SHIFT s číslicí jako kontrolní funkce napsaná bíle nad tlačítkem. CAPS SHIFT s jiným tlačítkem nemá význam klíčového slova v K modu. V L modu mění malá písmena na velká.
Mod C (pro velká písmena) je varianta L modu, ve kterém se mění malá písmena na velká. CAPS LOCK způsobuje změnu z L modu na C mod a opačně.
Mod E (rozšířený) je užit pro získání dalších znaků, zvláště příkazů a objeví se jsou-li stlačeny oba shifty současně. V tomto modu dostáváme funkce psané zeleně nad tlačítkem nebo pro horní řadu posloupnost barev. Je-li stlačen v E modu SYMBOL SHIFT dostáváme znaky psané červeně pod tlačítky.
Mod G (pro grafiku) se objeví po GRAPHICS (CAPS SHIFT a 9). Číslicová tlačítka dávají grafickou mozaiku a každé písmeno kromě V, W, X, Y a Z dávají uživatelskou grafiku.
Pokud je tlačítko drženo déle než 2 až 3 sekundy (dá se libovolně nastavit), bude se tisk znaku opakovat - autorepeat. Vstup z klávesnice se na obrazovce objeví v její dolní části a každý znak nebo symbol či funkce je vkládána před kurzor. Kurzorem můžeme posunovat vlevo pomocí CAPS SHIFT a 5, vpravo pomocí CAPS SHIFT a 8. Znak před kurzorem lze zrušit pomocí DELETE (CAPS SHIFT a 0). Celý řádek můžeme zrušit natypováním EDIT (CAPS SHIFT a 1) s následným ENTER. Pokud stlačíme ENTER, provede se řádek a neobsahuje-li syntaktickou chybu je zařazen do programu nebo je užit jako vstup dat.

Je-li v řádku chyba rozsvítí se vedle ní ? a řádek se ponechá v dolní části obrazovky. Výpis programu je zobrazován v horní části obrazovky. Způsoby jakými ho lze uskutečnit jsou komplikovanější a jsou blíže vysvětleny v kapitole 2. Poslední vložený řádek se nazývá běžný a je označen < . Můžeme ho však zrušit tlačítkem ⇩ (CAPS SHIFT a 6) a ⇧ (CAPS SHIFT a 7) Pokud je stlačeno EDIT, je běžný řádek shozen dolů a může být opravován. Když se provádí příkaz nebo běží-li program, výstup je zobrazen v horní části obrazovky. V dolní části obrazovky se objevují zprávy dávající kod (číslo nebo písmeno) odkazující na přílohu B se stručným popisem chyby, kde k ní došlo (číslo řádku a příkazu na něm). Zpráva na obrazovce zůstává dokud není stlačeno tlačítko a není indikován K mod. V některých případech se CAPS SHIFT a SPACE shodné jako BREAK, zastavuje počítač se zprávou D nebo L. To je
(I) na konci příkazu, pokud běží program
(II) používáli počítač magnetofon nebo tiskárnu.

Televizní obrazovka:
Má 24 řádků, každý po 32 znacích, rozdělených do dvou částí. Horních 22 řádků zobrazuje výpis nebo výstup programu. Pokud tisk v horní části dosáhne spodního konce, vše se o 21. řádků posouvá. Počítač se však napřed zeptá "scroll ?" Stlačíte-li N,SPACE nebo STOP, bude program zastaven se zprávou "D BREAK - CONT repeats", stlačení jiného tlačítka vyvolá posunutí řádků nahoru. Dolní část obrazovky je užita pro vstupní příkazy, vstup dat, psaní programu a sdělování zpráv.
Každý znak má atributy specifikující jeho papír/pozadí/ a inkoust/písmo/, dvě úrovně jasu a blýskání. Přípustné barvy jsou: černá, modrá, červená, fialová, zelená, světle modrá, žlutá a bílá. Okraj obrazovky může mít stejné barvy s použitím příkazu BORDER.
Atributy ve znakové pozici určují, zda je znak napsán na TV obrazovce nebo vytištěn na tiskárně. Přesný způsob je určen tiskovými parametry. Jsou dva druhy: stabilní a dočasné. Parametrů je šest a to: PAPER, INK, FLASH, BRIGHT, INVERSE, OVER. Stabilní parametry pro horní část obrazovky jsou stanoveny pomocí PAPER a INK /při prvním zapnutí je černá barva na bílém papíře bez blýskání a přepisování/. Stabilní parametry pro dolní část obrazovky užívají barvu BORDER jako barvu papíru s černobílým kontrastním inkoustem, normálním jasem bez blýskání a přepisování.
Dočasné parametry jsou dány příkazy PAPER, INK, FLASH, BRIGHT, INVERSE, OVER položkami, které jsou v PRINT, LPRINT, INPUT, DRAW a CIRCLE. Dočasné parametry trvají pouze do konce příkazu PRINT nebo INPUT atd. Pak jsou nahrazeny parametry stabilními. PAPER a INK jsou dány čísly Ø až 9. Parametr Ø až 7 jsou barvy :

Ø černá	4 zelená
1 modrá	5 světle modrá
2 červená	6 žlutá
3 fialová	7 bílá

Parametr 8/průhledný/ znamená, že barva na obrazovce je při tisku nezměněna. Parametr 9/kontrast/ dá kontrastní/opačnou/ barvu k barvě dané. FLASH a BRIGHT mohou být Ø,1,8. 1 znamená, že jsou funkční, Ø, že ne, 8 - barva se nemění. Parametry OVER a INVERSE mohou být Ø nebo 1.
OVER Ø staré znaky jsou nahrazeny novými
OVER 1 libovolné znaky staré i nové jsou spojeny užitím logických operací nebo jsou přepsány
INVERSE Ø znaky jsou tištěny normálně
INVERSE 1 znaky jsou tištěny s barvou INK, která odpovídá barvě PAPER na pozadí, jež má barvu INK.
Tiskárna:
Výstup na tiskárnu ZX je z důvodu rychlejšího tisku vybavena přes vyrovnávací paměť, takže se tiskne najednou celý řádek tj. 32 znaků,
(I) když tisk přechází z jedné řádky na druhou
(II) když se tiskne ENTER
(III) na konci programu, pokud je něco nevytištěno
(IV) když proměnná TAB nebo čárky posunou tiskovou pozici na nový řádek.
TAB a čárka řídí mezery stejně jako při použití televize.

AT mění pozici tisku a používá číslo sloupce, číslo řádku ignoruje. Tiskárna nereaguje na INVERSE a OVER. Rovněž PAPER, INK, FLASH, BRIGHT jsou bez účinku. Tiskárna se zastaví s hlášením B, pokud je stlačen BREAK. Pokud tiskárna není zapojena, výstup se ztrácí.

BASIC
Čísla jsou uchována s přesností 9 nebo 10 míst. Největší číslo smí být 10^{38}, nejmenší kladné číslo $4*10^{-39}$. Číslo se v ZX Spectru uchovává v pohyblivé řadové čárce s exponentním bytu ($1 <= e <= 255$) a 4 byte mantisy m ($1/2 <= m < 1$). To reprezentuje číslo $m*2^{e-128}$. Protože $1/2 <= m < 1$ významnější bit mantisy m je vždy 1, proto ho ve skutečnosti můžeme nahradit Ø pro kladná čísla a -1 pro čísla záporná. Malá celá čísla mají specifické zobrazení, ve kterém je první byte Ø, druhý je jako znaménko (Ø nebo FFhexa) a třetí a čtvrtý byte jsou čísla, kde první je méně významným. Numerické proměnné mají jména různé délky, začínají písmenem a mohou následovat písmena i čísla dohromady. Mezery a barevné proměnné jsou ignorovány a všechny znaky jsou přeměněny na malá písmena. Kontrolní proměnná cyklu FOR-NEXT má jméno dlouhé pouze 1 znak. Numerické pole má jméno dlouhé pouze 1 znak a může být totožný se jménem jednoduché proměnné. Pole může mít libovolný počet dimenzí, index začíná jedničkou. Řetězce jsou libovolně dlouhé, jméno řetězce se skládá z jednoho znaku doplněného $. Řetězcové pole může mít libovolně mnoho dimenzí. Jméno je písmeno, následováno $ a nesmí být stejné jako jméno řetězce. Všechny řetězce v poli mají tutéž pevou délku, která je specifikována jako zvláštní rozměr v příkaze DIM. Nejmenší rozměr je 1.
Řetězce se dají určit pomocí plátkování. Plátek může být:
(I) prázdný
(II) numerický výraz
(III) numerický výraz TOnumerický výraz a je užit k vyjádření části řetězce
(a) řetězcová proměnná (slicer)
(b) řetězcové proměnné pole
V(a) předpokládejme, že řetězec má hodnotu s$. Pokud je plátek prázdný, je výsledek s$, který se týká podčásti řetězce, taky prázdný. Pokud je slicer numerický výraz, pak výsledkem je n-tý znak s $ (podřetězec délky 1). Pokud slicer má formu (III) pak předpokládejme, že první numerický výraz má hodnotu m (bez udání = 1) a druhé n (bez udání je to délka s$).
Pokud $1 <= m <= n <=$ délka s$, pak výsledkem je část řetězce, která začíná m-tým znakem a končí n-tým.
Pokud $0 <= n < m$, pak výsledkem je prázdný řetězec. V jiných případech je výsledkem chybové hlášení 3.
Plátkování se provádí před provedením funkcí nebo operací, pokud závorky neurčí jinak. Části řetězců se mohou přiřazovat (podívejte se na LET). Pokud je řetězcová závorka napsána v řetězcové konstantě, musí být zdvojena.

Funkce:
Argument funkce nepotřebuje závorky, pokud to je konstanta nebo (indexovaná či částečná proměnná (pro TO));

FUNKCE	TYP ARGUMENTU	VÝSLEDEK
	(x)	
ABS	číslo	absolutní hodnota
ACS	číslo	arccosinus v radiánech, pokud je mimo -1 až +1, pak se objeví chyba A
AND	binární operace pravý operand je vždy číslo	
	numerický levý operand	A AND B = {A if B <> Ø; Ø if B=Ø}
	řetězcový levý operand	A$ AND B= { A$ if B<>Ø; "" if B=Ø }
ASN	číslo	arcussinus v radiánech, pokud je mimo -1 až +1, pak se objeví chyba A
ATN	číslo	arcustangens v radiánech
ATTR	dva argumenty x a y, obě čísla uzavřena v ( )	číslo, jehož binární forma dává kód řádku x a sloupce y na TV obrazovce.

BIN		Bit (nejvýznamnější) -1 pro blýskání, Ø neblýská se. Bit 6 je pro a, Ø pro normální jas. Bity 5 až 3 jsou barva papíru. Bity 2 - Ø jsou barvy inkoustu. Chyba B je když Ø <=x <=23 a Ø <=y <=31. To není skutečná funkce, ale možná forma čísel: BIN následuje sekvencí Ø a 1 s binární reprezentací.
CHR$	číslo	znak, jehož kod je x, zaokrouhlen na nejbližší celé číslo.
CODE	řetězec	kod prvéhoznaku v x (pro Ø je x prázdný řetězec).
COS	číslo (vradiánech)	cosinus x
EXP	číslo	e^x
FN		FN následuje písmeno, které volá uživatelsky definovanou funkci (podívejte se na DEF). Argumenty musí být uzavřeny v závorkách. Závorky musí být stále přítomny i když není žádný argument.
IN	číslo	Výsledek vstupu do procesoru musí být: x (Ø <=x <=FFFFhexa) (natahuje pár registrů x a odpovídá assemblerovské instrukci in a (c)).
INKEY$	žádný	čte klávesnici. Výsledek je znak v L nebo C modu reprezentující tlačítko, pokud je stlačeno.
INT	číslo	celá část (vždy zaokrouhlena dolů)
LEN	řetězec	délka
LN	číslo	logaritmus přirozený. Chyba A, je-li x <=Ø
OR	binární operace	Ø když a <> Ø, 1 když x=Ø; a OR b = { 1 pokud b <> Ø, a pokud b=Ø } má prioritu 4
PEEK	číslo	má prioritu 2 hodnota 1 byte v paměti na adrese x (zaokrouhleno na nejbližší celé číslo). Není-li x v rozsahu Ø až 65535, pak je hlášena chyba B.
PI	žádný	3,14159265...
POINT	argumenty x ,y obě čísla v závorkách	1 pokud obrazový bod at(x,y) má barvu INK, Ø když jde o papír. Není-li x v rozsahu od Ø do 255 a y od Ø do 175 - chyba B
RND	žádný	další pseudo-náhodné číslo v posloupnosti, kde se umocňuje 75 modulo 65537, odečítá se 1 a další 65536. Ø <=y <1.
SCREEN$	dva argumenty x, y, obě čísla uzavřena v ( )	objeví se znak (v TV), buď normální, nebo obrácený v řádce x a sloupci y. Výsledkem je prázdný řetězec, pokud znak chybí.
SGN	číslo	znaménko (-1 pro záporné číslo, Ø pro Ø, 1 pro kladné číslo) čísla x
SIN	číslo v radiánech	sinus x
SQR	číslo	odmocnina. Chyba A, když x <Ø
STR$	číslo	řetězec znaků, který by byl zobrazen při tisku x
TAN	číslo v radiánech	tangens
USR	řetězec	adresa bitového vzoru pro uživatelskou grafiku odpovídající x. Chyba A, když x není jednoduchý znak mezi a až u.
USR	číslo	používá strojového kodu podprogramu, jehož startovací adresa je x. Při návratu je výsledkem obsah bc registru.
VAL	řetězec	vyhodnocuje x (bez uvozovek) jako numerický výraz. Chyba C, když x obsahuje syntaktickou chybu.

VAL$	řetězec	Jsou možné další chyby, závisí na vyjádření vyhodnocuje x (bez uvozovek) jako řetězcové vyjádření. Dochází k chybě C, když x obsahuje syntaktickou chybu nebo dává numerickou hodnotu. Jsou možné i další chyby, např. VAL
-	číslo	negace

Následující jsou binární operace:

+	sčítání (čísel), slučování (řetězců)	
-	odčítání	
*	násobení	
/	dělení	
↑	umocňování. Dochází k chybě B, pokud je záporný levý operand.	
=	rovnost	oba operandy musí být téhož typu. Výsledkem je 1, když je výraz pravdivý, Ø když není.
>	větší než	
<	menší než	
<=	menší nebo rovno	
>=	větší nebo rovno	
<>	není rovno	

Funkce a operace mají následující priority:

Operace	Priorita
Indexování a plátkování	12
Všechny funkce mimo NOT a mínus (prosté)	11
umocňování	1Ø
Prosté mínus (mínus k negaci něčeho)	9
* a /	8
+, - (mínus je užito k odčítání 1 čísla od druhého)	6
=, <, >, <=, >=, <>	5
NOT	4
AND	3
OR	2

Příkazy:
V tomto seznamu platí:

c	reprezentuje prostý znak
v	reprezentuje proměnnou
x, y, z	reprezentuje numerický výraz
m, n	reprezentuje numerický výraz, který je zaokrouhlen na nejbližší celé číslo
e	reprezentuje výraz
f	reprezentuje řetězcově vyhodnocenou hodnotu
s	reprezentuje posloupnost příkazů, oddělených dvojtečkami
c	reprezentuje posloupnost barevných položek, každá je ukončena čárkou nebo středníkem. Barevné položky mohou mít formu: PAPER, INK, FLASH, BRIGHT, INVERSE, OVER

Všimněte si že výrazy jsou dovoleny všude (s výjimkou čísla řádku na počátku řádku).
Všechny příkazy mimo INPUT, DEF a DATA se mohou používat jako buď samostatné povely nebo přímo v programu. V programu jsou však mnohem citlivější. Povel nebo řádek příkazu může mít více příkazů, oddělených dvojtečkami. Neexistuje omezení, kde se nemůže dát příkaz s výjimkou IF a REM.

BEEP x,y	zazní tón z reproduktoru pro x vteřin ve výšce y půltónů nad C (nebo pod C je-li y záporné).
BORDER m	barevné lemování obrazovky a barvy papíru v dolní části obrazovky dle m. Objeví se K, jestliže m není v rozsahu Ø až 7.
BRIGHT	zvětšení jasu. Pro n=Ø jas normální, n=1 jas zvýšený, n=8 průhledný. Chyba K, jestliže n není Ø, 1nebo 8.

CAT	pracuje s microdrives
CIRCLE x, y, z	kreslí kružnici se středem (x, y), poloměru z
CLEAR	ruší všechny proměnné a uvolňuje prostor, který zabíraly. Provádí RESTORE a CLS, předává pozici PLOT do levého dolního rohu, uvolňuje GO SUB
CLEAR n	podobné CLEAR, ale pokud je to možné, posouvá systémovou proměnnou RAMTOP na adresu n a provádí tam GO SUB
CLOSE #	nepracuje bez microdrive
CLS	maže obrazovku, např. zobrazené pole
CONTINUE	pokračuje v programu a začíná tam, kde se zastavil se zprávou jinou než Ø. Pokud byla zpráva 9 nebo L, pokračuje na dalším příkaze, jinak opakuje příkaz. Pokud byla poslední zpráva v povelovém řádku, pak CONTINUE bude na tomto řádku pokračovat a půjde do smyčky. Pokud byla chyba Ø:1 dá zprávu Ø, pokud byla Ø:2 nebo dává chybu N, pokud bylo Ø:3 nebo více objeví se CONTINUE na obrazovce.
COPY	vysílá kopii horních 22 řádek TV na tiskárnu, pokud je připojena, jinak se neděje nic. Všimněte si, že COPY může být užito k tisku automatických listingů, pokud se objeví na obrazovce. Pokud je stlačen BREAK objeví se zpráva D.
DATA $e_1, e_2, \ldots$	čte ze seznamu DATA. Musí být v programu.
DEF FN $\alpha(\alpha_1, \ldots, \alpha_k)=e$	uživatelsky definované funkce, musí být v programu. Jak α tak $\alpha_1 \ldots \alpha_k$ je buď písmeno nebo písmeno následované $ pro řetězcové proměnné.
DELETE f	nepracuje bez microdrive
DIM $\alpha(n_1, \ldots, n_k)$	ruší pole se jménem α a vytváří pole α s dimensemi $n_1 \ldots n_k$ inicializuje hodnoty na Ø,
DIM $\alpha\$(n_1, \ldots, n_k)$	ruší pole nebo řetězec se jménem $\alpha$$, vytváří pole znaků dimensí $n_1 \ldots n_k$. Inicializuje všechny hodnoty na pole znaků. Lze považovat jako pole řetězců pevné délky n_k s k-1 rozměry $n_1 \ldots n_{k-1}$. Pokud není dostatek místa objeví se chyba K. Pole je nedefinované pokud není udán příkaz DIM.
DRAW x, y DRAW x, y, z	DRAW x, y, Ø kreslí přímku z běžné tiskové pozice, posouvá x vodorovně a y svisle. Případně kreslí oblouk o úhlu z. Pokud vychází tisk mimo obrazovku objeví se chyba B.
ERASE FLASH	nepracuje bez microdrive definuje, zda znak bude blikat či ne. n=Ø znak je stálý, n=1 znak bliká, n=8 nedá se měnit.
FOR α =xTOy FOR α =xTOy STEPz	FOR α =xTOySTEP 1 ruší proměnnou α a stanoví novou proměnnou s hodnotou x a limitem y s krokem z a vytváří adresu smyčky. Kontroluje, zda inicializační hodnota je větší jak Ø a zda krok je větší jak Ø a obráceně. Pokud chybí příkaz NEXT, nastává chyba 1. (Podívejte se na NEXT.) Chyba 4 se objeví, pokud není místo pro kontrolní proměnnou.
FORMAT f	nepracuje bez microdrive
GOSUB n	dává číslo příkazu GOSUB a funguje jako GO TO n. Chyba 4 se objeví, pokud schází příkaz RETURN.
GO TO n IF x THEN s	skáče na řádek n (když není použit, tak na řádek vyšší). pokud x je pravdivé (ne Ø), pak se provádí s. Všimněte si, že zahrnuje všechny příkazy až do konce řádku. Není dovolena forma "IF x THEN číslo řádku.
INK n	stanoví barvu inkoustu znaků tištěných sekvenčně. N je v rozsahu Ø až 7. 8 je průhledné a 9 kontrastní. Podívejte se do přílohy B na televizní obrazovku. Pokud n není v rozsahu Ø až 9, objeví se chyba K.
INPUT ...	"..." je sekvencí položek INPUT, oddělených jako v příkaze PRINT čárkami, středníky nebo apostrofy. Položky INPUT lze užít: (I) některá položka PRINT nezačíná písmenem (II) jméno proměnné (III) LINE, pak řetězcové jméno

položky PRINT a separátory (;) se provádějí přesně jako v PRINT, s výjimkou, že se tisknou v dolní části obrazu. Pro příklad (II) se počítač zastaví a čeká na výraz z klávesnice. Tato hodnota je pak přiřazena k dané proměnné. Vstup je avizován obvyklým způsobem a syntaktické chyby vyvolají svítící ?. Pro řetězcové proměnné se objeví automaticky dvojí uvozovky (které lze i smazat pokud je nezbytné). Pokud je první znak STOP, program se zastaví se zprávou H. (III) je podobné (II) s výjimkou, že vstup je považován jako literál bez uvozovek a mechanismus STOP nepracuje. K zastavení je nutno použít řízení kontrolních znaků pro tis. Pokud n=Ø jsou znaky tištěné normálně jako inkoust na papírové barvě. Pokud n=1 znaky jsou tištěny inversně, tj. barvou papíru na inkoustové barvě. Podívejte se do přílohy B -televizní obraz. Pokud n není ani 1, objeví se chyba K.

INVERSE n

přiřazení hodnoty e do proměnné v. Příkaz LET nemůže být vynechán. Prostá proměnná není definována, pokud neproběhne příkaz LET, READ nebo INPUT. Pokud je v indexovaná proměnná nebo část řetězce, je přiřazení "Procrusteam" (pevné délky): řetězcová hodnota je zkrácena nebo doplněna mezerami vpravo, aby byla stejné délky jako v.

LET v=e

LIST Ø
listuje program v horní části obrazovky, začíná prvním řádkem, jehož číslo je nejméně n a n je běžným řádkem.

LIST
LIST n

LLIST Ø
obdoba LIST na tiskárně.

LLIST
LLIST n

nahraje do paměti program a proměnné (z magnetofonu).

LOAD f

nahraje numerické pole

LOAD f DATA ()

nahraje do paměti pole $.

LOAD f DATA$ ()

nahraje do paměti nejvýše n byte začínající na adrese m

LOAD f CODE m,n

nahraje byte do paměti od adresy m

LOAD f CODE m

nahraje zpět bytes na adresu, odkud byly vytaženy.

LOAD f CODE

LOAD f CODE 16384,6912.

LOAD f SCREEN$

hledá zmíněné pole na pásku, nahraje ho a zruší předcházející verzi v paměti. Viz kapitola 2Ø;

Obdoba PRINT, ale s použitím tiskárny

LPRINT

podobná LOAD f, ale neruší řádky starého programu a proměnných, s výjimkou těch, které mají stejná jména nebo čísla řádků.

MERGE f

nepracuje bez microdrive

startuje systém BASICU, ruší program a proměnné vč. systémových proměnných RAMBOT a zachovává systémové proměnné UDG, P RAMT, RASP a PIP

MOVE f_1, f_2

NEW

(I) hledá kontrolní proměnnou
(II) přičítá krok k její hodnotě
(III) pokud STEP $\geq$ Ø a hodnota $>$ limit nebo STEP $<$ Ø a hodnota $<$ limit, vrací se na počátek smyčky.

NEXT

Chyba 2 - pokud α neexistuje
Chyba 1 - α není kontrolní proměnná.

nepracuje bez microdrive

OPEN #

vstupuje byte n v části m na úrovni procesoru (natahuje registr bc s m a registr s n provádí instrukci out (c),a.)

OUT m,n

Ø $\leq$ m $\leq$ 65535, -255 $\leq$ n $\leq$ 255, jinak chyba B

kontrolní znak pro přepsání znaků již vytištěných. Pokud n=Ø, nové znaky vynechávají znaky staré v dané pozici. Pokud n=1, jsou staré znaky spojeny se znaky novými. Viz příloha B. Pokud n není Ø ani 1, je chyba K.

OVER n

jako INK, ale řídí barvu papíru (pozadí)

PAPER n

zastavuje program a zobrazuje obrazovkové pole po n cyklů (5Ø nebo 6Ø cyklů za 1 s) nebo pokud je stlačeno tlačítko, program běží dále. PAUSE Ø zastaví program až do dalšího stlačení tlačítka. Ø $\leq$ n $\leq$ 65535 - jinak se hlásí chyba B.

PAUSE n

PLOT e;m,n — tiskne bod inkoustovou barvou (závisí na OVER a INVERSE) v bodě o souřadnicích (|m|,|n|) mění další pozici PLOT. Pokud nespecifikují položky barev jinak, inkoustová barva ve znakové pozici říká, že tiskový element je změněn na barvu inkoustu a další veličiny (barva, PAPER, blikání a jas) se nemění. $Ø <= |m| < =255$, $Ø <= |n| < =175$. Jinak se objeví chyba B.

POKE m, n — píše hodnoty n do byte na adrese m. $Ø <= m <= 65535$, $-255 <= n <= 255$, jinak chyba B.

PRINT ... — "..." je sekvence položek v PRINT, oddělených čárkami, středníky nebo apostrofy a jsou zapsány do pole pro televizní zobrazení. Středník mezi dvěma položkami nemá účinek a je užit zřídka. Další znak v tisku je čárka a apostrof. Na konci příkazu PRINT, pokud nekončí středníkem, čárkou nebo apostrofem, je výstupem znak ENTER. Printová položka může být:
(I) prázdná, tj. nic
(II) numerický výraz
První je tištěno znaménko - pokud je hodnota záporná. Nyní ať x je tvarem hodnoty. Pokud $x < =10^{-5}$ nebo $x > =10^{13}$, je tisk ve specifické notaci. Část mantisy má 8 čísel a desetinnou tečku. Exponentní část je E následované + nebo - a jedním nebo dvěma čísly. Jinak je x tištěno v obyčejné decimální notaci na 8 platných čísel bez Ø za desetinnou tečkou. Decimální tečka je vždy jako první znak, pokud před desetinnou tečkou není číslo př. .Ø3. Nula se tiskne jako Ø.
(III) řetězcové vyjádření. Uvozovky v řetězci jsou zdvojeny, je možná mezera před i za. Neznámý znak je tištěn jako otazník.
(IV) AT m,n Výstup AT - kontrolní znak následovaný 1 byte pro m (řádek) a n (sloupec).
(V) TAB n Výstup TAB - kontrolní znak následovaný 2 bytes pro n (méně významný je první byte), TAB sloupec.
(VI) položka barvy může mít formu PAPER, INK, FLASH, BRIGHT, INVERSE, OVER.

RANDOMIZE
RANDOMIZE n — RANDOMIZE Ø ustavuje systémovou proměnnou (SEED) ke generování další hodnoty RND. Pokud $n <> Ø$, SEED dostává hodnotu n, pokud n=Ø, dostává hodnotu další systémové proměnné (FRAMES), počítá cykly zobrazení v televizi a měly by být téměř náhodné. RANDOMIZE se objevuje na klávesnici jako RAND. Pokud n není v rozsahu od Ø do 65535, objeví se chyba B.

READ v1,v2 ...vk — přiřazení proměnných za užití výrazů v seznamu DATA. Je-li výraz typován chybně objeví se chyba C. Chyba E se objeví není-li v seznamu DATA další hodnota pro přiřazení v READ

REM ... — nemá v programu efekt. "..." může být jakákoliv posloupnost znaků s výjimkou ENTER. Může zahrnout i :, takže v daném řádku je pouze příkaz REM.

RESTORE
RESTORE n — RESTORE Ø znovu zpřístupňuje seznam DATA za číslem n (číslo řádku) pro další příkaz READ.

RETURN — odvolává se k příkazu GO SUB a vrací program na další řádek za GO SUB. Pokud je v programu chyba, objeví se hlášení 7.

RUN
RUN n — RUN Ø CLEAR a pak GO TO n

SAVE f — ukládá program a proměnné na pásek

SAVE f LINE n — ukládá program a proměnné a po jejich zpětném natažení do počítače automaticky skáče na řádek n.

SAVE f DATA () — ukládá numerické pole

SAVE f DATA$ () — ukládá znakové pole

SAVE f CODE m,n — ukládá n bytes od adresy m

SAVE f SCREEN$ — SAVE f CODE 16384,6912. Ukládá informaci se jménem f, k chybě F dochází je-li f prázdný řetězec nebo má víc jak 1Ø znaků. Viz kapitola 2Ø,

STOP	zastavuje program se zprávou 9, CONTINUE zajistí pokračování na dalším příkaze.
VERIFY	stejné jako LOAD s výjimkou, že data se nenatahují do RAM, ale porovnávají se s jejím obsahem. Pokud se data sobě nerovnají objeví se chyba R.

PŘÍLOHA D
Příklady programů
Programy ukazují schopnosti ZX Spectrum.

První z těchto programů vyžaduje na vstupu datum a k tomuto datu přiřadí den v týdnu podle skutečnosti.

Další program přepočítává jednotky: yards, fut, inches.

Třetí program - hází mincí.

Čtvrtý program - hra PANGOLINS. Myslíte si zvíře a počítač se snaží uhádnout co to je tím, že klade otázky, na které odpovídáte ano a ne. Počítač vaše zvíře nikdy neviděl a tak se jeho otázkám nedivte.

Další program kteslí anglickou vlajku.
Pokud nejste z Británie, nakreslete si vaši vlajku. Trikolory jsou celkem snadné, ačkoliv některé barvy, např. oranžová na vlajce Irska může působit potíže. Pokud jste Američan, měli byste být schopni vyplnit vnitřek vlajky hvězdičkami.

Další program hraje hru "hangman - oběšenec". Jeden hráč stanoví slovo a další hádají.

PŘÍLOHA E
Binární a hexadecimální čísla.

Tato příloha popisuje, jak s použitím binárního systému provádí počítač operace. Většina evropských jazyků počítá s použitím desítkové soustavy. Např.:
dvacet, dvacet jedna ..... dvacet devět,
třicet, třicet jedna ..... třicet devět,
čtyřicet, čtyřicet jedna ..... čtyřicet devět atd.
avšak podstatně přehlednější je to s užitím arabských čísel. Jediným důvodem pro rozšíření desítkové soustavy byl počet prstů na rukou. Místo decimálního systému se základem 1Ø, užívá počítač systém binární - hexadecimální (zkráceně hexa), který je založen na základě 16. Protože je dostupných pouze 1Ø čísel užíváme i písmena a to A, B, C, D, E a F.
Hexadecimální systém pak vypadá následovně:

Hexa	Desítkově
Ø	nula
1	jedna
2	dvě
.	.
.	.
.	.
9	devět
dosud to bylo stejné, nyní to bude odlišné	
A	deset
B	jedenáct
C	dvanáct
D	třináct
E	čtrnáct
F	patnáct
10	šestnáct
11	sedmnáct
.	.
.	.
.	.

19	dvacetpět
1A	dvacet šest
1B	dvacetsedm
.	.
.	.
1F	třicet jedna
2Ø	třicet dva
21	třicet tři
.	.
.	.
9E	štopadesát osm
9F	sto padesát devět
AO	sto šedesát
A1	sto šedesát jedna
.	.
.	.
FE	dvěstěpadesát čtyři
FF	dvěstěpadesátpět
1ØØ	dvěstěpadesát šest

Pokud užíváte hexadecimální notaci, píšte na konci čísla "h" a říkáme hexa. Budete se divit, co to má do činění s počítačem. Ve skutečnosti však počítače znají pouze dvě číslice, preprezentované nízkým napětím nebo proudem (Ø) a a vysokým napětím (1). Nazývá se to binární systém a dvě binární čísla se nazývají bity - tak bit je buď Ø nebo 1.

Počítání v různých systémech začíná takto:

Česky	decimálně	hexadecimálně	binárně	
nula	Ø	Ø	Ø nebo	ØØØØ
jedna	1	1	1	ØØØ1
dvě	2	2	1Ø	ØØ1Ø
tři	3	3	11	ØØ11
čtyři	4	4	1ØØ	Ø1ØØ
pět	5	5	1Ø1	Ø1Ø1
šest	6	6	11Ø	Ø11Ø
sedm	7	7	111	Ø111
osm	8	8	1ØØØ	
devět	9	9	1ØØ1	
deset	1Ø	A	1Ø1Ø	
jedenáct	11	B	1Ø11	
dvanáct	12	C	11ØØ	
třináct	13	D	11Ø1	
čtrnáct	14	F	111Ø	
patnáct	15	F	1111	
šestnáct	16	1Ø	1ØØØØ	

Ke změně binárního čísla na hexadecimální rozdělte binární číslo zprava do skupin po čtyřech bitech a pak každou skupinu změňte na odpovídající 16-ové číslo. Bity v počítači jsou většinou sloučeny po osmi - bytech (čti bajtech). Jeden byte může reprezentovat číslo od Ø do 255 (1111 1111 binárně nebo FFh) nebo jakýkoliv znak ze znáů ZX Spectra. Jeho hodnota může být napsána dvěma hexa čísly. Dva byty ce mohou spojit dohromady, aby vytvořily slovo. Slovo se může psát s užitím 16 bitů nebo 4 číslic a reprezentuje číslo od Ø do $2^{16}-1=65535$. Jeden byte je vždy 8 bitů, ale slova mají různou délku, to záleží na počítači. Zápis BIN v kapitole 14 poskytuje způsob psaní čísel binárně. BIN Ø reprezentuje Ø, BIN 1 reprezentuje 1, Bin 1Ø reprezentuje 2 atd. Můžete použít pouze Ø a 1 a číslo musí být nezáporné a celé, nemůžete např. psát 'BIN -11', ale musíte psát '-BIN 11'. Číslo nesmí být větší než 65535 - nemůžeme použít více jak 16 bitů.

Funkce ATTR je skutečně binární. Pokud kontrolujete výsledek s užitím ATTR, převedete ho do binárního tvaru a jednotlivé bity znamenají:

první je 1 pro blikání, Ø pro stálý znak
druhý je 1 pro jasný, Ø pro normální jas
třetí, čtvrtý a pátý - kody pro papír (binárně)
poslední tři bity - kody pro inkoust (binárně).

Kódy barev jsou rovněž binární - každý kód lze napsat pomocí tří bitů, první pro zelenou, druhý pro červenou a třetí pro modrou. Pro černou jsou všechny bity Ø. Čisté barvy-zelená, červená a modrá mají pouze 1 bit, kody jsou 1ØØ, Ø1Ø, ØØ1 - binárně 4, 2, 1.
Další barvy jsou vytvořeny z těchto kódů, takže jejich kódy mají binárně dva nebo více bitů.