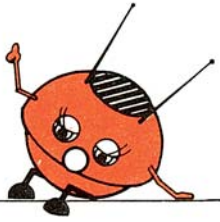
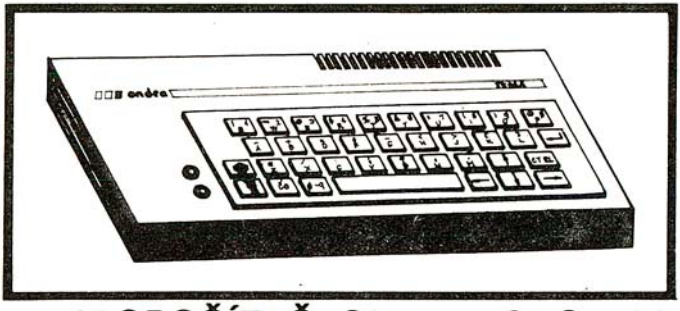


TESLA

UŽIVATELSKÁ PŘÍRUČKA MIKROPOČÍTAČE

ondra



MIKROPOČÍTAČ ONDRA SPO 186

4 BASIC

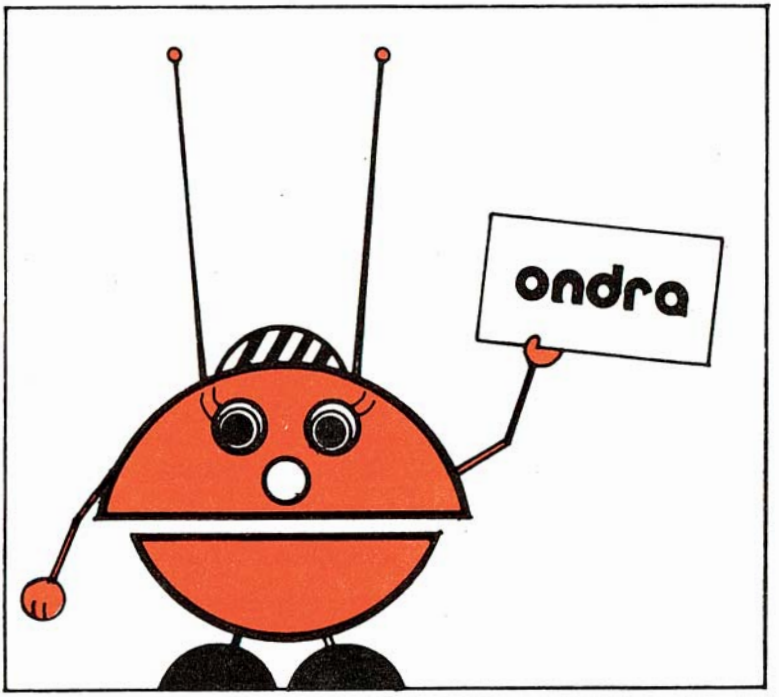
6XV 139 02

1 ÚVOD

Tato příručka ve stručnosti popisuje příkazy a povely jazyka Basic implementovaného na mikropočítači ONDRA a není učebnicí jazyka Basic.

BASIC EXP/G V5.0 byl vyvinut z Basicu V4.0 ze systému SAPI 1. Tento Basic byl rozšířen o nové příkazy a o základy strukturovaného programování, které známe z moderních programovacích jazyků (např. Pascal). Mikropočítač ONDRA má standardně zabudovanou grafiku,a proto byl Basic doplněn i o grafické příkazy.

Jazyk Basic byl vyvinut na Dathmouth College v USA pro výuku studentů v programování. BASIC je zkratkou z angličtiny Beginners All-purpose Symbolic Instrukction Code a ve volném překladu znamená jazyk pro začátečníky a všemožné použití. Programovací jazyk Basic na rozdíl od jiných vyšších programovacích jazyků nebyl normován, a proto se můžeme setkat s tolika různými verzemi jazyka Basic. Teprve před několika roky byl předložen návrh na normalizaci jazyka Basic, který vychází z Basicu vytvoře-


ondra

ného firmou Microsoft. Programovací jazyk Basic EXP V5.0/G rovněž vychází z tohoto Basicu.

Překladač jazyka Basic na mikropočítači ONDRA pracuje jako interpret. To znamená, že program napsaný v jazyce Basic se nepřekládá do strojového kódu, který by se zaváděl do paměti a pak se spustil, nýbrž do zprostředkujícího kódu. V průběhu provádění programu se postupně rozkódovávají příkazy a pomocí podprogramů interpretu se tyto příkazy provádí. Proto musí být interpret po celou dobu provádění programu přítomen v operační paměti.

BASIC V5.0 poskytuje uživateli efektivní prostředky pro vytváření, údržbu, ladění a provádění programu interaktivním způsobem. Má takové vlastnosti, že uživatel prakticky vystačí se samotným Basicem. Funkce vytváření a editace zdrojových programů jsou již obsaženy v samotném interpretu jazyka Basic.

Tyto přednosti jsou ovšem za cenu větší paměťové náročnosti (interpret musí být trvale v paměti) a rychlosti provádění programu. Malé mikropočítačové systémy s kazetovým magnetofonem používají interpret Basicu a nikoliv překladač do strojového kódu. Překladače se používa-

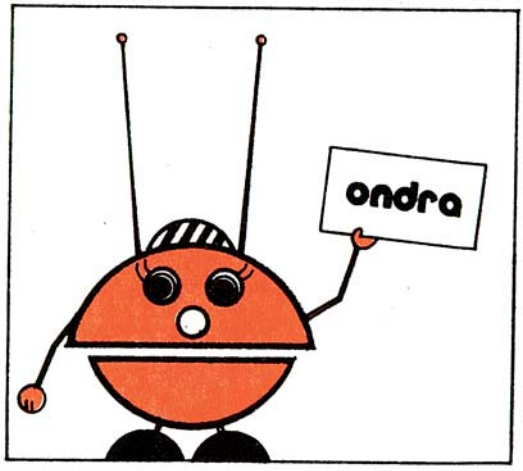
ji u vyšších systémů s disketami jako další verze Basicu pro odladěné programy.

Basic je vhodný pro výuku programování a pro programátory, kteří potřebují rychle vyřešit daný problém a odladit program. Dále lze s výhodou využít možností, které poskytuje Basic, při práci s řetězci. Basic umožňuje připojení podprogramů zapsaných ve strojovém kódu a tím zrychlit provádění programu. Použití podprogramů ve strojovém kódu vyžaduje podrobné znalosti z výpočetní techniky se zaměřením na mikropočítače a programování v assembleru. K mikropočítači ONDRA je i programové vybavení pro programování v assembleru, které je v souboru TOOL.

Interpret jazyka Basic pracuje ve dvou režimech práce - dialogovém a programovém. V dialogovém režimu se příkazy píší na řádek bez čísla řádku a po ukončení se příkazy okamžitě provedou. Mikropočítač může pracovat jako kalkulátor. V programovém režimu se nejdříve vytvoří program (zápisem z klávesnice nebo zavedením z vnější paměti počítače) a pak se spustí příkazem RUN.

Předpokladem práce s mikropočítačem ONDRA je dobrá znalost příručky "návod k obsluze". V této příručce je popsáno

sestavení mikropočítačového systému a dále se dozvíte, jak se obsluhuje počítač a jak se zavádí a spouští program z kazety. Proto si ve vlastním zájmu nejdříve prostudujte Návod k obsluze a pak můžete začít se zkoušením a programováním v jazyce Basic.



2 CHARAKTERISTIKA BASICu V5.0/G

Interpret jazyka Basic je dodáván na kazetě. V paměti mikropočítače ONDRA je uložen od adresy 1000H a zabírá 12 kB paměti (do adresy 3FFFH).

Basic pracuje s čísly řádků od 1 do 64999. Maximální délka příkazového řádku je 132 znaků včetně čísla řádků, oddělovačů a mezer. Na jeden řádek je možno zapsat i více příkazů vzájemně oddělených dvojtečkou (":"), oddělovačem příkazů. Je-li více příkazů na řádku, pak se příkazy provádí postupně zleva doprava. Basic neumožňuje skok na příkazy uprostřed řádku, skoky jsou přípustné pouze na začátek příkazové řádky. Vícepříkazové řádky zabírají méně místa paměti a jsou rychlejší, než stejný program rozepsaný na více řádek. Vícepříkazové řádky jsou někdy méně přehledné. Příkazy uvedené na jednom řádku za nepodmíněným skokem GO TO se neprovedou. Příkazový řádek je ukončen znakem "↵" návrat vozu ("CR"). V dalším textu budeme používat označení konce řádku "CR".

Pro zvýšení přehlednosti programu je možno používat poznámky, a to buď poznámku jako celý řádek příkazem REM a

nebo lze použít poznámku i na příkazovém řádku, která je uvedena znakem vykřičník ("!"). Veškerý text za vykřičníkem do konce řádku je brán jako poznámka. Poznámky se ukládají do operační paměti, a proto pozor u dlouhých programů, že překročíme rozsah paměti. Proto doporučujeme mít dvě verze programu, jednu při ladění s poznámkami pro orientaci v programu a druhou po odladění, kdy poznámky v programu zrušíme. Při provozování programu používáme verzi bez poznámek a při opravách a editaci používáme verzi s poznámkami.

BASIC EXP V5.0/G může pracovat ve dvou režimech:

a) Dialogový režim - příkazy se provádí bezprostředně po zadání příkazového řádku. Příkazové řádky jsou bez čísla řádku. Některé příkazy nelze zadávat v dialogovém režimu.

b) Programový režim - příkazy se zadávají jako program s čísly řádků. V programovém režimu platí celý příkazový soubor. Program se nejdříve předzpracuje do speciálního vnitřního tvaru, který se po povelu RUN provádí.

Basic pracuje s čísly v pohyblivé řádové čárce jednoduché délky (typu REAL), tj. 4 Byte s přesností sedmi dekadických číslic (osmá číslice se zaokrouhluje). Číslo mů-

že být v rozsahu od 1.70141E-38 až 1.70141E+38.

Čísla se mohou zapisovat i jako celočíselná, tj. bez desetinné tečky nebo jako desetinná čísla v přímém tvaru (tj. celá část čísla, desetinná tečka a desetinné číslo, např. 28.25) nebo v exponenciálním tvaru: ±XXE+NN nebo ± XXE-NN, kde XX je číslo v přímém tvaru.

E označuje exponenciální tvar a značí "krát 10 na N-tou"

+ - znaménka čísla nebo exponentů (znaménko + není povinné)

NN maximálně dvoumístná hodnota exponentu (celé číslo) a musí být vždy uveden i když je nulový ($0 \leq NN \leq 38$)

Př. stejné číslo 28.25 lze v exponenciálním tvaru zapsat 28.25E0 nebo 0.2825E+2, 2825E-2 a další kombinace.

Výstupní čísla jsou zobrazována v "normálním" (přímém) tvaru, jsou-li v rozsahu $0.1 < X < 999999$. V ostatních případech jsou čísla zobrazována v exponenciálním tvaru.

V případě, že potřebujeme formátovaný výstup čísel, lze v příkazu pro výstup PRINT použít příkaz USING, který provádí formátování výstupních čísel podle zadaného vzoru, s možností určení polohy desetinné tečky.

Označení proměnných (identifikátory) mohou obsahovat pouze písmena a číslice. První znak musí být písmeno. Délka identifikátoru není omezena, ale významné jsou pouze první dva znaky. Tak např. identifikátor AB je totožný s identifikátorem ABC (protože Basic bere první dva znaky AB a znak C již ignoruje) a přiřadí je stejné proměnné.

Pozor! Začátek nebo jiná část identifikátoru nesmí obsahovat "klíčové slovo" (tj. příkaz nebo povel). Př. identifikátor ABCDEF je chybný, protože obsahuje klíčové slovo (příkaz DEF, rovněž identifikátor TON je chybný, protože obsahuje klíčové slovo TO (část příkazu GO TO).

Identifikátory pro znakové řetězce jsou ukončeny znakem měny (buď dolar "$" nebo znakem "%"). Identifikátory pro číselné a řetězcové proměnné mohou být stejné a přitom označují různé proměnné (dáno znakem "měny").

Pro řetězové identifikátory platí stejné omezení jako pro numerické identifikátory. Basic V5.0 může pracovat až s osmirozměrnými poli, která mohou být buď numerická nebo řetězcová. Pro identifikátory polí platí výše uvedená pravidla. Identifikátory polí se mohou shodovat s iden-

tifikátory jednoduchých proměnných. Rozměr pole se deklaruje pomocí příkazu DIM. Uvádí se horní hranice pole, dolní je implicitně "0". Pokud není provedena deklarace pole pomocí příkazu DIM, provede se při prvním výskytu prvku pole implicitní deklarace pole s horní hranicí 10 (tj. pole o 11-ti prvcích). Počet prvků pole je horní mez +1. Doporučujeme používat vždy pro deklaraci pole příkaz DIM, protože se včas rezervuje prostor pro pole v paměti počítače. Zvlášť důležité je to u menších vícerozměrných polí, kdy se deklarací šetří paměť.

Basic V5.0 má příkaz pro deklaraci jednoduchých proměnných VAR. Deklarací proměnných se zrychlí provádění programu a současně získáme přehled o použitých proměnných. Příkaz VAR má ještě další význam pro předávání výsledků z podprogramů volaných jménem.

Basic V5.0 má <u>povely</u> pro řízení, sledování chodu programu a pro pomocné funkce; některé povely nelze uvádět v programu.

<u>příkazy</u> se provádí při chodu programu nebo v dialogovém režimu

funkce implicitně zabudované v interpretu jazyka Basic pro numerické a řetězcové proměnné. Výsledkem provedení funkce je vždy jedno číslo nebo řetězec.

operátory jsou aritmetické, relační a logické

Aritmetické operátory jsou: + ... plus pro sečítání
- ... mínus pro odečítání
* ... krát pro násobení
/ ... lomítko pro dělení
^ ... šipka nahoru pro umocňování

Aritmetické operátory mají prioritu. Nejvyšší má mocnění, na druhém místě jsou operace násobení a dělení a na třetím je sečítání a odečítání. Například aritmetický výpočet zapsaný 2+3*4^2 dá výsledek 50. Nejdříve se provede mocnění 4 na 2=16; dále se vynásobí 16*3=48 a nakonec se provede sečtení 48+2=50.

Pokud potřebujeme změnit posloupnost provádění operací, musíme použít kulaté otevírací a zavírací závorky "(,)". Výraz v závorkách se vyhodnotí nejdříve podle výše

uvedených pravidel. Při stejné prioritě operátorů se operátory provádí zleva doprava.

Relační operátory: = ... rovnost (shoda)
> ... větší
< ... menší
>= ... větší nebo rovnost
<= ... menší nebo rovnost
<> ... nerovnost (neshodný)

Relační operátory se používají mezi dvěma operandy (proměnná nebo konstanta) v příkazech podmíněného skoku IF podmínka THEN příkaz nebo cyklech typu WHILE, REPEAT. Výsledkem relačních operací je logická hodnota "true" (pravda, podmínka je splněna) nebo "false" (nepravda, podmínka není splněna).

Pozor, znak "=" (rovná se) se používá i v jiném významu v přiřazovacích příkazech a znamená - proměnné na levé straně přiřaď hodnotu výrazu na pravé straně. Programovací jazyk Pascal již rozlišuje relační operátor "=" a přiřazovací příkaz, který je složen ze dvou znaků ":=".

Logické operátory: NOT ... negace, logická inverze
OR ... nebo, logický součet
AND ... i, logický součin
XOR ... exclusiv OR, logická neekvivalence

Logický operátor NOT je unární, tj. platí pouze pro jednu proměnnou, před kterou je uveden. Ostatní logické operátory jsou binární, tj. provádí operace se dvěma proměnnými. Logické operace se mohou provádět mezi logickými proměnnými (např. ve složených logických podmínkách) nebo mezi numerickými proměnnými.

Numerické proměnné se nejdříve převedou na celočíselné proměnné (16-ti bitové číslo) a pak se provede příslušná logická operace. Pokud je číslo větší než 32767, pak je hlášena chyba a operace se neprovede. Pokud je použito necelé číslo, pak se před použitím desetinná část odřízne (nezaokrouhluje se) a číslo se převede na celočíselné (typu integer).

U logických operátorů platí implicitní priority podobné jako u aritmetických operátorů. Nejvyšší prioritu má unární operátor NOT, dále pak následují AND (logický součin), OR (logický součet) a nejnižší prioritu má XOR

(nonekvivalence). Jsou-li kombinovány aritmetické a logické operátory, pak aritmetické operátory mají vyšší prioritu než logické operátory.

2.1. Rozšíření Basicu V5.0

Basic EXP V5.0/G byl rozšířen oproti Basicu EX V4.0 o grafické příkazy a další příkazy. Některá rozšíření nemají pro mikropočítač ONDRA podstatný význam, ale jsou zde zařazeny z důvodu kompatibility se systémem SAPI 1. Jsou to především příkazy pro řízení přerušení a přímé řízení portů. Programy v Basicu jsou plně přenositelné mezi systémem SAPI 1 a mikropočítačem ONDRA, pokud nepoužívají speciální funkce (např. podprogramy v assembleru, řízení přerušení a přímé vstupy a výstupy z portů), které jsou podmíněny rozdílnou technickou konstrukcí obou mikropočítačů.

a) Řídicí programové struktury obdobné Pascalu

- deklarace bloků příkazů (BEG-BEND); používá se u složených podmíněných skoků
- podmíněné skoky IF-THEN-ELSE, lze vnořovat do více úrovní a mohou být na více řádcích. Pro víceřádko-

vé je nutno použít bloky (BEG-BEND).

- příkazy cyklu: REPEAT - UNTIL

  WHILE-DO-WEND

- deklarace procedur (podprogramů) jménem s parametry a jejich volání jménem. Možnost zadávat vstupní a výstupní parametry.
- deklarace procedury se provádí příkazem PROC jméno (parl, par2,...,parN). Volání se provádí jménem v příkazu GOSUB jméno.
- deklarace víceřádkových uživatelských funkcí s parametry. Funkce mohou být jak pro numerické, tak i pro řetězcové proměnné.

  Funkce se deklaruje příkazem FUNC jméno (par1,par2, ...,parN) a volá se jménem, před kterým je znak "&", tj. jméno.
- Basic V5.0 obsahuje přepínače (obdoba CASE v Pascalu) pro skoky ON-GOTO,nebo pro volání podprogramů ON-GOSUB. Ve volání podprogramů lze použít číslo řádku podprogramu nebo jméno. Oba přepínače jsou doplněny o příkaz ELSE, tj. skok, je-li přepínač mimo rozsah.

b) Nové ediční funkce

- možnost automatického číslování řádků programu, při tvorbě (zápisu) programu. Příkaz AUTO umožní zadání počátečního čísla a kroku číslování řádků.
- příkaz SEEK umožní vyhledat ve zdrojovém textu programu zadanou část textu a přechod do editoru
- příkaz "@"pro vyvolání editoru (bez zadání čísla řádku) po chybě v průběhu provádění nebo opravy programu
- příkaz RENUMBER pro přečíslování programových řádků a odkazů na čísla řádků v příkazech GOTO, GOSUB, THEN atd.
- příkaz DELETE pro zrušení části nebo celého programu

c) Další rozšíření

- příkaz USING pro formátování výstupu numerických dat v příkazu PRINT podle zadaného vzoru
- příkaz DEG ON/OFF pro přepínání mezi radiány a stupni v trigonometrických funkcích
- trasovací příkazy a možnost trasovat po krocích (příkazových řádcích)
- nová filozofie ovládání vstup/výstupních souborů na kazetě. Možnost zadávání jména souboru při volání (tj. funkce FIND a OPEN). Možnost ukládat a číst předkompilované (tj. ve vnitřní formě) programy z kazety. Mož-

nost soukromých programů, které lze zavést z kazety bez možnosti jejich výpisů, spojování více programů a možnost uložit nebo zavést grafickou část paměti z kakazety.

- zvětšený rozsah čísel řádků od 1 do 64999

d) Možnost generování některých klíčových slov při zápisu programu z klávesnice mikropočítače nebo povelů pro řízení

e) Možnost vytváření "oken" na obrazovce, tj. části obrazovky pro texty.
Oken může být větší počet.

f) Velice účinné grafické příkazy. Možnost slučování grafických příkazů do jednoho složeného příkazu (viz příkaz DRAW). Speciální příkaz pro grafický vstup DRAW INPUT, kterým je možnost vytvářet řídící řetězce podle pohybu grafického kurzoru na obrazovce.

g) Modul pro řízení přerušení (viz příkazy INTRP).
Tyto příkazy nemají na mikropočítači ONDRA přímo význam, ale lze je použít, když připravujeme programy v Basicu pro systém SAPI 1.

Basic EXP V5.0/G poskytuje uživatelům velice účinný prostředek pro tvorbu programů a jejich ladění. Zabudované grafické příkazy Vám umožní zobrazovat různé obrázky a výsledky zpracování ve formě grafů. Grafický výstup rozšiřuje i možnost použití počítače pro různé hry, které přitahují mládež k výpočetní technice.

3 ZAVEDENÍ A SPUŠTĚNÍ BASICU

Předpokládáme, že jste se seznámili s příručkou k obsluze mikropočítače ONDRA a že máte mikropočítačový systém zapojen podle této příručky.

V této kapitole si stručně zopakujeme zavádění programů a interpretu jazyka Basic z kazety a jejich spuštění. Uvádíme tuto kapitolu proto, aby jste nemuseli neustále listovat v různých příručkách a vystačili při práci v Basicu s touto příručkou. Naučte se zapojovat systém podle návodu, věnujte pozornost upozorněním. Po zacvičení se naučíte sestavovat systém i bez návodu. Po sestavení a spuštění mikropočítače ONDRA se provede studený start mikropočítače ONDRA, kdy se systémové programy z paměti EPROM přepíší do paměti RAM, systém se nastaví do základního stavu. Rozsvítí se obě LED diody u klávesnice, klávesnice je nastavena na vstup velkých písmen a systém se na obrazovce televizního displeje ohlásí zprávou "Ondra V.5" a na dalším řádku se zobrazí znak "." (tečka), který signalizuje připravenost monitoru přijímat povely. Při inicializaci systému se provede i akustická signalizace.

3.1. Zavedení interpretu Basicu do paměti

Překladač (interpret) jazyka Basic EXP V5.0/G je dodáván na distribuční kazetě jako binární soubor. Kapacita standardně zabudovaných paměti EPROM neumožňuje trvalé umístění Basicu v této paměti, a proto musíme před použitím Basic zavést z kazety do operační paměti počítače ONDRA. Operační systém mikropočítače ONDRA má dva způsoby pro zavádění programů do paměti počítače: první zkrácený, kdy se program jedním povelem zavede do paměti a spustí; druhý způsob je konverzační a vyžaduje spolupráci obsluhy s počítačem. Oba způsoby jsou rovnocenné, ale první je jednodušší a je určen dětem a mládeži pro jednoduché a snadné zavádění programů.

3.1.1. Zkrácený způsob zavedení Basicu

Tento způsob lze použít u distribučních kazet, když systémové programy jsou uloženy na začátku kazety. Tento způsob lze použít i když nepoužíváme distribuční kazety. Princip tohoto způsobu zavádění spočívá v tom, že kazeta v magnetofonu je nastavena před požadovaným sou-

borem a po zadání příkazu se provede čtení daného souboru na kazetě s binárním programem. Není-li to binární soubor, pak se program nezavede a řízení se vrátí zpět monitoru. Při tomto povelu se systém nedotazuje na jméno souboru, zavádí soubor z kazety, který přečte jako první (a pokud splňuje podmínky binárního systémového programu)zavedený program spustí.

Posloupnost příkazů:

.L čtení - hotovo? ⟨CR⟩ ... zkrácený příkaz pro zavedení binárního souboru. Systém vypíše text čteni - hotovo a očekává nastavení magnetofonu na čtení dat z kazety. Po nastavení magnetofonu stiskne libovolnou klávesu a systém začne číst soubor z kazety. Je-li soubor správně zaveden do paměti, spustí se zavedený program.

*BASIC - EXP V5.0/G *... ohlášení se zavedeného in-
(C) 1986 TESLA DIZ terpretu jazyka BASIC. Program
READY se hlásí jménem a verzí a připravenost přijímat povely a

příkazy jazyka Basic hlásí slovem

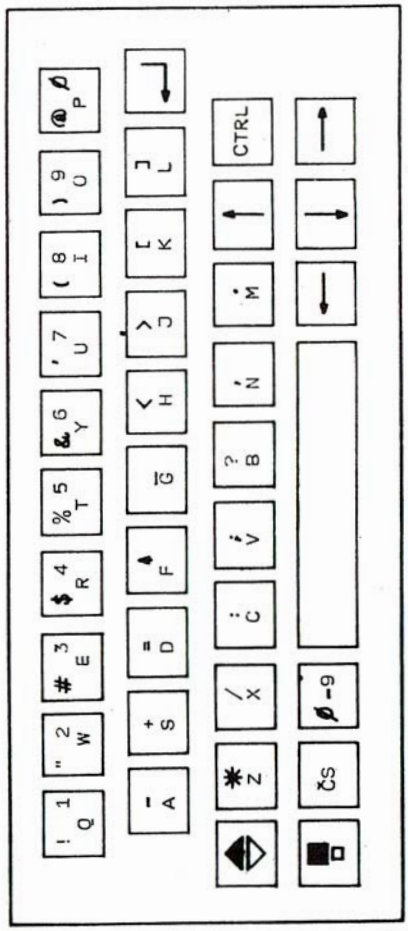
: "READY" a znakem ":" dvojtečka, který který označuje režim vstupů příkazů a povelů.

Poznámka:

Texty vypisované systémem budeme v této kapitole označovat podtržením, abychom odlišili texty zadávané z klávesnice mikropočítače. Příkazový řádek jsme ukončili znakem "CR" - návrat vozu, který se ovšem na obrazovce nezobrazuje písmeny CR, ale návratem kurzoru (ukazovátka) na nový řádek.

V případě, že soubor není systémový binární soubor, pak se na obrazovce zobrazí znak otazník "?" a hlášení monitoru "Ondra V.5" a na novém řádku znak ".", který signalizuje režim monitoru. Kazetopáskový systém Mikos vypisuje na obrazovce hlavičku (tj. název) přečteného souboru.

Z uvedeného postupu je vidět, že zavedení překladače Basicu je velmi jednoduché a stačí zadat pouze jedno písmeno "L" z klávesnice mikropočítače.



Obr. 1 Klávesnice ONDRA

3.1.2. Normální zavedení Basicu

Tento způsob je delší než zkrácený (viz bod 3.1.1.) a systém se dotazuje na jméno zaváděného souboru. Program se zavede do paměti počítače, ale automaticky se nespustí. Pro spuštění zavedeného programu musíme použít příkaz monitoru. Tento způsob umožňuje vyhledávat soubory na kazetě podle zadaného jména souboru nebo dotazovacím způsobem (při hvězdičkové konvenci). Při zadání jména začne systém vyhledávat na kazetě soubor se zadaným jménem, pokud soubor nalezne, zavede ho do paměti. V případě hvězdičkové konvence (tj. místo jména souboru se napíše znak "*" hvězdička), pak se systém po nalezení nové hlavičky souboru dotáže na obrazovce displeje, zda-li je to požadovaný soubor. Na dotaz se odpovídá znakem "Y" (Yes - ano), že požadujeme daný soubor nebo libovolným znakem v záporném významu, že soubor nepožadujeme. Při vyhledávání souboru (zadaného jménem nebo hvězdičkou) můžeme používat rychloposuvů a popřípadě můžeme vyhledávání zastavit a hlavičku souboru zobrazit na obrazovce. Zastavení se provede přidržením tlačítka šipka nahoru po stisknutí klávesy "CR" nebo "SP" (mezerník) se pokračuje ve vyhledávání.

Posloupnost příkazů:

:K=L ... příkaz pro zavedení binárního souboru do paměti počítače

nazev ... ohlášení systému, že požaduje zadat jméno souboru

: jméno ⟨CR⟩ ... dvojtečka signalizuje připravenost počítače pro zadání jména souboru z klávesnice počítače. Můžeme zadat jméno souboru nebo hvězdičku. Po zadání jména se systém dotáže na připravenost magnetofonu.

cteni - hotovo? ... po nastavení magnetofonu na čtení

⟨CR⟩ stiskneme libovolnou klávesu a systém začne číst data z kazety. Je-li zadáno jméno souboru, pak systém po nalezení požadovaného souboru přečte a uloží soubor do paměti.

... znak tečka signalizuje ukončení operace - čtení souboru z kazety a předání řízení zpět monitoru.

Při hvězdičkové konvenci systém vypisuje hlavičky nalezených souborů na kazetě a dotazuje se, zdali je to požadovaný soubor. V každém případě musíme stiskem klávesy počítači odpovědět, aby mohl pokračovat v činnosti.

Např.: Máme na pásce dvakrát program Basic a chceme zavést druhý. Na obrazovce se vypíše:

BASIC PRAC --(Y/N)? N ... první program Basicu, který se jmenuje BASIC a PRAC je poznámka. Na dotaz odpovídáme stiskem klávesy "N" (nebo libovolné jiné kromě "Y").

BASIC KOPIE .. Y/N? Y ... druhý soubor s programem BASIC, který se jmenuje stejně a KOPIE je poznámka. Je to požadovaný soubor, a proto odpovíme znakem "Y". Systém přečte a uloží binární soubor do paměti počítače.
Spuštění interpretu jazyka Basic se provede příkazem monitoru.

Poznámka:

Jméno souboru v Mikosu může obsahovat až 11 znaků. Doporučujeme pro jméno používat 7 znaků a dále pak příponu oddělenou od jména znakem "." (tečka). Pro binární soubory používejte příponu .BIN, pro znakové soubory s programem v jazyce Basic příponu .BAS atd. Používání přípon Vám usnadní orientaci při zavádění programů a dat do paměti počítače. Podle příponky okamžitě poznáte, jaký máte použít povel pro zavedení souboru.

Oddělovačem jména je znak mezera nebo maximální počet znaků pro jméno. Při zadávání jména z klávesnice počítače se pro porovnávání se jménem souboru bere počet zadaných znaků z klávesnice.

Například: Na kazetě máme dva soubory označené jmény TEST1.BIN a TEST2.BAS a soubor TESLA.BIN. Při zadání jména souboru z klávesnice TEST se vybere jeden ze souborů TEST1 nebo TEST2 podle toho, který je dříve na kazetě nalezen (podle okamžitého nastavení pásky v kazetě). Při zadání jména TEST se může vybrat libovolný soubor, ať již TEST1 nebo 2, tak i soubor pojmenovaný TESLA, protože se ze jména nalezeného souboru porovnávají první tři znaky TES, které jsou obsaženy ve všech jménech souborů.

Vzhledem k tomu, že příkazem K-L můžeme zavádět pouze binární soubory, pak při výběru souboru TEST2.BAS vznikne chyba a soubor se do paměti počítače nezavede.

Bližší informace o zavádění programů do paměti a chybových hlášeních a možnostech oprav jsou uvedeny v příručce "Návod k obsluze" mikropočítače ONDRA.

3.2. Spuštění interpretu jazyka Basic

Po zavedení programu Basic do paměti mikropočítače ONDRA příkazem K-L je možné spustit interpret. Použijeme-li příkaz "L", pak se předání řízení provádí automaticky a program Basicu se spustí. Uvedený postup můžeme použít i v ostatních případech, když se z Basicu vrátíme do monitoru a znovu potřebujeme spustit Basic. Monitor má dva způsoby pro předání řízení uživatelským a systémovým programům.

3.2.1. Spuštění systémových programů

Systémové programy (např. interpret Basicu, TOOL atd.) mají na počátku instrukci skoku pro studený start programu (tj. OC3H) a teplý start je o 3 Byte posunut. Pro předání řízení systémovému programu použijeme příkaz monitoru "B". Při prvním použití příkazu po nahrání programu se provede "studený start". Při tomto startu se provede počáteční inicializace interpretu, vynulují se vnitřní proměnné a ukazatelé. Byl-li v paměti počítače nějaký program v Basicu, je beznadějně ztracen.

Při dalších použitích příkazu "B" se již provádí "teplý start", kdy všechny proměnné a programy zůstávají zachovány.

Program Basicu (interpret) je ukládán do paměti mikropočítače od adresy 1000H. Na této adrese je studený start a na adrese 1003H je teplý start.

Programem pro použití příkazu "B" je, že systémový program musí být zaveden v operační paměti počítače.

Posloupnost příkazů:

.B	... povel monitoru pro předání řízení interpretu jazyka Basic
BASIC EXP V5.0/G	... hlášení interpretu jazyka Basic,
(C) 1986 TESLA DIZ	že je připraven přijímat povely (viz
READY	příkaz "L").
:	

Poznámka:

Při skončení práce v Basicu a návrat do monitoru, např. povelem BYE nebo CTRL W se při opětovném spuštění příkazem "B" provádí teplý start. Totéž platí i pro stisknutí tlačítka "NMI". Pokud ovšem v monitoru použijeme příkaz "C" ... studený start systému, pak příkaz "B" nelze

použít.

3.2.2. Spuštění uživatelských a systémových binárních programů

Monitor umožňuje předání řízení na libovolnou adresu v paměti. Této možnosti můžeme využít při spuštění uživatelských programů v assembleru nebo i pro spuštění systémových programů, např. Basicu. Pro předání řízení potřebujeme znát adresu uložení programu v paměti mikropočítače ONDRA. Adresu zavedeného programu z kazety můžeme zjistit příkazem Mikosu K-T (resp. T), který nám zobrazí počáteční a koncovou adresu zavedeného programu.

Známe-li startovací adresu programu, můžeme použít povel monitoru pro předání řízení programu.

Posloupnost příkazů:

.G0000-1000 ⟨SP⟩ ... příkaz monitoru pro spuštění programu od adresy. Počítač po zadání příkazu "G" vypíše posledně použitou (nastavenou) adresu, pokud nám vyhovuje, potvrdíme stisknutím klávesy "SP" (mezerník), po-

kud bychom chtěli teplý start, pak napíšeme adresu 1003. Nová adresa se ukončí znakem "SP".

Monitor předá řízení (tj. provede skok) na zadanou adresu. Je-li zaveden v paměti počítače Basic, pak se ohlásí (viz příkaz "L" a "B").

Poznámka:

Příkaz "G" můžeme použít pro teplý start Basicu i v případě, že jsme nechtěně v režimu monitor zadali příkaz "C" a potřebujeme uchovat rozpracovaný program v jazyce Basic uložený v paměti počítače.

V tom případě však doporučujeme uložit rozpracovaný soubor na kazetu a tak ho zachránit. Pak provedeme nový start Basicu a zpět zavedeme rozpracovaný soubor z kazety. Pokud bychom pokračovali v práci po příkazu "C", mohlo by dojít při nesprávné obsluze ke ztrátě rozpracovaného programu. Příkaz "C" zruší ukazatele zavedeného programu a systém nemá informaci o uložení programu, proto nelze použít příkaz "B".

3.2.3. Spuštění programu zavedeného z kazety

Operační systém Mikos umožňuje předání řízení (spuštění) programu zavedeného z kazety. Při zavádění programu si Mikos uchovává počáteční a koncovou adresu zavedeného programu. Příkaz K-T (resp.T) nám zobrazí tyto údaje. Pro spuštění programu je v Mikosu zabudován příkaz K-J nebo jeho zkrácená forma J. Tento příkaz umožňuje u systémových programů pouze studený start programu, protože provádí skok na zaváděcí adresu a to je u systémových programů studený start.

Posloupnost příkazů:

.K-J ... příkaz Mikosu pro spuštění programu
nebo zkrácená forma : .J

1000 3FFF ... hlášení Mikosu o uložení programu
(počáteční a koncová adresa programu)

3.3. Zavedení a spuštění programu v Basicu

Interpret jazyka Basic EXP/G V5.0 umožňuje zavádění programu do paměti z kazety. Pracuje s dvěma tvary vstupních souborů. První tvar je znakový a na kazetě jsou uváděny příkazové řádky tak, jak je zapíšeme z klávesnice. Tento tvar je nezávislý na modifikacích interpretu a je přenositelný mezi mikropočítačem ONDRA a SAPI 1. Lze ho zobrazovat popřípadě upravovat i programem PEDIT ze souboru TOOL.

Druhý tvar je binární soubor a obsahuje vnitřní předzpracované zobrazení příkazových řádků. Je to vlastně výpis potřebné paměti RAM, kterou obsahuje program a jeho proměnné. Tento tvar je závislý na mikropočítači a verzi interpretu, proto binární tvar nelze přenášet z mikropočítače ONDRA na SAPI a naopak. Zvláštním typem binárního souboru je tzv. "soukromý" soubor, který lze zavést do paměti a spustit, ale nelze ho vypsat nebo v něm provádět úpravy. Jinak se s takovým typem souboru pracuje stejně jako s binárním souborem.

3.3.1. Zavedení znakového souboru

Zavedení znakového souboru vyžaduje kazetový magnetofon s dálkovým ovládáním spouštění a zastavování motorku magnetofonu. Blok z kazety se uloží do vyrovnávací paměti a interpret znakové příkazové řádky převádí do vnitřního tvaru, který ukládá do své pracovní oblasti operační paměti. Převádění je časově náročné a nestihne se provést v mezibloko-vých mezerách. Proto se po každém přečteném bloku pohyb pásky zastaví a interpret provádí transformaci. Po zpracování bloku automaticky přečte další blok. Tak pracuje až do konce souboru.

Posloupnost příkazů:

READY ... hlášení Basicu o připravenosti přijímat povely

:LOAD ⟨CR⟩ ... povel pro zavedení programu (znakového souboru) do paměti

nazev ... požadavek Mikosu na zadání jména souboru

:TEST.BAS ⟨CR⟩ ... zadání jména souboru z klávesnice (znakový soubor se jmenuje "TEST.BAS". Zadání jména se ukončí znakem "CR".

čtení hotovo? ... požadavek na obsluhu, aby se nastavil

⟨CR⟩ magnetofon na čtení. Po nastavení stiskneme libovolnou klávesu (např. CR). Systém začne vyhledávat zadaný soubor. Místo jména můžeme použít i znak "*" hvězdička a systém po nalezení hlavičky souboru se dotáže, zdali nalezený soubor požadujeme. Po nalezení se soubor přečte a uloží se do paměti počítače.

READY ... oznámení, že Basic ukončil čtení programu a je připraven přijímat další povely. Program můžeme spustit příkazem RUN.

:RUN ... spuštění programu

Příkaz LOAD umožňuje zadání jména souboru. V tomto případě pak odpadají dotazy na jméno souboru. Jméno v příkazu LOAD se uvádí v uvozovkách.

Posloupnost příkazů:

:LOAD "TEST.BAS" ⟨CR⟩ ... povel pro zavedení znakového souboru, který se jmenuje TEST.BAS

čtení hotovo? ... požadavek na nastavení magnetofonu na čtení dat

⟨CR⟩ ... potvrzení, že magnetofon je připraven

READY ... soubor byl uložen do paměti

3.3.2. Zavedení binárního souboru

Basic EXP/G V5.0 umožňuje ukládání a zavádění předzpracovaných programů na kazetu nebo z kazety do paměti. Tato forma je pro mikropočítač ONDRA vhodnější, protože se ukládá na kazetu v binární formě a proto vlastní zavádění nebo ukládání na pásku je podstatně rychlejší než u znakových souborů. Proto uživatelům doporučujeme používat převážně tento tvar pro programy v Basicu. Musíme připomenout, že tento tvar není přenositelný mezi různými mikropočítači nebo verzemi základního programového vybavení.

Binární soubory se po zavedení automaticky spustí (tj. provede se povel RUN).

Posloupnost příkazů:

READY ... hlášení Basicu o připravenosti přijímat povely

:LOAD B "TEST.BIN" ⟨CR⟩ ... příkaz pro zavedení binárního souboru s názvem TEST.BIN Příkazový řádek se ukončí znakem "CR". Příkaz LOAD B je pro zavede-

ní binárního souboru. Lze použít i samostatný příkaz LOAD B, pak se systém dotazuje na jméno. Místo jména lze použít hvězdičkovou konvenci.

čtení - hotovo? ⟨CR⟩... požadavek na nastavení magnetofonu na čtení. Provedení ohlásíme stisknutím libovolné klávesy (např. "CR"). Provede se zavedení a spuštění uživatelského programu v Basicu.

3.4. Uložení programu na kazetu

Interpret jazyka Basic se ukládá na kazetu příkazem operačního systému K-S. V paměti je uložen od adresy 1000H až po adresu 3FFFH. Podrobnější informace najdete v Návodu k obsluze. V této části Vás ve stručnosti seznámíme, jak si můžete uložit program v Basicu na kazetu, který jste si zapsali z klávesnice nebo zavedli z kazety a chcete si pořídit kopii souboru. Podle toho, jaký soubor budete potřebovat, tj. znakový nebo binární, budete volit příkazy pro zápis souboru. Pro vlastní potřebu doporučujeme používat binární soubory, které jsou podstatně rychlejší a práce s nimi je jednodušší než se znakovými soubory. Ty používáme, chce-

me-li mít zdrojovou verzi programu, která není závislá na typu mikropočítače nebo verzi základního programového vybavení. Např. chceme-li převést program z mikropočítače ONDRA na systém SAPI 1, musíme použít znakový soubor.

Pokud chceme předat někomu náš program, ale nechceme, aby ho mohl rozkódovat popřípadě měnit, použijeme zápis soukromého souboru.

3.4.1. Zápis znakového souboru

Znakový soubor se na kazetu ukládá s dlouhými mezerami, aby se kazeta mohla zastavit. Tento typ souboru vyžaduje magnetofon s dálkovým ovládáním.

Posloupnost příkazů:

:SAVE	... povel pro uložení znakového souboru. Tento příkaz může mít uvedeno jméno souboru v uvozovkách a pokud máme nastaven magnetofon na správném místě, tj. za posledním souborem, můžeme vyvolat funkci připojení (append) tak, že před jménem uvedeme znak

"@". V tomto případě se systém nedotazuje na jméno souboru, za který má uložit nový soubor.

posledni nazev ... dotaz na jméno souboru, za který se má nový soubor uložit na pásce.

:jméno souboru ⟨CR⟩ uvedeme jméno souboru ukončené znakem "CR" . Lze použít i hvězdičkovou konvenci.

cteni - hotovo?⟨CR⟩ nastavíme magnetofon na čtení a potvrdíme libovolným znakem, např. ⟨CR⟩ . Systém hledá soubor podle jména a po nalezení se nastaví za soubor.

Zápis! hotovo?⟨CR⟩ požadavek na nastavení magnetofonu na zápis, po provedení potvrdíme stiskem libovolné klávesy (např. CR)

nový nazev ... požadavek na uvedení jména souboru, pod kterým se uloží znakový soubor

: jméno nového souboru ⟨CR⟩ zapíšeme jméno ukončené znakem CR. Systém zapíše soubor na pásku.

READY ... ukončení zápisu a připravenost přijímat povely.

3.4.2. Zápis binárního souboru

V příkladu si uvedeme "append", tj. připojení souboru, jestliže máme pásku správně nastavenou za posledním souborem (např. prováděli jsme zápis).

Posloupnost příkazů:

:SAVEB"@TEST.BIN" ⟨CR⟩ ... chceme uložit na pásku binární soubor pod jménem TEST.BIN. Znak "@" není součástí ukládaného jména a systému signalizuje, že požadujeme funkci připojení. Příkaz SAVE B označuje, že se bude ukládat binární soubor.

zapis!hotovo? ⟨CR⟩ ... požadavek na nastavení magnetofonu na zápis. Po nastavení potvrdíme stiskem klávesy. Systém provede uložení binárního souboru na kazetu.

READY ... ukončení zápisu a připravenost Basicu přijímat povely.

3.4.3. Zápis soukromého souboru

Tento typ souboru používáme pro programy, které se mohou spustit a provádět, ale nelze je vypisovat a upravovat. Proto pozor při používání tohoto typu souboru, aby jste měli verzi, která není utajená a mohli jste si dělat změny a úpravy ve zdrojové verzi programu. Soubory s utajenými programy předáváme tam, kde se programy mohou provozovat, ale není žádoucí, aby se prováděly úpravy.

Posloupnost příkazů:

:SAVE P"TEST.SKR"⟨CR⟩ ... příkaz pro uložení "soukromého" (utajeného) programu na kazetu pod jménem TEST.SKR. Příkaz ukončíme znakem CR.

čtení - hotovo? ⟨CR⟩ ... požadavek na nastavení magnetofonu na čtení. Potvrzení stiskem klávesy.

jméno 1 --(Y/N)? N ... nalezení souboru se jménem 1. Nepožadujeme uložení nového souboru za nalezený soubor, po stisknutí klávesy N pokračuje systém v hledání dalšího souboru.

jméno 2 --(Y/N)? Y .. nalezení požadovaného souboru, za který chceme uložit nový soubor a proto potvrdíme znakem Y. Soubor se dočte do konce.

zapis!hotovo? (CR) ... požadavek na nastavení magnetofonu na zápis, provedení potvrdíme stiskem klávesy. Systém provede zápis souboru.

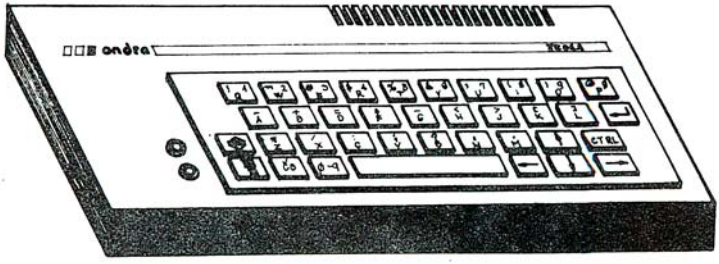
READY ... ukončení zápisu

V povelech jazyka Basic si probereme další varianty příkazů LOAD a SAVE, které nám umožní ukládat na pásku obsah videopaměti RAM a části programů nebo znakových řetězců. Dále se seznámíte s možností spojování a překrývání programů. Při zadání příkazu LOAD a je-li v paměti mikropočítače již uložen program, se systém zeptá, zda chcete programy spojit. V kladném případě vám stejné řádky programu přepíše novými řádky ze zaváděné části. K programu v paměti můžeme připojovat pouze znakové soubory (nikoliv binární nebo soukromé soubory).

4 ZÁKLADNÍ POJMY

V této kapitole si ve stručnosti vysvětlíme některé pojmy používané v Basicu tak, abyste snáze pochopili, co jednotlivé příkazy a povely provádí. Neklademe si za cíl seznámit Vás s přesnými definicemi jednotlivých pojmů, ale jednoduchou formou vám vysvětlit základní princip. Tuto kapitolu můžete vynechat, pokud znáte základní pojmy jako počítač, paměť, vstup-výstupní zařízení, bit, Byte atd.

4.1. Počítač



Popis a princip práce počítače je uveden v příručce "Návod k obsluze mikropočítače ONDRA". Počítač je souhrn technických a programových prostředků vzájemně propojených tak, aby plnily požadované funkce, tj. správné provádění instrukcí programů.

Počítač můžeme rozdělit na dvě základní skupiny: technické vybavení (hardware) a programové vybavení (software).

Technické vybavení lze rozdělit na čtyři základní díly: procesor, paměť, vstupy a výstupy.

a) Procesor řídí chod celého počítače a provádí instrukce programu. Pracuje s videopamětí (vybírá a ukládá data) a ovládá vstup-výstupní zařízení. Procesor je řízen časovými pulzy - hodinami. Součástí procesoru je aritmeticko-logická jednotka, která provádí aritmetické operace (sečítání, odečítání) a logické operace (srovnání, práce s bity atd.). Řídicí jednotka vybírá a provádí instrukce.

b) Paměť slouží k ukládání a vybírání dat a instrukcí.

Podle způsobu práce s pamětí rozdělujeme paměti na pevné (ROM - read only memory, EPROM), ze kterých můžeme pouze číst, ale nemůžeme do nich zapisovat a paměti typu RAM (random access memory), které umožňují jak zápis, tak i čtení dat. Této paměti se také říká operační paměť, protože si do ní procesor ukládá instrukce programu a zpracovávaná data.

Konstrukčně se paměti mohou značně lišit, mohou pracovat na magnetickém principu (např. feritové paměti) nebo na principu elektrických obvodů (klopné obvody). Tyto paměti mají i různé vlastnosti, např. magnetické obvody si pamatují i po vypnutí napájení, kdežto paměti na elektrickém principu po vypnutí napájení informaci ztratí. V mikropočítači ONDRA jsou použity integrované paměti na elektrickém principu. Zde je důležité si zapamatovat, že informace v operační paměti se po vypnutí počítače ztratí.

Paměti typu ROM (PROM, EPROM) umožňují čtení dat, ale neumožňují zápis. Obsah této paměti se určuje obvykle již při návrhu daného počítače a programují se mimo vlastní počítač na speciálním zařízení.

V mikropočítači ONDRA je použita paměť EPROM. Můžete mít oprávněnou otázku, proč je použita tato

paměť, když je v počítači paměť RAM, tak nač ještě paměť EPROM, když ji nepotřebujete?

Vysvětlení je velmi jednoduché, řekli jsme si, že paměť RAM se po vypnutí vymaže a při spuštění počítače potřebujeme mít v paměti "nějaký" program, který se začne provádět a takový program je právě uložen v paměti EPROM. Je to program od výrobce mikropočítače (firmware), který umožňuje základní nastavení a komunikaci s počítačem. Podle typu a zaměření počítače jsou tyto programy jednoduché (programy typu Monitor) až po velmi rozsáhlé a složité programy - operační systémy.

U mikropočítače ONDRA je jednoduchý operační systém typu Monitor a pro práci s kazetovým magnetofonem Mikos (malý kazetopáskový operační systém). Monitor vám umožní pracovat s klávesnicí, zobrazovat na obrazovce displeje atd.

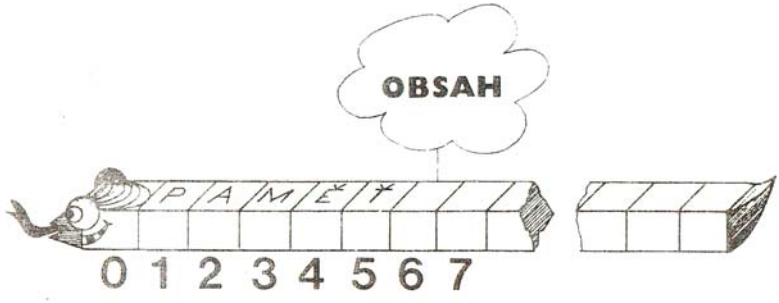
Monitor obsahuje i další užitečné podprogramy, které jsou uživatelům přístupné přes služby Monitoru. Bez operačního systému by počítač byl pouze krabičkou, která nic neumí a která se nedá ani použít. To proto, že po zapnutí by počítač začal provádět náhodný obsah paměti jako instrukce které nedávají žádný rozumný smysl. Je třeba si

uvědomit, že i pro zadání znaku z klávesnice je v paměti počítače uložen program, který musí zjistit, které tlačítko na klávesnici bylo stisknuto a tomuto tlačítku přiřadit správný význam.

Nyní se vraťme k paměti a názorně si vysvětlíme, jak paměť pracuje. Paměť je složena z paměťových míst. Představme si, že jedno paměťové místo je krabička, do které lze uložit jeden lístek, na kterém je napsáno např. písmeno (nebo číslice). Pak vložení lístečku do krabičky je zápis do paměti.Pokud jsme tam již lístek měli, tak starý vyhodíme a vložíme nový. Čtení z paměti je vytažení lístečku a jeho přečtení z krabičky a lístek znovu vrátíme do krabičky pro další použití.

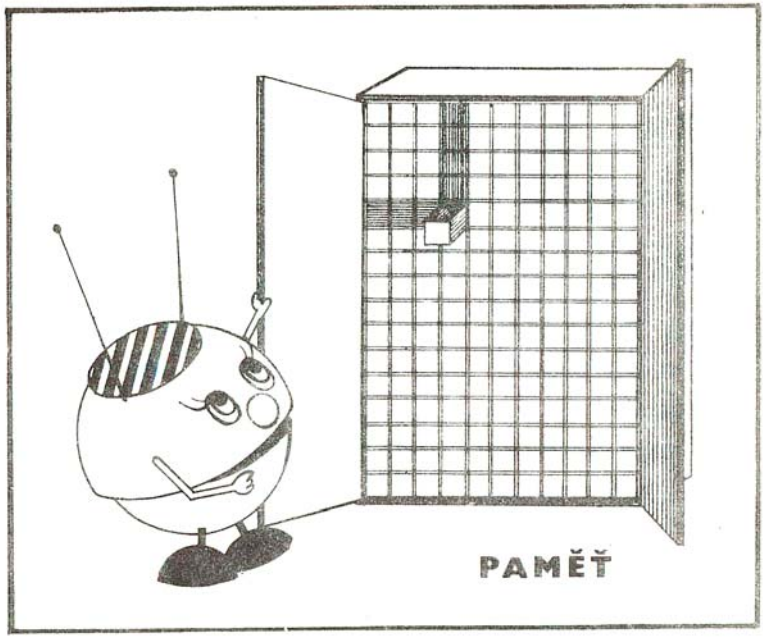
Počítač s jednou krabičkou nevystačí a takových krabiček potřebuje hodně. Tomu, kolik krabiček má počítač, se říká velikost paměti. U mikropočítačů budeme krabičce říkat "BYTE" (čti bajt) a velikost paměti se udává v tisících Byte "KB" (kilobyte). Mikropočítač ONDRA má operační paměť 64 KB. Aby počítač věděl, s kterou krabičkou má pracovat, musí si krabičky označit (například čísly). Tomuto označení pak říkáme adresa paměti. Paměť si můžeme představit jako jednoho dlouhého "hada" složeného

z krabiček a jednotlivé krabičky jsou očíslovány od nuly až do nejvyššího čísla (u Ondry 65 535).



Jinou představou může být skříňka, ve které jsou uloženy jednotlivé krabičky, které jsou uspořádány po řádcích a sloupcích. Pak, abyste nalezli správnou krabičku, stačí uvést číslo řádku a sloupce.

Pro paměť EPROM si uděláme jinou představu. Představme si, že paměťové místo je razítko s pevným obsahem. Když potřebujeme vědět, co je na daném místě, pak vezmeme příslušné razítko a provedeme si otisk razítka, které vrátíme na svoje místo.



c) Vstupy a výstupy

Abychom mohli počítači zadávat, co má provádět a on nám mohl předat výsledky, potřebujeme zařízení pro vstup do počítače a výstup z počítače. Některá zařízení mohou být pouze vstupní, např. klávesnice, jiná pouze výstupní, např. obrazovka televizního displeje a některá zařízení mohou být jak vstupní, tak i výstupní, např.

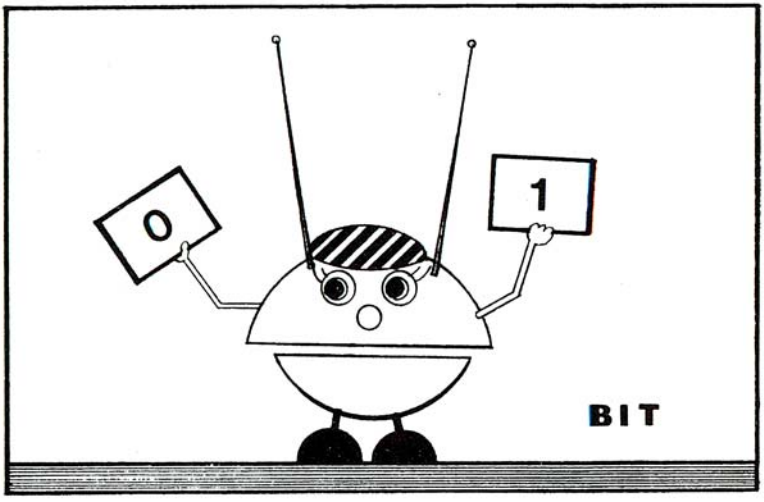
kazetový magnetofon. Bez těchto zařízení by počítač nebyl počítačem, protože bychom mu nemohli nic zadávat, ani bychom se nedozvěděli, co vlastně vypočítal. Proto vstupní a výstupní zařízení tvoří nedílnou součást počítače. Mikropočítač ONDRA má jako vstupní zařízení klávesnici, křížový ovladač (pro řízení ukazovátka na obrazovce) a kazetový magnetofon pro vstup programů a dat. Výstupními zařízeními Ondry jsou televizní displej, tiskárna a kazetový magnetofon (pro uložení programů a dat). Základním vybavením jsou klávesnice, televizní displej a magnetofon, jako doplňující jsou křížový ovladač a tiskárna. Připojení dalších zařízení se u Ondry nepředpokládá.

4.2. Struktura dat

Když jsme si vysvětlovali paměť, použili jsme pojem Byte a v této části si blíže vysvětlíme strukturu dat.

Základní jednotkou informace je "bit" (pozor, nezaměňujte se slovem Byte - bajt). Bit má dvě hodnoty "0" a "1" (někdy se používá "ano", "ne" nebo "pravda, nepravda" - "true, false" a nebo označení písmeny "H"

(high - vysoká), "L" (low - nízká). Tato jednotka se také technicky nejsnáze realizuje.



Protože bit je malá jednotka, sdružují se bity do větších skupin podle určitých pravidel. Skupina osmi bitů tvoří další základní jednotku "Byte", která bývá nejmenší samostatně adresovatelnou jednotkou paměti. Uspořádání bitů v Byte je podle pravidel dvojkového čísla. My jsme zvyklí na desítkovou soustavu. Číslo zapisujeme zprava doleva a na pravé straně jsou nejnižší řády. Kaž-

dý další řád představuje násobek desítek, stovek, tisiců atd. Obecný zápis čísla:

$a_m \cdot\cdot a_2 a_1 a_0 = a_m * 10^n + \cdot\cdot + a_2 * 10^2 + a_1 * 10^2 + a_1 * 10^1 + a_0 * 10^0$,

kde n je řád čísla a a_i jsou číslice 0 - 9.

Tak např. číslo

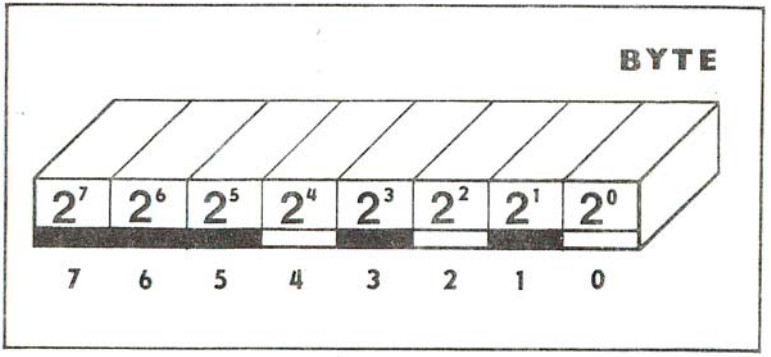
$234 = 2 * 10^2 + 3 * 10^1 + 4 * 10^0 = 2 * 100 + 3 * 10 + 4$

Podobně je tomu i u dvojkového čísla, které má místo mocniny deseti mocninu čísla 2.

Obecný zápis dvojkového čísla:

$a_m \cdots a_2 a_1 a_0 = a_m * 2^m + \cdots + a_2 * 2^2 + a_1 * 2^1 + a_0 * 2^0$,

kde m je řád čísla a a_i jsou číslice "0" nebo "1".



Tak například desítkové číslo 234 zapíšeme ve dvojkové soustavě jako 1110 1010.

$1110\ 1010 = 1*2^7 + 1*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + +\cdot 1*2^1 + 0*2^0 = 1*128 + 1*64 + 1*32 + 0*16 + 1*8 + + 0*4 + 1*2 + 0*1 = 128 + 64 + 32 + 8 + 2 = 234.$

Jsou-li v Bytu samé jedničky (1111 1111) dostaneme dekadické číslo 255. To je ovšem pro běžné počítání malé číslo, a proto slučujeme více Byte. Je-li číslo tvořeno dvěma Byte, pak první Byte tvoří řády od 0 do 7 a druhý Byte od 8 do 15. Šestnáct pozic dvojkového čísla umožní zapsat dekadické číslo 65 535.

Vidíte, že dvojkové číslo zabírá příliš mnoho řádů a pro psaní programů je nepraktické. Proto se ve výpočetní technice používají i jiné soustavy, které jsou odvozeny od dvojkové, např. osmičková nebo šestnáctková soustava. U minipočítačů řady SMEP se používá osmičková soustava a u mikropočítačů se používá šestnáctková soustava.
U osmičkové soustavy se používají číslice 0 až 7. Převod dvojkového čísla na osmičkové je velmi jednoduchý. Dvojkové číslo rozdělíme zprava po trojici bitů a hodnotu každé trojice zapíšeme číslicí. Největší hodnota, kterou může trojice bitů nabývat je číslice 7.

Příklad: 1110 1010 = 011 101 010 = $352_{(8)}$ = $234_{(10)}$
3 5 2

$352 = 3 * 8^2 + 5 * 8^1 + 2 * 8^0 = 3 * 64 + 5 * 8 + 2 =$

$= 192 + 40 + 2 = 234$

U mikropočítače ONDRA se používá šestnáctková soustava, zejména při programování v assembleru, ale i v Basicu se může použít čísla v šestnáctkové soustavě. Zde již nevystačíme s číslicemi 0 až 9 a proto pro označení dalších šestnáctkových číslic se používají písmena A až F. Číslice 0 - 9 mají stejný význam jako v desítkové soustavě a pro další číslice se používají písmena:

des.	hex.	bin	des.	hex.	bin
10	A	1010	13	D	1101
11	B	1011	14	E	1110
12	C	1100	15	F	1111

Převod dvojkového čísla na šestnáctkové je rovněž jednoduchý, dvojkové číslo si rozdělíme po čtveřicích bitů zprava. Každou čtveřici vyjádříme šestnáctkovou číslicí.

Příklad: $\underbrace{1110}_{E}\ \underbrace{1010}_{A} = EA$

Aby se odlišila dekadická čísla od hexadecimálních, připojuje se za hexačíslo písmeno "H".

$EAH = E * 16^1 + A * 16^0 = \underbrace{(14)}_{E} * 16 + \underbrace{(10)}_{A} * 1 = 224 + 10 = 234_{10}$

Vidíte, že hexadecimální zápis je velmi úsporný a jeden Byte můžeme zapsat dvěmi hexadecimálními číslicemi. Snadno se převádí na dvojkové číslo a naopak dvojkové na hexadecimální.

Znaky

Zatím jsme se seznámili s organizací paměti a pojmem Byte. Řekli jsme si, že počítač může zpracovávat pouze dvojkové čísla a proto pokud chceme zpracovávat alfanumerické znaky, pak je musíme převést na dvojková čísla. Tyto znaky jsou normalizovány a mikropočítač ONDRA pracuje podle československé normy KOI 8/čs-2. Mezinárodní abeceda má 26 písmen a 10 číslic a další znaky, dále se rozlišují velká a malá písmena a znaky české abecedy (tj. písmena s háčky, čárkami). Dále norma obsahuje řídící znaky pro formátování tisku a řízení přenosu dat. Každý znak má přesně určenou binární kombinaci. V příloze je uvedena tabulka znaků.
Například velké písmeno "A" má bitovou kombinaci 0100 0001, tomu odpovídá hexadecimální kombinace 41H a dekadická 65.

Jeden znak je uložen na jednom Byte. Pokud potřebujeme uložit více znaků, pak se ukládají do posloupnosti paměťových míst.

V Basicu říkáme znakovým proměnným a konstantám řetězce (string). Aby Basic poznal, s jakou proměnnou má pracovat, označují se řetězcové proměnné znakem měny "$" (dolar) nebo "%". Řetězcové konstanty se píší do uvozovek.

Čísla

Uvedli jsme si zobrazování dvojkových čísel v paměti počítače. Toto zobrazení se převážně používá pro zadávání adres. Pro aritmetické výpočty potřebujeme čísla se znaménkem (rozlišit kladná a záporná čísla). Důležitý je i rozsah zobrazovaných čísel. U Ondry jsou použity dva formáty čísel: čísla celočíselná (integer) a čísla v pohyblivé řádové čárce (float point). Mikroprocesor umí zpracovat pouze celočíselná čísla, čísla v pohyblivé řádové čárce jsou zpracovávána programově.

Čísla celočíselná

Čísla celočíselná jsou uložena na dvou bytech. Poslední horní bit (tj. 15. bit) je znaménkový bit. Je-li "0", pak je číslo kladné, je-li "1", pak je číslo záporné. Záporná čísla jsou zobrazována v tzv. dvojkovém doplňku. Kladné dvojkové číslo se převede na záporné tak, že se provede jedničkový doplněk (negace - záměna nul za jedničky a

jedniček za nuly) a přičte se 1.

Příklad: kladné číslo 5 - 0000 0101

```
  1111 1010 .. jedničkový doplněk
+ 0000 0001 .. plus 1
  1111 1011 .. dvojkový doplněk
    F    B      čísla 5 (hexa)
```

Podobně lze převést i ostatní čísla.

0111 1111	7FH	127	max.jednobytové kladné číslo
0111 1110	7EH	126	
•			
•			
0000 0011	03H	3	kladná čísla
0000 0010	02H	2	
0000 0001	01H	1	
0000 0000	00H	0	nula
1111 1111	FFH	-1	
1111 1110	FEH	-2	
1111 1101	FDH	-3	
•			záporná čísla
•			
1000 0010	82H	-126	
1000 0001	81H	-127	
1000 0000	80H	-128	

V příkladu je jednobytové slovo (7. bit je znaménkový).

Basic V5.0 používá převážně čísla v pohyblivé řádové čárce. Celočíselná čísla používá při logických operacích AND, OR a některých dalších operacích. Číslo v pohyblivé řádové čárce se převede na celočíselné, se kterým se pak pracuje.

Čísla v pohyblivé řádce

Čísla v pohyblivé řádové čárce jsou ukládána ve zvláštním tvaru na více Bytech. V Basicu V5.0 se používají čísla jednoduché délky, která jsou uložena na 4 Bytech. Čísla se ukládají v exponencionálním tvaru mantisa a exponent.

Mantisa je číslo menší než 1 (normalizované) a exponent je mocnina, kterou se vynásobí mantisa.

Příklad: číslo 125,45 se převede do exponencionálního tvaru $0.12545 * 10^3$.

Normalizace se provádí tak, aby desetinná tečka byla před první platnou číslicí. Posunem tečky vlevo se řád exponentu zvyšuje, vpravo se snižuje.

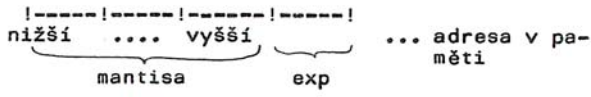
Obecně lze čísla zapsat ve tvaru:

$\pm$ XE-NN nebo $\pm$ XE+NN ,

kde X je číslo v normálním tvaru, znak "E" je znakem exponenciálního tvaru a NN je exponent (deset na NN), který mů-

že být v rozsahu 0 ≦ NN ≦ 38.

Čísla v paměti jsou ukládána ve tvaru:



Na 7. bitu nejvyšší mantisy je uloženo znaménko mantisy. Exponent je uložen v modifikovaném tvaru. Exponent "0" je zobrazen jako 80H. Je-li exponent menší než nula, pak je v rozsahu 0-7FH a je-li větší než nula, pak je v rozsahu 81H až FFH. Čísla mantisy mají tzv. "skrytý bit" a vychází z toho, že normalizované číslo mantisy je v rozsahu 1/2 ⟨ = číslo ⟨ 1 a tudíž nejvyšší bit je vždy roven "!", a proto ho není nutno ukládat do paměti (proto termín skrytý bit). Mantisa na třech bytech umožňuje přesnost 7 1/2 platných dekadických číslic. Osmá číslice je již zaokrouhlená. Exponent může být až ± 38 (desítkově). Zobrazovaná čísla mohou být v rozsahu od 1.70141E-38 až do 1.70141E+38. Čísla v Basicu jsou v paměti ukládána vždy v exponenciálním tvaru.

Basic V5.0 připouští vstup i výstup čísel v "normálním" tvaru, tj. číslo je zapisováno s desetinnou tečkou na libovolném místě (pro vstup) a pro výstup, je-li číslo v rozsahu 0.1 < X < 999999. Pokud je výstupní číslo

mimo uvedený rozsah, pak vystupuje v exponenciálním tvaru.

4.3. Prvky jazyka Basic

V předchozí části jsme se seznámili s tím, jak jsou data ukládána v paměti počítače. V této části se seznámíme s ukládáním dat v Basicu V5.0 a dalšími charakteristickými rysy jazyka.

Proměnné

Často potřebujeme označit a rezervovat místo v paměti pro výsledky aritmetických nebo znakových operací. Mnohdy ani nevíme, jaké hodnoty budou vstupovat do řešení (např. vstup z klávesnice nebo výsledek předchozího řešení). V matematice často používáme obecný zápis rovnice, např. rovnice přímky má tvar:

$$Ax + By + C = 0 \ ,$$

kde A,B,C jsou parametry přímky a x, y jsou souřadnice bodů ležících na přímce. Při řešení této rovnice dosadíme konkrétní hodnoty parametrů přímky a např. pro určitou hodnotu souřadnice y hledáme odpovídající souřadnici x bodu ležícího na přímce.
Vidíme, že v této jednoduché rovnici se mohou měnit prakticky všechny parametry a hledané souřadnice.

V Basicu potřebujeme pro takové hodnoty, které mohou měnit svojí velikost (svůj obsah), vymezit paměťový prostor a tento prostor označit. Označeným místům paměti říkáme proměnné.

Použijeme opět analogii s krabičkami (jak byla vysvětlována paměť). Představme si, že na skupinu krabiček napíšeme jména. Např. čtyři krabičky označíme písmenem A, druhou čtveřici B, další pak písmeny C, X, Y a máme připravené proměnné pro řešení rovnice přímky. Vymezujeme čtveřici krabiček proto, že Basic pracuje s čísly v pohyblivé řádové čárce, které zabírají v paměti 4 Byte. Označení slovním názvem proměnné je vlastně adresa paměťového místa. Basic má ve své pracovní oblasti tabulku, ve které jsou jednotlivým proměnným přiřazeny skutečné adresy. V případě změny umístění proměnné si Basic sám upraví tabulku tak, aby vždy správně ukazovala na umístění proměnných v paměti.

Nyní Vás seznámíme s pravidly pro názvy proměnných. Basic V5.0 připouští jméno proměnných libovolně dlouhé, ale ze jména si bere pouze <u>první dva znaky!</u> Jméno proměnné musí začínat písmenem. Dále jméno proměnné nesmí obsahovat "klíčová slova" nebo jejich části (klíčová slova jsou příkazy nebo povely jazyka Basic, např. GO TO, IF, THEN atd.) a nesmí obsahovat české znaky (háčky, čárky).

Jména proměnných (identifikátory) pro řetězcové proměnné se tvoří podle stejných pravidel jako pro numerické proměnné, ale za jméno se připojuje znak měny (dolar "$" nebo na některých počítačích "↑").

Příklady proměnných: A, A1, AB, ALFA, BETA ... správné numerické proměnné

TON ... je nesprávný, protože obsahuje klíčové slovo TO

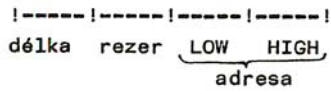
1A ... je nesprávné, protože nezačíná písmenem

AB, ABCEDA označují jednu a tutéž proměnnou AB, protože Basic bere první dva znaky

A$, JMENO$, B3$ ... správné řetězcové proměnné

Jména numerických a řetězcových proměnných se mohou shodovat a přitom označují různé proměnné (dáno znakem měny). Numerické proměnné a pole se ukládají v pracovní oblasti Basicu, která je umístěna za překladačem Basicu. Řetězcové proměnné jsou ukládány od horní dostupné paměti až do dosažení rezervované oblasti příkazem Basicu "CLEAR". Implicitní velikost oblasti pro řetězcové proměnné je 255 Byte.

Řětězcové proměnné se ukládají "nepřímo", tj. je vytvořen 4 Bytový ukazatel ve tvaru:



Vlastní řetězcové proměnné jsou uloženy dynamicky v horní části paměti, ukazatele jsou uloženy v pracovní oblasti. Při práci s proměnnými se jejich umístění v paměti dynamicky mění tak, aby paměťový prostor byl maximálně využit. Operace s řetězcovými proměnnými jsou poměrně pomalé, protože se provádí zhušťování proměnných v paměti počítače.

Konstanty

Jazyk Basic umožňuje uživatelům zapisovat v přiřazovacích příkazech (na pravé straně příkazu) numerické nebo znakové konstanty. Numerické konstanty se zadávají jako desitkové číslo buď jako celočíselná nebo desetinná čísla. Zadání čísel s desetinnou tečkou může být v normálním tvaru (tj. celé číslo se znaménkem, desetinná tečka, celé číslo) nebo v exponenciálním tvaru.

Řetězcové konstanty jsou uzavřeny v uvozovkách.

Příklad: LET A=123.45 ... přiřazení konstanty 123.45 proměnné "A"

LET A$="123.45" ... přiřazení znakové konstanty 123.45 do proměnné A$ (pozor jsou to znaky, nikoliv číslo, se kterým lze provádět aritmetické operace

LET B$="TEXT" ... přiřazení znakové konstanty TEXT do proměnné B$

Pole

Pole je zvláštní typ proměnných, které mají společné jméno a na jednotlivé proměnné se odkazujeme pomocí pole a indexu. Pole mohou být v Basicu V5.0 až osmirozměrná. Velikost pole se vymezí deklarací pomocí příkazu DIM. V příkazu se pro každý rozměr pole uručuje horní hranice. Pokud pole není deklarováno příkazem DIM, pak se při prvním použití prvku pole provede implicitní (tj. samočinná deklarace) vymezení pole s horní hranicí 10 pro každý rozměr. Dolní hranice pole je vždy "nula", tudíž počet prvků jednoho rozměru je horní hranice + 1.

Doporučujeme používat pro deklaraci pole příkazu DIM, protože v paměti vymezíme nezbytně nutné místo a dále je

i provádění programu rychlejší, protože Basic nemusí provádět reorganizaci dat v paměti. Je nutné si uvědomit, že s počtem rozměrů pole se rychle zvyšují nároky na paměť. Pole mohou být numerická nebo řetězcová. Řetězcová pole se ukládají do prostoru pro řetězcové proměnné, tj. od horního konce paměti a je nutno rezervovat dostatečný prostor příkazem CLEAR.

Nyní si uděláme názornou představu jedno a dvourozměrného pole. Jednorozměrné pole si opět můžeme představit jako "hada" složeného z krabiček, do kterých se ukládají proměnné. Blok krabiček je pojmenován (jménem pole) a jednotlivé krabičky jsou očíslovány. První krabička má číslo "0" a poslední má číslo, uvedené v příkazu DIM. Číslo na krabičce je index pole. Pokud chceme něco vložit do krabičky nebo z ní něco vybrat, pak musíme překladači uvést jméno pole. Překladač si najde příslušný blok krabiček a dále v závorce uvedeme číslo (index). Tím Basic ví, kterou krabičku má použít. (V tomto výkladu pod pojmem krabička, chápeme více Byte - pro numerické pole 4 Byte, pro řetězcové nezbytně nutný počet, podle počtu znaků).

U dvojrozměrného pole si pole můžeme představit jako skříňku s poličkami (řádky) a krabičky jsou uspořádá-

ny do sloupců. První index pak ukazuje řádek a druhý sloupec, kde je příslušná krabička uložena. Celá skříňka je pojmenována jménem pole. V paměti se ovšem tato skříňka rozloží opět do "hada" s tím, že se za sebou postupně ukládají jednotlivé řádky.

Příklad: DIM A(1,2)
se v paměti uloží následovně:
A(0,0) A(0,1) A(0,2) A(1,0) A(1,1) A(1,2) A(2,0) A(2,1) A(2,2)

Komentáře

V Basicu V5.0 je možno používat komentáře - poznámky pro zvýšení přehlednosti programu v Basicu. Komentářem může být celá řádka, pokud je uvedena příkazem REM, nebo část příkazové řádky pokud je víceříkazová řádka a příkaz REM je použit za oddělovačem příkazů ":". Zbytek řádku za příkazem REM je brán jako poznámka a dále se nevyhodnocuje. V Basicu V5.0 je doplněn znak "!" (vykřičník), který rovněž uvozuje poznámku (místo příkazu REM). Další text za vykřičníkem je chápán jako poznámka.

Poznámky se ukládají do paměti počítače. Proto pokud potřebujeme šetřit s pamětí, doporučujeme zrušit poznámky.

Když si vytváříte program, tak je vhodné používat poznámky. Komentovaný program si uložte na kazetu. U odladěných programů nepoužívejte poznámky, protože mají menší nároky na paměť a zpracování je rychlejší. Poznámky nemají význam u utajených programů (uložených příkazem SAVE P), protože je nelze vypisovat příkazem LIST.

Řádek, oddělovače příkazů

Programový nebo dialogový řádek má v Basicu V5.0 délku 132 znaků. Nejsou povoleny pokračovací řádky, a proto musí být příkaz ukončen na řádku. Na jednom řádku může být více příkazů, které jsou odděleny znakem ":" (dvojtečka), který je oddělovačem příkazů.

Do počtu znaků na řádku se započítávají čísla řádků (v programovém režimu) a mezery. Řádek se ukončuje tlačítkem "↵" (na klávesnici mikropočítače Ondra), které generuje dvojici znaků "CR" (návrat vozu) a "LF" (posun o řádek).

Dialogový řádek je bez čísla řádku a příkazy nebo povely se po ukončení řádku okamžitě provedou. Některé příkazy nelze použít v dialogovém řádku.

Programové řádky se zapisují s číslem řádku. Po zapsání programu se program v jazyce Basic spustí povelem RUN. Před spuštěním nebo po ukončení programu (resp. zastavení při chybě) lze program editovat (tj. opravovat a doplňovat program). Basic umožňuje vytváření programů, tj. zápis programových řádků. Programové řádky nemusíme bezpodmínečně psát za sebou, ale můžeme je psát libovolně a překladač si je uspořádá podle vzestupných čísel řádků. Tuto vlastnost oceníme zejména při opravách, když zjistíme, že potřebujeme doplnit řádek programu. Přesto doporučujeme, abyste si program nejdříve řádně promysleli, napsali na papír a teprve potom zapsali program do počítače. Při číslování řádků programu nepište čísla za sebou, ale s nějakým krokem (např. 10) tak, abyste měli možnost v případě potřeby doplnit programové řádky. Basic V5.0 připouští čísla řádků od 1 do 64999.

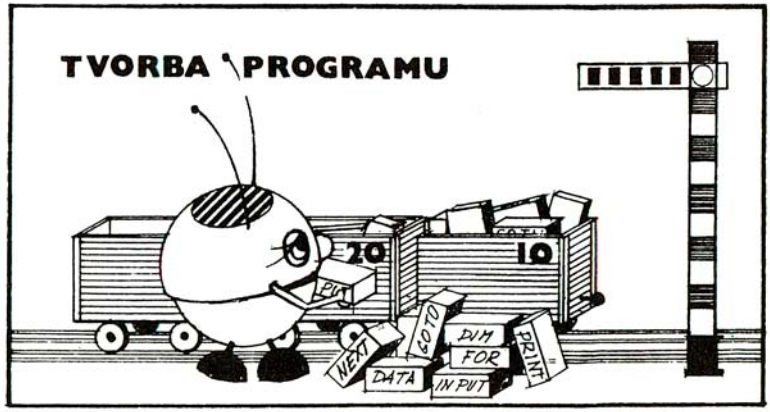
Napíšete-li řádek se stejným číslem řádku, který již máme v programu, pak se starý řádek přepíše novým řádkem a starý je ztracen. Rovněž napíšete-li pouze číslo řádku ukončené znakem "↵" pak jestliže existoval řádek s tímto číslem, se tento řádek zruší a prázdný řádek se do paměti nezavede.

Uvedený postup je vlastně nejjednodušší editací, která Vám umožňuje zrušit řádky a napsat nové. Tato editace je nevýhodná, protože musíte vždy znovu napsat celý řádek (zejména u vícepříkazových řádků). Proto je Basic V5.0 doplněn příkazem "@" (edit), který umožní vymazat, změnit, doplnit jednotlivé znaky do programového řádku, bez nutnosti přepisovat celý řádek. Editor má i další vlastnosti, se kterými Vás seznámíme v popisu příkazů.

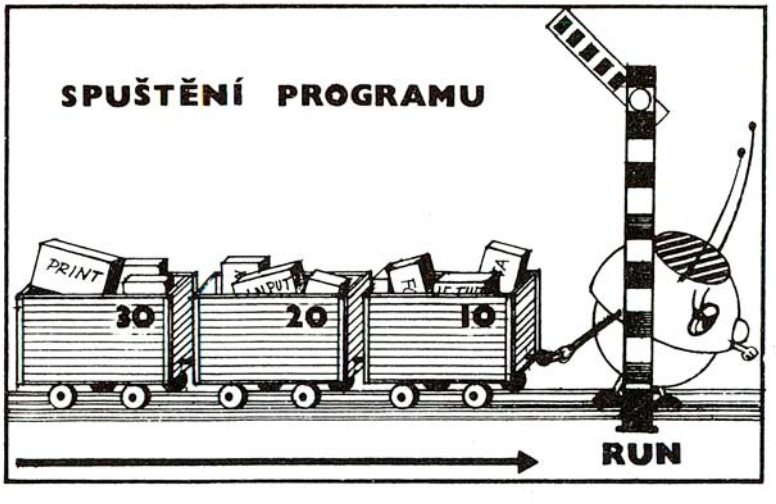
Dialogový režim Basicu nám může udělat z počítače "kalkulačku". Ta např. chceme vypočítat pátou mocninu čísla 2, pak napíšeme příkaz:

PRINT 2^5 ↵

a na obrazovce se nám vypíše výsledek 32.



Vytváření programu si můžeme představit jako naplňování vagonů nákladního vlaku, který stojí na nádraží. Jednotlivé vagony jsou očíslovány (čísla programových řádků) a do vagonu nakládáme jednotlivé příkazy. Semafor máme nastaven na stůj a tudíž můžeme nakládat. Povelem RUN dáme na semafor volno a vlak se rozjede - začne provádět program.



Program se ukončí při provádění příkazů END nebo STOP. Příkazy nastaví semafor na stůj v konečné stanici.

Operátory

Jazyk Basic rozlišuje tři typy operátorů:

- aritmetické
- relační
- logické.

Operátory mohou být pouze na pravé straně přiřazovacího příkazu, nebo za klíčovým slovem v podmíněných příkazech a cyklech.

Aritmetické operátory:

^ ... umocněno

* ... násobeno

/ ... děleno

+ ... plus, sečítání

- ... mínus, odečítání

Aritmetické operátory se používají při zápisu aritmetických výrazů s numerickými proměnnými nebo numerickými konstantami. Aritmetické operátory mají stanovenou prioritu provádění. Nejvyšší prioritu má umocnění, nižší prioritu má násobení a dělení (obě mají stejnou) a nej-

nižší prioritu mají sečítání a odčítání. Při stejné prioritě se výrazy vyhodnocují zleva doprava. Při vyhodnocování výrazu se nejdříve vyhodnotí operace s nejvyšší prioritou, pak se střední a nakonec s nejnižší prioritou. Změnu v pořadí vyhodnocování můžeme provést pomocí kulatých závorek. Nejdříve se vyhodnotí výraz uvedený v závorkách (podle platných priorit) a pak se vyhodnocují další operátory.

```
Příklad:  X = 2 + 3*4 ^ 2 - 4 / 2
              4   2   1   5   3      ... pořadí vyhodnocení
          X =  2 + 3 * 16 - 4 / 2  ... 1 priorita 1 krok
          X =  2 +   48   -  2     ... 2 priorita, 2,3 krok
          X =        48            ... 3 priorita  4,5 krok
```

Změna ve vyhodnocování při použití závorek:

```
Příklad:  X = (2 + 3) * 4 ^ 2 - 4 / 2 ... pořadí vyhodnoc.
              1   3   2   5   4   ... pořadí vyhodnoc.
          X =  5   *  4   2  - 4 / 2 .. závorky, 1 krok
          X =  5   *  16    - 4 / 2 .. 1prior., 2 krok
          X =        80     - 2     .. 2 prior.,3,4 krok
          X =        78             ... 3 prior., 5 krok
```

Při použití závorek se operace s nejnižší prioritou provedla jako první (tj. provedli jsme sečtení čísel 2 a 3), pak se provede mocnina, dále se zleva provedou operace ná-

sobení a dělení a nakonec se provede odečtení.

Závorky můžeme používat i u ostatních operátorů (relačních, logických).

Relační operátory

se používají při srovnávání dvou aritmetických nebo řetězcových výrazů za klíčovým slovem "IF" při větvení programu nebo v cyklech u klíčových slov "UNTIL" nebo "WHILE". Výsledkem srovnání je logická hodnota "pravda" (true), je-li podmínka splněna a hodnota "nepravda" (false), není-li podmínka splněna.

Relační operátory:

> ... větší

< ... menší

= ... rovno (shoda)

>= ... větší nebo rovno

=< ... menší nebo rovno

<> ... nerovno (neshoda)

Logická hodnota může nabývat dvou hodnot "true", které je přiřazeno číslo "- 1" a hodnota "false", která má hodnotu "0". Logická hodnota je uložena na 16 bitech (2 Byte), false - samé nuly, true - samé jedničky, proto při převo-

du na numerické číslo je true hodnota - 1.

Znak "=" se v Basicu používá ve dvou významech:

a) jako relační operátor "rovná se", neboli shoda v operacích srovnání dvou výrazů (numerických, řetězcových, logických)

b) jako "znak" pro přiřazení výrazu (nebo proměnné, konstanty) vpravo od rovnítka proměnné vlevo. V některých vyšších jazycích (např. Algol, Pascal, ADA a další) používají pro přiřazení jiný znak nebo kombinaci znaků ":=".

Například zápis X= X + 1 je z matematického hlediska nesmysl , ale v programování se často používá a znamená: vyber obsah proměnné označené "X", přičti k tomuto obsahu jedničku a nové číslo ulož do proměnné X.

V našem příkladu s krabičkami máme jednu krabičku označenou písmenem X, z této krabičky vytáhneme lísteček s číslem, k tomuto číslu připočítáme jedničku a lístek vrátíme do té samé krabičky.

Poznámka:
Relační operátory můžeme použít pro vyhodnocení srovnání dvou výrazů a výslednou logickou hodnotu přiřadit do proměnné (numerické).

Příklad: X = A>= B

pak první znak "=" je ve významu přiřaď, druhá dvojice znaků jsou již relační operátory - větší nebo rovno. Do proměnné X se přiřadí hodnota "0" (false), je-li A menší než B a hodnota "-1" (true), je-li A větší nebo rovno B.

Např.: Je-li A=2 a B=3, pak X=0 (false)

a pro A=3 a B=3, pak X=-1 (true)

Pro zvýšení přehlednosti Vám však doporučujeme v přiřazovacím příkazu uzavřít relační výraz do závorek. Např. X = (A >= B). V Basicu se obvykle této konstrukce (tj. přiřazení výsledku relace) nepoužívá. Je však možné ji použít při složitém a nebo vícenásobném použití stejné relace a logickou proměnnou pak použít místo relace.

Příklad: X = (A=B)

```
IF X THEN ... ELSE ...    je-li X true (tj.A=B)
   .                      pak se provede příkaz
   .                      za THEN, v ost. přípa-
   .                      dech za ELSE

IF X THEN ...      ...    další použití
```

Logické operátory

se používají k zápisu a vyhodnocení logických operací, podle pravidel Boolovské algebry. Logické operátory mů-

žeme použít i pro numerické proměnné, které se nejdříve převedou na celé (16ti bitové číslo) a pak se provedou logické operace. Nelze-li číslo převést, hlásí Basic chybu.

Logické operátory:

NOT ... negace, inverze

AND ... logický součin (logické "i")

OR ....logický součet (logické "nebo")

XOR ... neekvivalence

Logický operátor NOT je unární, ostatní operátory jsou binární, tj. operátor NOT se vztahuje pouze k jedné proměnné, před kterou je uveden (podobně jako znaménko "-" mínus u numerických proměnných). Binární operátor spojuje dvě proměnné, mezi kterými je uveden (stejně jako znaménko pro násobení atd.).

NOT - negace - změna logické "0" na "1" a naopak

X	NOT X
0	1
1	0

AND - logický součin (logické "i")

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Logický součin má hodnotu pravda (true, "1") jen je-li A "i" B pravda. V ostatních případech má hodnotu nepravda (false, "0").

OR - logický součet (logické "nebo")

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Logický součet má hodnotu pravda ("1") je-li A "nebo" B pravda. Hodnotu nepravda ("0") má tehdy, je-li A i B nepravda.

XOR - neekvivalence (exclusive OR)

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Logická neekvivalence má hodnotu pravda ("1") nejsou-li prvky shodné. Jsou-li shodné (oba rovny "0" nebo "1"), pak funkce nabývá logickou hodnotu nepravda.
Pozor, nezaměňujte logický operátor XOR za dvojkovou aritmetiku sečítání, která má stejné výsledky jako XOR, s tím rozdílem, že jsou-li oba operandy rovny "1", vzni-

ká přenos do vyšších řádů čísla.

Logické operátory pracují pouze s jednotlivými bity bez ohledu na ostatní bity v operandu.

Nyní si uvedeme pravidlo dvojkového sečítání, abyste si uvědomili rozdíl mezi sečítáním a logickou funkcí XOR.

Dvojkové sečítání

A	B	A+B	přenos
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Při sečítání dvojkových čísel postupujeme stejně jako u desítkových čísel (tj. od nejnižších řádů) a vznikne-li přenos, přičteme ho k nejbližšímu vyššímu řádu.

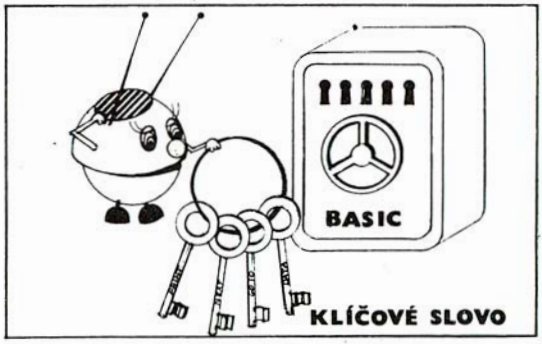
Příklad: Pro jednoduchost budeme počítat pouze s Bytem (místo s 16 bity, jak pracují logické operace).

součet		XOR	
0 0 1 1 1 1 0 0	$(60)_{10}$	0 0 1 1 1 1 0 0	... A
0 1 0 1 1 0 1 0	$(90)_{10}$	0 1 0 1 1 0 1 0	... B
1 1 1 1 0 0 0	...přenos	0 1 1 0 0 1 1 0	A XOR B
1 0 0 1 0 1 1 0	A+B $(150)_{10}$		

Tím jsme se seznámili, jak jsou ukládány proměnné v Basicu a jaké operátory můžeme v Basicu používat a jak pracují. V následující části se seznámíme s prvky programů.

4.4. Prvky programů v Basicu

V této části se ve stručnosti seznámíme s pojmy jako jsou příkazy, povely, podprogramy, funkce, skoky podmíněné a nepodmíněné, přepínače, cykly. V každém programovacím jazyku jsou vybrána určitá slova nebo zkratky, kterým říkáme "klíčová slova" a překladač (interpret) Basicu je vždy provede stejně.



Klíčovým slovům, které provádí určitou činnost v programu nebo v dialogovém (přímém) režimu a mají přímou souvislost s programem, říkáme příkazy

(instrukce, statements).

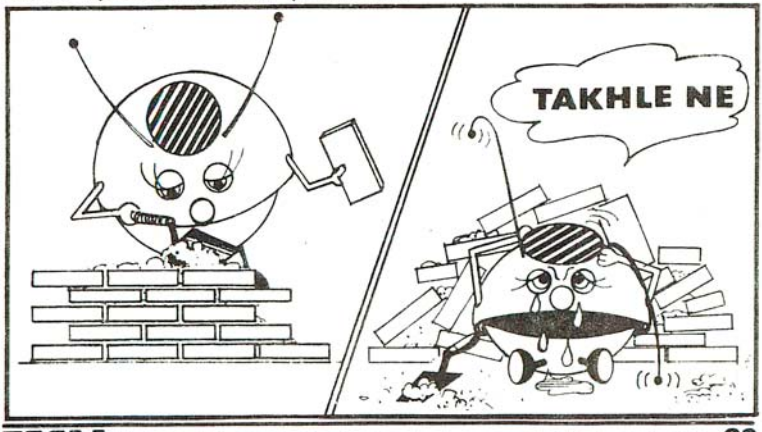
Klíčová slova, která nemají přímou souvislost s programem a pouze řídí překladač nebo ovládají a sledují chod programu, říkáme povely (commands). Některé povely nelze zadávat v programu.

Mezi povely řadíme například klíčová slova RUN pro spuštění programu, LIST pro výpis programu atd.

V dialogovém režimu nemůžeme použít všechny příkazy, ale můžeme použít všechny povely. Obvykle se povely používají v dialogovém režimu a výjimečně se použijí v programu.

Některé příkazy se skládají z více klíčových slov. Např. příkaz pro nepodmíněný skok GO TO, který se skládá z klíčového slova "GO" a "TO". Podmíněný skok dokonce ze tří slov "IF", "THEN", "ELSE". Některá klíčová slova se vyskytují v různých příkazech. Například slovo "TO" se vyskytuje v nepodmíněném skoku "GO TO" a příkazu cyklu "FOR, TO". Klíčová slova nesmí být jménem (identifikátorem) proměnné ani jeho části. Tak např. identifikátor TON je nepřípustný, protože obsahuje klíčové slovo "TO".

Při výkladu příkazu jsme použili slovo "program". V předcházející části jsme si řekli, že programové řádky jsou očíslovány a provádí se až po spuštění programu povelem RUN. Pokud zadáme příkazový nebo povelový řádek bez čísla řádku, pak jsme zadali povel, příkaz v přímém (dialogovém) režimu a po ukončení řádku znakem "CR" se povel okamžitě provede. Posloupnost programových řádků zapsaných podle pravidel jazyka Basic tvoří program. Ovšem náhodná posloupnost příkazů by neprováděla požadovanou činnost a tak musí být program správně zapsán podle pravidel a dále musí mít logickou výstavbu. Stejně jako když z cihel stavíme dům, když je budeme pokládat náhodně, dům nepostavíme, ale když budeme stavět podle plánu, pak stavba povede k cíli. Cihly jsou příkazy a plánem je "algoritmus" úlohy (tj. vytvoření posloupnosti jednoduchých kroků, které vedou k vyřešení daného problému.



Program můžeme přirovnat k železniční trati, po které jede vlak, který může jet pouze dopředu (příkazové řádky se provádí zleva doprava a podle vzestupných čísel řádků). Ovšem někdy si potřebujeme program rozvětvit podle výsledku operace, stejně jako se rozvětvují koleje výhybkami nebo se vrátit zpět a část programu opakovat. K tomu slouží příkazy pro **větvení** programu - podmíněné skoky, cykly, přepínače.

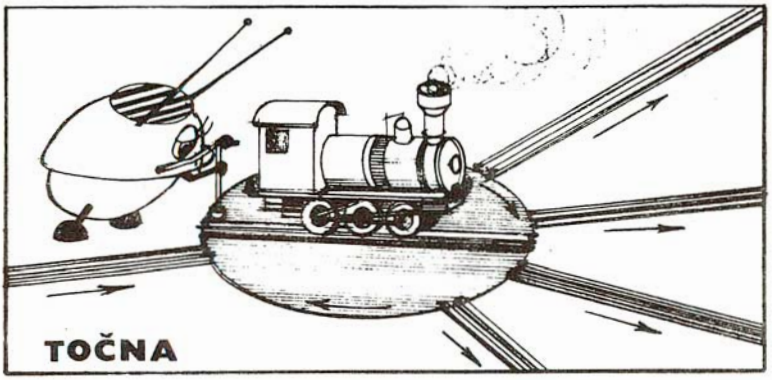
Podmíněný příkaz má tvar IF podmínka THEN příkaz 1 ELSE příkaz 2. Je-li podmínka za IF splněna, provede se příkaz 1 a příkaz 2 se neprovede. Není-li splněna, pak se provádí příkaz 2. Stejně jako výhybka, která může být nastavena na jednu nebo druhou kolej.

Když jsme se zmínili o podmíněném příkazu (skoku), vysvětlíme si další příkaz "nepodmíněný skok". Nepodmíněný skok nám umožňuje přerušit normální posloupnost provádění příkazů a v provádění pokračovat na zadaném čísle řádku, které je uvedeno za příkazem GO TO. Tento příkaz například používáme pro spojení jednotlivých větví pro zpracování podmíněných příkazů. Pokud je nepodmíněný skok použit na vícepříkazovém řádku, pak musí být posledním příkazem na řádku (kromě poznámky), protože příkazy uvedené za nepodmíněným skokem by se nikdy nemohly provést.

Pokud si tento příkaz představíme jako koleje, pak příkaz GO TO bude představovat oblouk, který odklání trať jinam od přímého směru a číslo řádku opět představuje výhybku, která se umístí na řádek podle čísla a spojuje dvě koleje do jedné.

Někdy potřebujeme rozvětvit program do více větví podle hodnoty řídicí proměnné. Problém můžeme řešit posloupností podmíněných příkazů, ale Basic je vybaven příkazem, který daný problém řeší za nás - přepínačem. Jsou dva typy přepínačů - přepínač nepodmíněných skoků a přepínač podprogramů. Přepínače si vysvětlíme na přepínači nepodmíněných skoků, protože pojem podprogram jsme zatím nevysvětlili. Příkaz má tvar ON proměnná GO TO N1, N2, N3 ... ELSE N_m, kde N_i jsou čísla řádků, kam se má skočit, bude-li hodnota proměnné rovna 1,2,3... Není-li hodnota proměnné rovna žádnému požadovanému číslu seznamu skoků, pak se provede skok na řádek uvedený za ELSE. Pokud ELSE není uvedeno, program pokračuje následujícím příkazem za přepínačem. Např. příkaz ON I GO TO 10, 20, 50 ELSE 100 provede skok na řádek 10, je-li i=1, na řádek 20 pro i=2 a na řádek 50 pro i=3. Pro všechny ostatní hodnoty i se provede skok na řádek 100.

V našem kolejovém modelu si můžeme přepínač představit jako "točnu", na kterou vjede vlak a ta se pak s celým vlakem otočí (podle hodnoty proměnné) do požadovaného směru a vlak může pokračovat v cestě.



Nyní bychom si měli vysvětlit pojem podprogram, jeho použití v programech. Často se nám stane, že v různých místech programu potřebujeme opakovat stejnou posloupnost příkazů. Pak by bylo vhodné posloupnost zapsat pouze jednou v programu a mít možnost tuto posloupnostpříkazů použít na různých místech v programu podle potřeby. Této posloupnosti příkazů budeme říkat podprogram. Aby Basic věděl, že danou posloupnost chceme volat jako podprogram, má pro vo-

lání podprogramů speciální příkaz GO SUB.

Příkaz RETURN označí, kde končí podprogram a kdy se má vrátit k provádění hlavního programu.

Co udělá Basic, když při provádění příkazů programu narazí na příkaz GOSUB N (kde N je číslo řádku, kde začíná podprogram)? Musí přerušit posloupnost provádění příkazů (poznačí si číslo prováděného řádku a adresu příkazů za příkazem GOSUB, aby věděl, kam se má vrátit po vykonání podprogramu) a provede skok na označený řádek (podle čísla řádku) a začne provádět program (podprogram) od daného čísla řádku tak dlouho, až narazí na příkaz konce podprogramu RETURN. Potom si musí vyzvednout číslo řádku a adresu příkazu za voláním podprogramu (tyto údaje si Basic ukládá do zásobníku - sklípku. Příkaz GOSUB provede uložení a příkaz RETURN si údaje vybere ze zásobníku.) Program pokračuje za voláním podprogramu.

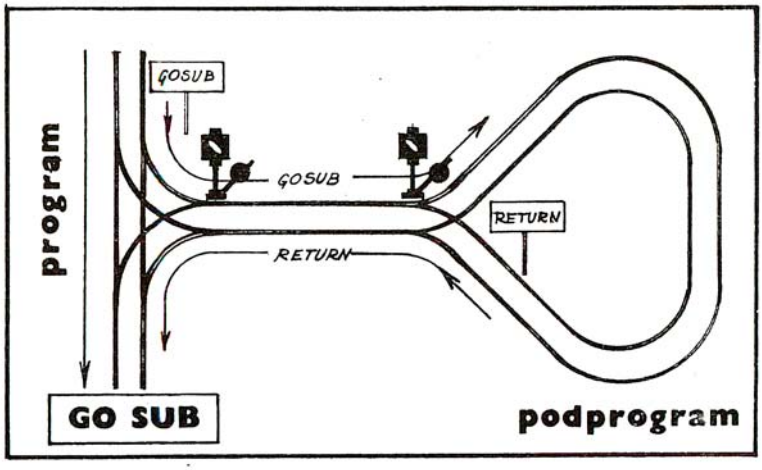
Z podprogramu můžeme volat další podprogramy. Volání podprogramů z programů a podprogramů říkáme vnořování podprogramů. Počet vnoření není teoreticky omezen, ale prakticky je omezen velikostí zásobníku. Podprogramy nám šetří čas při vytváření programů, místo v paměti a programy rychleji odladíme, protože jednou odladěný pod-

program používáme vícekrát (na různých místech v programu). Rovněž při opravách opravujeme pouze na jednom místě a víme, že jsme opravu provedli na všech místech v programu, kde se volá podprogram.

Nyní zbývá odpovědět na otázku, kdy použijeme podprogram? Je samozřejmé, že pro jeden příkaz nebudeme volat podprogram, ale že příkaz napíšeme znovu, protože volání podprogramu je poměrně zdlouhavá operace a navíc je ještě příkaz návratu. Takže bychom prakticky nic neušetřili. Jakmile máme více příkazů a častější požadavek na opakování, pak nebudeme váhat a danou posloupnost zapíšeme jako podprogram. Někdy i v případě jednoho příkazu (složitý výraz) nebo předpokládáme, že výraz se bude měnit (nebo závisí na vstupních podmínkách) a je velice často používán, uvedeme z praktických důvodů daný příkaz jako program.

V naší analogii s kolejemi je podprogram složitější pro znázornění. Příkaz GOSUB nám vybuduje výhybku s dvěma oblouky. Přímé koleje se odkloní obloukem na výhybku, která směřuje na trať zvanou "podprogram". Náš vlak se bude pohybovat po trati podprogramu, až narazí na příkaz RETURN. Tento příkaz si opět můžeme představit jako výhybku, která nás zavede na spojku, kterou jsme postavili příkazem

GOSUB. Oba příkazy ovládají obě výhybky. Příkaz GOSUB nastaví první výhybku na spojku a druhou výhybku tak, aby vlak vjel na začátek podprogramu a příkaz RETURN nastaví druhou výhybku tak, aby se vlak vrátil na spojku a první výhybku nastaví tak, aby vlak pokračoval za příkazem GOSUB.



Příkazy cyklu (smyčky)

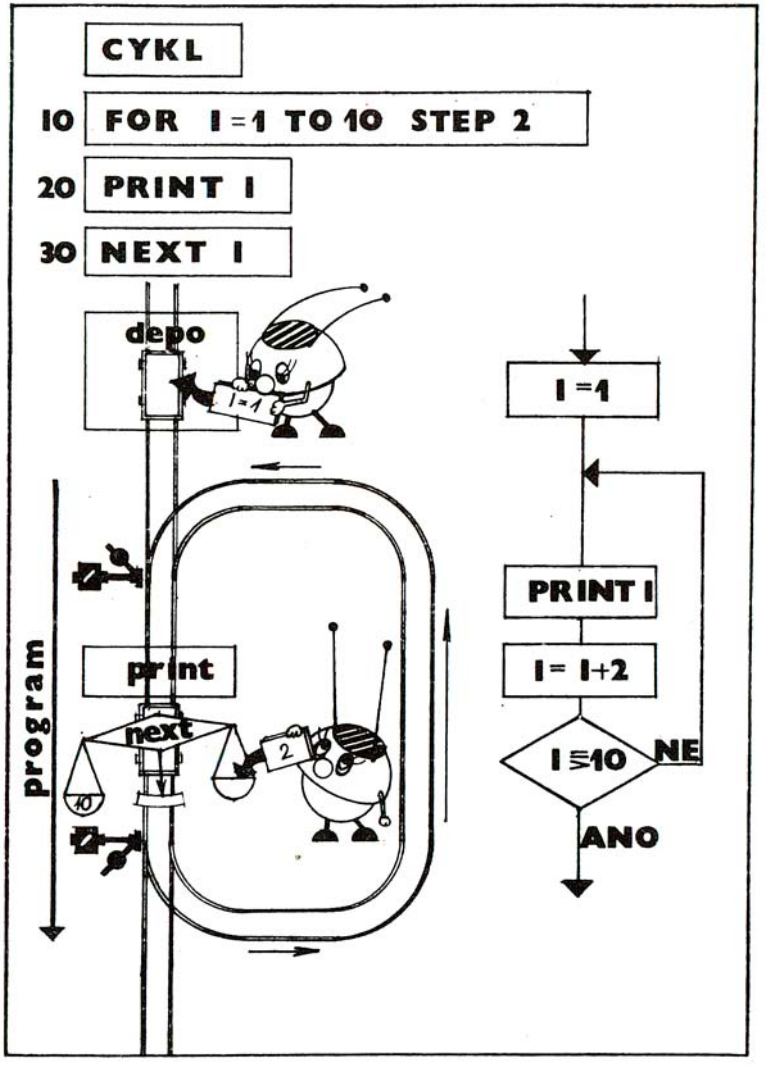
V mnoha případech potřebujeme opakovat část programu po krocích, dokud řídící proměnná nedosáhne určité hodnoty. Po dosažení hodnoty,nebo při splnění určité podmínky, chceme přerušit opakování a pokračovat v dalším zpracování programu. Uvedený problém můžeme řešit několika způsoby. Za prvé , že budeme opakovat posloupnost příka-

zů s různými hodnotami řídicí proměnné, za druhé, že použijeme podmíněných skoků. První způsob nemá smyslu, protože bychom museli psát dlouhý a nepřehledný program. Druhý způsob se již používá častěji. Abychom nemuseli vždy vytvářet si svoje řízení smyčky, je překladač Basicu vybaven příkazy pro řízení smyček (cyklů). Jako základní příkaz pro řízení cyklů je příkaz FOR řídicí proměnná, TO horní mez (STEP krok řídicí proměnné). Za klíčovým slovem FOR je přiřazení výchozí hodnoty řídicí proměnné. Pro označení konce smyčky (tj. posloupnosti příkazů, které se mají provádět) je příkaz NEXT. Příkaz pracuje tak, že se řídicí proměnné přiřadí výchozí hodnota a provede se smyčka až do příkazu NEXT. Tento příkaz provede zvýšení řídicí proměnné o krok a pak provede srovnání řídicí proměnné se zadanou horní mezí. Pokud je řídicí proměnná menší než horní mez, vrátí se provádění smyčky na počátek, je-li větší, pak se provádění smyčky ukončí a program pokračuje následujícím příkazem za příkazem NEXT. Klíčové slovo STEP v příkazu cyklu je nepovinné a určuje, o jaký "krok" (hodnotu) se bude měnit řídicí proměnná cyklu. Není-li krok uveden, bere se za krok hodnota "+1". Řídicí proměnná nemusí být celočíselná a "krok" může být libovolný. Příkaz na počátku nekontroluje, je-li výchozí hodnota menší než horní hranice, z toho vyplývá, že smyčka se vždy provede alespoň jednou.

Smyčka nemusí být pouze vzestupná, ale můžeme smyčku provádět sestupně. V tomto případě počáteční hodnota je větší než "mez" uvedená za klíčovým slovem "TO" a musí být uveden záporný "krok".

Jak zobrazit cyklus na naší trati s vlakem? Příkaz FOR nám naplní vagon - řídicí proměnnou základním nákladem a zároveň nastaví automatickou váhu, která bude vážit vagon - řídicí proměnnou. Dále příkaz FOR vybuduje spojovací výhybku, kterou umístí za první naplňování vagonu (tj. za příkaz cyklu). Vlak se rozjede po trati až přijede k příkazu NEXT. Zde se vagonek doplní o další bednu s nákladem (krok cyklu) a najede na automatickou váhu, která ovládá výhybku. Není-li vagon dostatečně naložen, nastaví výhybku tak, aby se vlak vrátil zpět na nakládání. Je-li již naložen, nastaví výhybku na pokračování. U sestupného cyklu nebudeme na vagon nakládat, ale budeme z vagonu "řídicí proměnná" náklad vykládat.

Basic EXP byl doplněn dalšími příkazy cyklu podle jazyka Pascal. Jsou to příkazy REPEAT - UNTIL a WHILE--DO-WEND. Příkaz REPEAT (opakuj) uvádí posloupnost příkazů, které se mají opakovat. Posloupnost je ukončena příkazem UNTIL, za kterým je uvedena podmínka. Je-li podmínka splněna, pak program pokračuje za příkazem

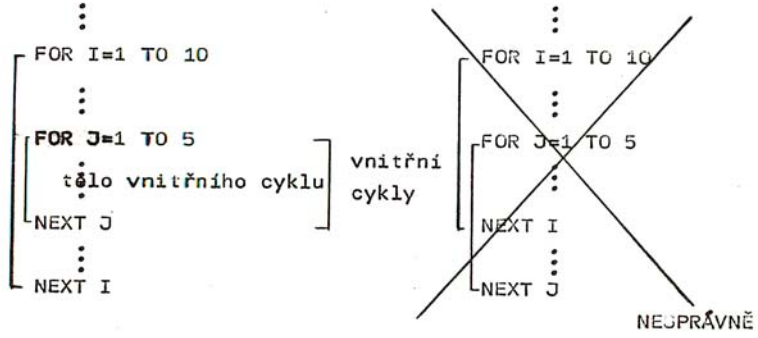

CYKL
10 FOR I=1 TO 10 STEP 2
20 PRINT I
30 NEXT I
depo
I=1
print
next
2
10
program
I=1
PRINT I
I= I+2
I≤10
NE
ANO

UNTIL (tj. je ukončen cyklus), není-li podmínka splněna, program se vrací za příkaz REPEAT. U tohoto příkazu není přímo uvedena řídicí proměnná a její změnu musíme sami zajistit v "těle cyklu" (tj. mezi příkazy REPEAT - UNTIL). Protože podmínka se vyhodnocuje až po provedení těla cyklu, tak cyklus proběhne alespoň jednou.

Druhým typem cyklu je příkaz WHILE podmínka DO. (WHILE - dokud je splněna podmínka DO - dělej). Za klíčovým slovem DO následuje vlastní tělo cyklu. Aby překladač poznal, kde končí tělo cyklu, je použito klíčové slovo - příkaz WEND (WHILE - END - konec příkazu WHILE). Pro řídicí proměnnou platí stejné zásady jako u příkazu REPEAT. U příkazu WHILE se prvně provádí vyhodnocení podmínky a je-li podmínka splněna, pak se provede tělo cyklu, není-li splněna, tělo se neprovede. Z toho vyplývá, že v případě,není-li podmínka splněna, při vstupu do cyklu tělo se neprovede ani jedenkrát. Pozor, u těchto příkazů můžeme vytvořit nekonečnou smyčku, když zapomeneme měnit řídicí proměnnou v těle cyklu!

V těle cyklu můžeme vyvolat další cyklus (vnitřní neboli vnořené cykly). U vnořených cyklů ovšem platí zásada, že vnitřní cyklus musí být ukončen v těle vnějšího

cyklu (tj. FOR - NEXT vnitřního cyklu musí být za FOR a před NEXT vnějšího cyklu).



Další důležitou zásadou u cyklů je princip výstupu z cyklu výstupním bodem, tj. příkazem NEXT (resp. UNTIL, WEND). Výstup z cyklu jiným způsobem (např. GO TO) může vést k chybě programu. Proto pokud chcete předčasně ukončit cyklus, změňte řídící proměnnou cyklu tak, aby byla splněna podmínka pro výstup cyklu a skočte na příkaz konce cyklu.

Podobně je tomu i u podprogramů, kdy návrat z podprogramu se provádí výstupním bodem, tj. příkazem RETURN. Tento princip odpovídá požadavkům moderního strukturovaného programování, kdy bloky provádějící určitou činnost mají přesně určené vstupní a výstupní body z bloků. Jazyk

Basic toto pravidlo umožňuje porušovat tím, že umožňuje skok na libovolný řádek. V některých případech nesprávného skoku nahlásí chybu (např. provedeme-li skok do těla cyklu, aniž bychom vstoupili přes vstupní bod cyklu - příkaz FOR). Proto si proveďte řádnou přípravu před vlastním programováním, úlohu si rozdělte na funkční body s přesnými vstupy a náplní činnosti těchto bloků. Příkazy nepodmíněných skoků používejte co nejméně. Při dobré analýze řešeného problému zjistíte, že se téměř obejdete bez příkazu GO TO.

Abyste si mohli program rozdělit do bloků (a s blokem pracovat jako s jediným příkazem), je Basic EXP doplněn strukturami bloků z jazyka Pascal. Jsou to příkazy "BEG" a "BEND" (BEG je zkratkou slova BEGIN - z Pascalu a znamená počátek bloku, BEND - ukončuje blok - BLOCK END). Nebylo možné použít pouze slovo END, protože je již v Basicu použito pro označení konce programu). Bloky nám umožňují vytvářet složité rozhodovací struktury, vkládat do sebe podmíněné skoky.

Nyní nám zbývá vysvětlit pojem "funkce". Funkce je zvláštním typem podprogramu, který má přiřazeno jméno. Při definování funkce se určí jméno funkce a činnost, kterou má funkce vykonat. Výsledkem funkce je jedna hodnota, která se vrací při vykonání funkce. Pro definování jedno-

řádkové funkce je vyhraženo klíčové slovo DEF a jméno funkce musí začínat písmeny FN, za kterými následuje identifikátor (jméno) funkce. Funkce může mít jednu vstupní proměnnou. Za rovnítkem na pravé straně je výraz, který má funkce provést. Tak např. často se nám v programu bude vyskytovat vzorec: $X^2 + X + 10$. Abychom ho nemuseli neustále psát, nadefinujeme si funkci např.

```
DEF FNA(X)= X *(X+1) + 10
```

Pokud v programu potřebujeme řešit uvedenou funkci, provedeme volání funkce. Všude tam, kde můžeme použít identifikátor (tj. na pravé straně přiřazovacího příkazu, ve výrazech atd.) napíšeme jméno funkce a parametr, pro který chceme provést výpočet.

Příklad definování a použití funkce:

```
10 DEF FNA(X)=X*(X+1)+10   ! definice funkce X^2+X+10
20 INPUT "I="; I           ! vstup čísla z klávesnice a
                             uložení do proměnné I
30 PRINT FNA(I)            !tisk výsledku funkce X^2+X+10
40 IF I<>0 THEN 20         ! není-li I=0, pak opakuj výpočet, pro I=0 ukonči
50 END
RUN     ... spuštění programu
```

```
I=? 1  ... program vytiskne I= a "?" a očekává vstup
           čísla. Zadali jsme "1"
           Pozn.: podtržený znak zadáváme z klávesnice
12     ... výsledek
I=? 2
16
.
.
I=? 0  ... ukončení programu
```

Jednořádkové funkce mohou být pouze pro numerické proměnné. Kromě uživatelem definovaných funkcí má jazyk Basic zabudované funkce pro numerické proměnné (např. ABS(X) ... absolutní hodnota SIN(X) ... sinus atd.), funkce pro práci s řetězci (např. LEN(X$) počet znaků v řetězci ...) a další pomocné funkce (např. TAB(30) ... tabulátor). Tyto zabudované funkce značně ulehčují práci programátorům tím, že nemusí pracně vymýšlet již vymyšlené a často používané funkce. Zřejmě mnozí z vás by ani nevěděli, jak naprogramovat třeba goniometrické funkce.

Basic EXP je doplněn možností definování víceřádkových uživatelských funkcí a podprogramů jako pro numerické, tak i pro řetězcové funkce. Tyto funkce a podprogramy budou popsány až při popisu příkazů jazyka Basic EXP.

Poslední částí, kterou stručně vysvětlíme, jsou příkazy pro vstup čísel a znaků z klávesnice nebo jiného zařízení a výstup znaků na obrazovku nebo tiskárnu či kazetový magnetofon. Abychom mohli programu v Basicu zadávat vstupní data, která potřebujeme zpracovat, musí mít jazyk nějaké prostředky, které nám umožní vstup dat. Takovým prostředkem je příkaz INPUT. Pro výstup dat je to příkaz PRINT.

Za příkazem pro vstup znaků INPUT je uveden seznam proměnných, které chceme naplnit z vnějšího zařízení. V seznamu mohou být jak numerické, tak i řetězcové proměnné. Numerické proměnné převádí překladač (interpret) Basicu do vnitřní formy a u řetězců ponechává vstupní data ve znakové formě. Pokud příkazem INPUT zadáváme pouze jednu proměnnou, ukončíme její zadání stiskem klávesy "↵" (tj. CR), která ukončuje vstup znaků z klávesnice. V případě, že potřebujeme vstup více proměnných, můžeme je zadat najednou nebo postupně. Jestliže je zadáváme najednou, musíme jednotlivá data od sebe oddělit, aby překladač poznal, kde končí jedny a kde začínají druhé. V Basicu je jako oddělovač dat použit znak "," - čárka. Mezera je v numerických datech nevýznamná a překladač ji ignoruje (vynechává). Proto kdybychom čísla zapsali pouze oddělená mezerami,

spojí jednotlivá čísla do jednoho.

Příklad: Potřebujeme vstup tří čísel a uložit je do proměnných I, J, K. Pak příkaz INPUT I,J,K provede požadavek na čtení dat z klávesnice. Na obrazovce se zobrazí znak "?", že program potřebuje data.

Zadáme-li z klávesnice 1,2,3, pak do proměnné I se přiřadí 1,do J=2 a K=3. Pokud bychom zadali čísla 1 2 3, pak se do proměnné I přiřadí hodnota 123 a na obrazovce se vytiskne opět "?", že příkaz INPUT potřebuje data pro zbývající proměnné.

Nyní na chvíli přerušíme výklad příkazu pro vstup dat a věnujeme se výstupu, tj. příkazu PRINT. Tento příkaz umožní výstup dat na obrazovce displeje. Vystupovat mohou jak proměnné, tak i konstanty. Proměnné mohou být numerické nebo řetězcové (znakové). O jaký typ proměnné se jedná pozná překladač podle znaku měny ("$") pro řetězce. Znakové konstanty se zapisují s uvozovkami. Tak např. příkaz PRINT "AHOJ" po ukončení "CR" vypíše na obrazovce text AHOJ. Příkaz pro výstup umožňuje výstup více proměnných a konstant v jednom příkazu. Podle použitého znaku pro oddělení proměnných se provádí umístění výstupních znaků na obrazovce. Znak "," čárka umístí data na další ta-

belační pozici (tj. modul o 14 pozicích), znak ";" středník umístí data bezprostředně za předcházející data. Dále jsou pro příkaz PRINT v překladači zabudovány funkce, které umožňují umístění i na jiná místa na obrazovce. Tyto funkce budou podrobně popsány u popisu příkazu PRINT. Není-li poslední proměnná ukončena žádným že znaků "," nebo ";", pak se po výpisu posledních dat přejde na nový řádek. Vynechání řádku se provede použitím příkazu PRINT bez proměnných a konstant. Pomocí příkazu PRINT si můžeme vypsat nápovědný text pro vstup dat. Tak např. pro vstup tří čísel můžeme zapsat:

```
10 PRINT "I,J,K=";        ! tisk textu na obrazovce
20 INPUT I,J,K            ! vstup 3 čísel
```

Po spuštění tohoto programu se na obrazovce vypíše: I,J,K=? a program čeká na zadání tří čísel, která uloží do proměnných I,J,K.

Nyní se opět vrátíme k příkazu INPUT. Abychom nemuseli vždy pro výpis pomocného textu při vstupu používat příkaz PRINT, je příkaz INPUT doplněn možností výpisu zadaného řetězce v příkazu INPUT. Řetězcová konstanta musí být uvedena v uvozovkách a ukončena oddělovačem (čárka, středník). Výše uvedený příklad můžeme zapsat:

```
INPUT "I,J,K="; I,J,K
```

Výsledek bude úplně stejný jako v předcházejícím příkladu při použití příkazu PRINT. Je vidět, že tato vlastnost příkazu INPUT nám bude šetřit práci při programování a zlepšuje přehled o vstupech dat.

Příkaz INPUT umožňuje vstup dat nejen z klávesnice, ale i z kazetového magnetofonu. Aby překladač poznal, odkud bude brát vstupní data, je příkaz INPUT pro případ čtení dat z kazety doplněn označením zařízení. Tvar příkazu pro vstup z kazetového magnetofonu je INPUT#R seznam ... kde #R označuje vstup z kazety, seznam ... vstupní proměnné. Podobně je tomu i u příkazu PRINT, který může vypisovat data nejen na obrazovce, ale i na tiskárně nebo zapsat data na kazetu. Příkaz je rovněž doplněn označením výstupního zařízení. Možnosti jsou uvedeny u popisu příkazu PRINT.

Kromě uvedených možností vstupu z klávesnice nebo kazetového magnetofonu je překladač jazyka Basic vybaven příkazy pro čtení dat, které jsou součástí programu. Pro tuto činnost jsou v Basicu příkazy DATA, READ, RESTORE. Příkazem DATA si deklarujeme v programu data (numerické nebo znakové) a příkazem READ můžeme tato data přiřadit proměnným. Příkaz RESORE umožní vrátit se zpět na požadovaná data pro příkaz READ. Příkaz READ postupně vybírá údaje definované příkazem DATA (nastavuje ukazovátko

na požadovaná data.

Příklad: Řetězcové proměnné A$ chceme postupně přiřazovat jména dnů a pak je vytisknout.

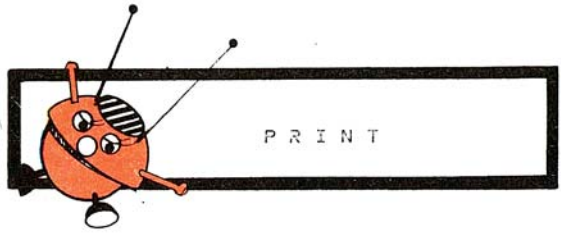
```
10 DATA "PONDELI","UTERY","STREDA","CTVRTEK","PATEK","SO-
BOTA","NEDELE"
20 FOR I=1 TO 7
30 READ A$
40 PRINT A$
50 NEXT I
```

Uvedený jednoduchý program nám postupně zobrazí jména dnů. Řádek 10 deklaruje řetězcová data - jména dnů. V cyklu pak postupně přiřazujeme do proměnné A$ jména dnů, která zobrazíme.

V našem přirovnání Basicu k vlaku si můžeme vstupy přirovnat k nakládání vagonů v průběhu pohybu vagonu po trati, nazvané program. Výstupy můžeme přirovnat k vykládání vagonů. Obě operace nakládání a vykládání (tj. vstupy a výstupy) jsou velmi důležité. Proč by měl jet vlak, který by nic nevezl, nenaložil ani nevyložil a tak bychom se ani nedozvěděli, že vlak něco vezl.

5 PŘÍKAZY JAZYKA BASIC EXP V5.0/G

Příkazy jazyka Basic, který je implementován na mikropočítači ONDRA si budeme probírat postupně tak, abyste si je mohli vyzkoušet na mikropočítači a snáze pochopili jejich funkci. Některé příkazy si budeme vysvětlovat vícekrát, protože mají více významů a ne vždy byste okamžitě pochopili jejich činnost bez předchozího vysvětlení.



Příkaz pro výstup znaků (řetězců a numerických znaků) na určeném výstupním zařízení. Nejdříve si probereme zobrazení na obrazovce displeje.

Formát příkazu:

PRINT seznam

kde seznam může obsahovat řetězcové konstanty, numerické konstanty, řetězcové proměnné, numerické proměnné a výrazy, funkce jak numerické, tak i speciální funkce pro formátování výstupu. Prvky v seznamu jsou odděleny oddělovači. To, co je složitě napsáno, vysvětlíme si na příkladech a tak pochopíme, jak používat příkaz PRINT.

Napíšeme si řádek s příkazem, kde použijeme řetězcovou konstantu AHOJ.

10 PRINT "AHOJ" ... řetězcová konstanta je uzavřena mezi uvozovkami

RUN ... spuštění nejjednoduššího programu (o jednom příkazu)

Na obrazovce se napiše text na novém řádku:

AHOJ

Nyní si program doplníme o další příkaz s řetězcovou konstantou. Napíšeme další řádek s číslem 20.

20 PRINT "ZDRAVI TE ONDRA"

V paměti počítače zůstal zachován původní řádek s číslem 10 a tak po spuštění programu příkazem RUN se nám na obrazovce vypíše text:

AHOJ

ZDRAVI TE ONDRA.

Můžeme si povšimnout, že další příkaz je vypsán na novém řádku, protože jsme nepoužili žádný oddělovač a náš seznam má pouze jeden prvek. Nyní si doplníme další příkaz PRINT bez seznamu a příkaz s numerickou konstantou. Napíšeme řádky 30 a 40.

30 PRINT

40 PRINT 1986

Po spuštění programu příkazem RUN se zobrazí text:

AHOJ
ZDRAVI TE ONDRA

1986

Příkaz PRINT bez seznamu nám vynechal jeden řádek na obrazovce. Číselné konstanty se zobrazují s levostrannou mezerou, která je pro znaménko. Záporné číslo bude mít místo mezery znaménko "-" mínus.

Abyste si usnadnili práci při psaní příkazu PRINT, má Basic V5.0 zabudován jednoznakový příkaz "?" (otazník), který lze použíti místo klíčového slova PRINT, při zápisu příkazu. Do paměti se však uloží celé klíčové slovo PRINT.
O tom se můžete přesvědčit, když použijete příkaz pro výpis programu (LIST).

Nyní si vysvětlíme použití oddělovačů v seznamu. Jednotlivé prvky seznamu můžeme od sebe oddělit znakem "," (čárka) a ";" (středník). Znak čárka určuje, že zobrazení (tisk) čísla nebo řetězce se provede na další tabulační pozici (modulo 14). Zrušíme si zdravici příkazem NEW, který zruší program v paměti. Napíšeme si příkaz, který nám zobrazí použití oddělovače čárka.

```
10 ?1,2,3,4,5,6,7,8
```

Po spuštění se na obrazovce zobrazí numerické konstanty

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

První číslice se zobrazí ve druhém sloupci (levostranná mezera místo znaménka plus), další na 15 sloupci, třetí na 29 sloupci a čtvrtá by se měla zobrazit na 43 sloupci, ale displej má pouze 40 znaků na řádek a proto číslice 4 se na novém řádku zobrazí na třetí pozici. Další znaky jsou na pozicích 17 (57), 31 (71) a číslice 7 by se měla zobrazit na 45 (85) a tudíž se opět provede nový řádek a číslice 7 se zobrazí v 5 sloupci a poslední číslice 8 se zobrazí ve sloupci 19 (99).

Řetězcové konstanty se zobrazují bez levostranné mezery (nepředpokládá se znaménko) a proto se znaky zobrazí na první, čtrnácté atd. pozici.

Oddělovač čárka může být použit i za posledním prvkem seznamu, pak ovšem další příkaz PRINT nezačíná na novém řádku, ale pokračuje na další tabelační pozici.

Dalším oddělovačem je znak středník. Ten pracuje tak, že následující prvek seznamu je zobrazen nebo vytištěn na další volné pozici. U řetězců provede spojení znaku obou řetězců, u numerických proměnných a konstant je nutno si

uvědomit, že číslo je zobrazováno s levostrannou mezerou a je ukončeno rovněž mezerou (pravostrannou).

```
10 PRINT "A";"B";1;2
```

V příkazu jsou použity řetězcové a numerické konstanty a po spuštění příkazem RUN se zobrazí:

```
AB 1 2
```

Řetězcové konstanty A a B se zobrazí společně, číslo 1 je od řetězce odděleno jednou mezerou a číslice 2 je oddělena dvěma mezerami (pravostranná mezera od čísla 1 a levostranná mezera od čísla 2).

Říkali jsme si,,že v seznamu mohou být i výrazy. Interpret Basicu nejdříve vyhodnotí výraz a pak zobrazí výsledek.

```
10 PRINT 2*3/4
```

Po spuštění zobrazí na obrazovce výsledek

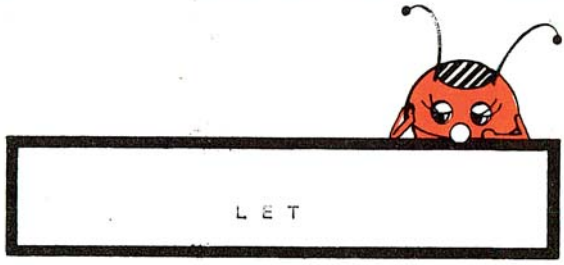
```
1.5
```

nejdříve vyhodnotí výraz podle pravidel priorit (násobení a pak dělení) a u stejných priorit zleva doprava (tj. provede násobení 2 krát 3 = 6 a pak výsledek vydělí 4 a dostane konečný výsledek 1.5).

Zde si povšimněte, že u desetinných čísel se používá desetinná tečka a nikoliv čárka, jak jsme zvyklí používat v matematice. Čárka je oddělovač prvků seznamu!

Této vlastnosti Basicu, že příkaz PRINT umožňuje vyhodnotit výraz, můžeme s výhodou využít pro zkrácení zápisu programu, když výsledek potřebujeme pouze zobrazit a dále s ním nebudeme pracovat.

Nyní vysvětlování příkazu PRINT přerušíme a seznámíme se s přiřazovacím příkazem LET.



Příkaz LET je pro přiřazení výrazů a konstant do proměnné uvedené na levé straně přiřazovacího příkazu. Přiřazovat můžeme numerické nebo řetězcové výrazy podle typu proměnné.

Formát příkazu:

```
LET proměnná = výraz
```

nebo

```
proměnná = výraz
```

klíčové slovo "LET" není povinné a v Basicu V5.0 doporučujeme klíčové slovo LET nepoužívat, protože šetříme místo v operační paměti a čas při zápisu programů. Používejte ho tehdy, budete-li svoje programy přenášet na jiné systémy, které použití klíčového slova LET vyžadují. Proměnná může být jednoduchá numerická nebo prvek pole (s indexem) nebo

řetězcová proměnná nebo prvek pole. Výraz je uveden na pravé straně od znaku "=" (rovnítko) a musí odpovídat typu proměnné na levé straně (tj. numerické číslo nebo znakový řetězec). Výsledek vyhodnocení výrazu musí být číslo nebo řetězec. Výraz může obsahovat proměnné, pak se obsah proměnné použije při vyhodnocení výrazu. Ve výrazu můžeme použít aritmetické, logické a relační operátory. Pro změnu priorit při vyhodnocování výrazů můžeme použít kulatých závorek.

Pozor znak "=" v přiřazovacím příkazu neznamená "rovná se", nýbrž do proměnné na levé straně od rovnítka přiřaď hodnotu výrazu na pravé straně.

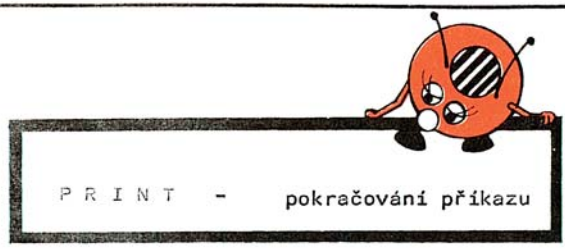
Připomínáme, že identifikátor proměnné může být libovolně dlouhý, ale Basic V5.0 pracuje pouze s prvními dvěma znaky.

Příklady použití:

X = 5	proměnné X přiřaď hodnotu "5" (číslo)
Y = X	do proměnné Y přiřaď hodnotu proměnné X (tj. 5)
A=2+(X-3)*4	do proměnné A přiřaď hodnotu výrazu, který se vypočítá podle pravidel - závorka, násobení, součet. V proměnné A bude uložen výsledek 10 (pro X=5).

`B = 10 + 5 OR 3`	do proměnné B se uloží výsledek hodnocení výrazu, nejdříve se provede součet a pak logická operace "nebo" a výsledek 15 se uloží do B.
`C = 10 +(5 OR 3)`	do proměnné C se uloží výsledek 17, nejdříve se vyhodnotí výraz v závorce a pak se přičte konstanta 10.
`D = 5 OR 3+10`	do proměnné D se uloží výsledek 13, nejdříve se provede součet 3+10 a pak se provede logická operace OR.
`LET X = Y+1`	stejné přiřazení použito klíčové slovo
`X = Y+1`	bez použití klíčového slova
`E = 2*SIN(0.5)`	do proměnné E se přiřadí hodnota funkce sinus pro 0.5 radiánu vynásobená dvěmi. Výsledek 0.958851 je uložen v proměnné E.
`A$ = "AHOJ"`	do řetězcové proměnné se uloží znaky AHOJ.
`B$ = A$+" KAMARADE"`	do řetězcové proměnné B se uloží řetězec AHOJ KAMARADE
`C$ ="123"`	do řetězcové proměnné C se uloží znaky 123 (nikoliv číslo, se kterým můžeme počítat!). Pro převody mezi znaky a čísly jsou v Basicu vestavěné funkce.

Výrazy mohou být velmi složité, zde jsme omezeni délkou řádky, která může obsahovat maximálně 132 znaků včetně čísla řádky a mezer.



P R I N T - pokračování příkazu

Nyní se vrátíme k vysvětlování příkazu PRINT. Zatím jsme se seznámili, jak můžeme psát konstanty a nyní si ukážeme výstup proměnných a některé funkce pro příkaz PRINT. Pro tisk (zobrazení na obrazovce displeje) proměnných pomocí příkazu PRINT, použijeme v seznamu identifikátory proměnných (numerických nebo řetězcových).

Příklad:

```
10 X = 4
20 Y = 2
30 PRINT X,Y
```

po spuštění programu se vytisknou čísla 4 a na další tabelační pozici číslo 2

Změníme-li příkaz na řádku 30 na:

```
30 PRINT "X="; "Y=";Y
```

pak se nám vytiskne X=4 a na další tabelační pozici Y=2. V uvozovkách jsou uvedeny řetězcové konstanty. Dále se vytisknou hodnoty proměnných (X a Y).

Doplníme-li další řádek s příkazem PRINT:

```
40 PRINT "X+Y=";X+Y,"X-Y=";X-Y
```

pak se po spuštění programu nám na jednom řádku vytisknou vstupní proměnné a na druhém řádku X+Y=5 a dále X-Y=2. V příkazu PRINT se provede vyhodnocení výrazů - součet a rozdíl proměnných.

Příklad:

```
10 A$ = "VYSLEDEK ="
20 B$ = "PROMENNA ="
30 X = 4: Y = 2
40 PRINT B$;X,B$;Y,A$;X*Y
```

po spuštění programu se zobrazí PROMENNA = 4 a na další tabelační pozici PROMENNA = 2 a na další tabelační pozici (na novém řádku od druhého sloupce) VYSLEDEK = 8.

Basic vypisuje čísla tak, že je zarovnává zleva. V mnohých případech potřebujeme zobrazovat čísla tak, aby byly zarovnané zprava (tj. aby desetinné tečky byly nad sebou). Nebo potřebujeme zobrazit číslo menší než 0.1 v normálním tvaru a ne v exponenciálním tvaru tak, jak to normálně Basic zobrazuje. Pro tento účel je v Basicu V3.0 klíčové slovo USING.

Formát:

```
PRINT USING "maska";seznam
```

Maska je uzavřena v uvozovkách a v masce jsou znaky "#" a znak "." pro určení polohy desetinné tečky ve výstupním zobrazení. Znaky "#" určují pozice pro výstupní číslo (vč. znaménka). Při provádění se znaky v masce nahrazují platnými číslicemi (levostranné nuly jsou nahrazeny znakem mezera). Číslo se zobrazuje na tolik desetinných míst, kolik je předepsáno v masce. V masce může být maximálně devět znaků "#". Je-li překročen rozsah masky výstupním číslem, pak se zobrazí celé výstupní číslo se znakem "%", který indikuje překročení vymezeného rozsahu pro číslo maskou.

Seznam může obsahovat proměnné,výrazy i numerické konstanty. V seznamu jsou jednotlivé prvky odděleny platnými oddělovači (čárka, středník).

Maska je od seznamu oddělena znakem středník. V příkazu PRINT může být použito vícekrát klíčové slovo USING s maskou. Maska platí buď do změny jinou maskou nebo ukončení příkazu.

Příklad:
Zobrazení rozdílu mezi normálním zobrazováním a příkazem s maskou.

```
10 PRINT 1
20 PRINT 10
30 PRINT 100
40 PRINT 1000
50 PRINT USING "####";1
60 PRINT USING "####";10
70 PRINT USING "####";100
80 PRINT USING "####";1000  po spuštění programu se zobrazí

1
10
100                    ... normální zobrazování - zarovnání
1000                       zleva
   1
  10
 100                   ... zobrazení pomocí USING - zarovnání zprava
1000
```

Poznámka:

Uvedený příklad lze v Basicu zapsat podstatně kratším a jednodušším způsobem pomocí cyklů, se kterými jsme se zatím neseznámili. Zde je důležité uvědomit si rozdíl v zobrazování numerických proměnných. Každé numerické číslo má pozici pro znaménko (kladné znaménko se nezobrazuje a je nahraženo mezerou) a číslo je rovněž ukončeno mezerou.

To platí i pro čísla zobrazovaná maskou (USING).

Příklad:

Zobrazení malého čísla (menší než 0.1)

```
10 X = 0.05
20 PRINT X, USING "#.##";X
```

... po spuštění programu se zobrazí

```
5E-02          0.05
```

... první číslo je v exponenciálním tvaru (číslo menší než 0.1), druhý tvar .05 je výstup maskou.

Příklad:

Překročení masky.

```
10 X=15.25
20 PRINT X, USING "#.##";X
15.25   %15.25
```

... první zobrazené číslo je bez masky a ve druhém čísle je výstup podle masky, která byla překročena. Je rezervováno pouze jedna pozice před desetinnou tečkou a výstupní číslo požaduje dvě pozice. Překročení je indikováno znakem "%".

Příklad pro změnu masky:

```
10 A = 123 : B = 1.23 : C = 0.123
20 PRINT USING "###.##"; A;B; USING "#.###";C
123.00   1.23   .123
```

... první dvě čísla jsou zobrazena podle první masky na dvě desetinná místa, poslední číslo podle druhé masky na tři desetinná místa.

Funkce pro příkaz PRINT

V Basicu jsou funkce pro formátování výstupního zobrazení. S prvními dvěmi funkcemi - tj. oddělovači jsme se již seznámili - čárka a středník. Zde bychom jenom připomněli, že čárka zobrazuje na další tabelační pozici a středník bezprostředně za předcházejícím výstupem. Pro formátování lze použít i prázdný seznam pro vynechání tabelační pozice. Příklad PRINT 1,2,3 zobrazí čísla 1 na první tabelační pozici, 2 na další a 3 na třetí. V případě zápisu PRINT 4,,5 se číslo 4 zobrazí na první tabelační pozici a číslo pět na třetí pozici. Pro druhou tabelační pozici je prázdný prvek.

Funkce TAB - tabulace

Funkce TAB se používá v příkazu PRINT pro určení pozice výstupu prvku seznamu.

Formát:

```
TAB (pozice)
```

pozice určuje číslo sloupce, od kterého se má začít zobrazovat proměnná (konstanta). Funkce se provede pouze tehdy, je-li pozice uvedená ve funkci TAB větší než je aktuální pozice.

Poznámka:

Funkce TAB neumožňuje návrat zpět na předcházející pozice. Tento příkaz byl do Basicu zabudován zejména pro tisk na tiskárnách (a ty se obvykle neumí vrátit zpět). Pro pohyb kurzoru na obrazovce displeje jsou v Basicu příkazy, které nám umožní nastavit kurzor na libovolnou pozici. Pozor pak při použití příkazu PRINT s funkcí TAB, kdy se neuvažuje s přemístěním kurzoru pomocí těchto speciálních příkazů.

Příklad: Chceme vytisknout text od sloupce 5 a 25.

```
10 V = 10 : S =33
20 PRINT TAB(5) "VYSKA";TAB(25) "SUMA"
30 PRINT TAB(5) V;TAB(25) S
```

```
VYSKA       SUMA      ... řádek 20 nám vytiskne hlavičku
                          (od pozice 5 a 25)
 10          33       ... řádek 30 vypíše hodnoty do
                          sloupců
```

Za funkcí TAB se nemusí používat oddělovače (čárka nebo středník). Proměnné před příkazem TAB by měly být odděleny středníkem.

Funkce SPC - výstup zadaného počtu mezer

Funkce SPC se používá v příkazu PRINT k tisku (zobrazení) zadaného počtu mezer. Mezery se tisknou od aktuální pozice pro tisk.

Formát:

```
SPC(počet)
```

počet - udává požadovaný počet mezer mezi prvky seznamu příkazu PRINT, mezi kterými je uvedena funkce SPC. Pro příkaz SPC platí stejné zásady jako pro TAB (tj. vztah ke kurzoru a jeho změny pomocí speciálních příkazů).

Funkce TAB a SPC se v seznamu nemusí oddělovat oddělovači. Pokud použijeme oddělovačů, pak si musíme uvědomit, že čárka má při výkonu přednost, tj. prvně se provede skok na následující tabelační pozici a pak se provádí požadovaná funkce. Tak např. použijeme-li čárku před SPC, provede se nejdříve skok na tabelační pozici a pak teprve vystoupí požadovaný počet mezer.

Příklady:

Abyste správně pochopili vztah oddělovačů a funkci SPC, vyzkoušejte si jednoduchý příklad.

```
10 PRINT "+";"-"
20 PRINT "+","-"
30 PRINT "+";SPC(3);"-"
40 PRINT "+",SPC(3);"-"
50 PRINT "+" SPC(3) "-"
60 PRINT "+" SPC(3),"-"
```

```
+-          ... příkaz PRINT na řádku 10, znaky jsou od-
                děleny ";" následují za sebou

+           - ... řádek 20,oddělovač "," - následující ta-
                belační pozice

+  -          ... řádek 30, funkce SPC(3), prvky seznamu
                odděleny ";"

+           -  ... řádek 40, nejdříve se provede tabelační
                pozice a pak funkce

+  -           ... řádek 50 - bez oddělovačů (implicitní
                oddělovače ";")

+           -  ... řádek 60 nejdříve se provede funkce a
                pak posun na další tabelační pozici.
                U této kombinace pozor, kdyby funkce
                SPC překročila následující tabelační
                pozici, pak oddělovač "," nastaví dal-
                ší tabelační pozici.
```

Funkce TAB a SPC lze použít pouze v příkazech PRINT, kdybyste je chtěli použít v jiných případech, interpret Vám nahlásí syntaktickou chybu.

Funkce POS - aktuální pozice ukazatele v příkazu PRINT

Funkce POS nám předá číslo sloupce, na který ukazuje ukazatel v příkazu PRINT (tj. číslo sloupce - pozice, na které se bude tisknout nebo zobrazovat). Je to vlastně první volná poziće na řádku. Tato funkce se může použít samostatně v příkazech jazyka Basic i v příkazu PRINT. Použití v příkazu PRINT nemá význam, protože nám vytiskne sice číslo pozice, ale po provedení funkce POS je ukazatel nastaven na novou volnou pozici (tj. za číslem, které bylo zobrazeno funkcí POS).

Tuto funkci používáme pro text konce řádku nebo chceme-li zjistit, zda vstupní text lze ještě zobrazit ve vymezeném prostoru.

Formát:

```
POS
```

Funkce POS nemá žádné parametry.

Příklad:

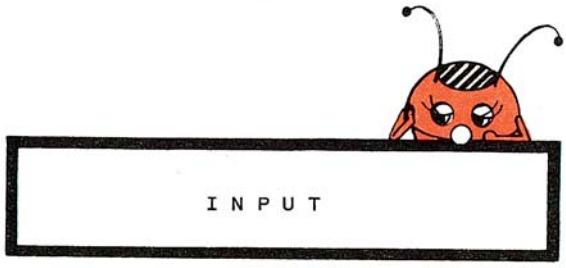
```
10 PRINT POS;POS,POS
 1   4          14
```

První funkce POS zobrazí číslo 1, tj. první volná pozice pro tisk na začátku příkazu PRINT, druhá funkce POS

zobrazí číslo 4, tj. při zobrazení čísla "1" se použily tři sloupce, číslo 1 předchází a ukončuje znak mezera a tudíž další volná pozice je 4 sloupec. Druhý příkaz POS byl ukončen čárkou a tudíž další tisk se provádí na tabelační pozici "14", která zobrazuje třetí příkaz POS.

Poznámka:
Obvykle se funkce POS využívá v podmíněných příkazech větvení programu IF, THEN, ELSE, které jsme dosud neprobírali a až se seznámíte s příkazy pro řízení programu, můžete se vrátit k funkci POS. Jistě sami velmi rychle najdete možnosti využití této funkce.

Příkaz PRINT nyní opustíme a znovu se k němu vrátíme až si budeme vysvětlovat práci se soubory. Pak si ukážeme, jak se příkaz PRINT používá pro tisk znaků na tiskárně a zápis znaků na vnější paměť - kazetový magnetofon. Zatím umíme používat příkaz PRINT pro zobrazování na obrazovce displeje (televizního přijímače).



Příkaz INPUT slouží pro vstup dat z vnějších zařízení do počítače. V této části se seznámíme s použitím příkazu pro vstup znaků z klávesnice mikropočítače.

Formát:

```
INPUT seznam
```

```
INPUT "text"; seznam
```

Seznam obsahuje jména proměnných, do kterých se uloží data ze vstupního zařízení (z klávesnice). Text je možno zobrazit na obrazovce displeje.

Při použití příkazu INPUT se na obrazovce displeje zobrazí znak "?" (otazník), který signalizuje, že počítač očekává vstup dat z klávesnice. V seznamu mohou být numerické i řetězcové proměnné. Pokud zadáme méně dat, než je

uvedeno v seznamu, na obrazovce se opět zobrazí znak "?", že příkaz dále očekává data. Zadáme-li více dat, než je počet prvků v seznamu, pak interpret nahlásí zprávu EXTRA DATA IGNORED, která znamená, že nadbytečná data jsou ignorována (dále se s nimi neuvažuje). Basic kontroluje i správnost vstupu podle typu prvků v seznamu. Zadáme-li do numerické proměnné jiný znak než číslice a nebo desetinnou tečku, pak nahlásí chybu INPUT ERROR a vstupní data ignoruje a požaduje nový vstup dat. Vstupní data lze oddělit od sebe pouze znakem "," čárka, mezery nejsou v Basicu V5.0 platným oddělovačem. Řetězcové proměnné lze zadávat bez uvozovek a pak se zavádí znaky do oddělovače - tj. čárky. Pokud chceme zavést řetězec, který obsahuje čárku, pak musíme řetězec uzavřít do uvozovek. Jako oddělovače lze použít i znak "↵" (CR), který ukončuje vstupní řádku. Pokud není vyčerpán vstupní seznam, opakuje Basic požadavek na vstup znaků. Na rozdíl od verze V4.0 (použité u systému SAPI 1), lze zadávat i prázdné řetězce a numerické údaje. U vstupu numerických dat se po stisknutí "CR" dosadí číslo "0" a u řetězce prázdný řetězec (s délkou řetězce 0). Levostranné mezery v řetězci se vynechávají, pokud chceme levostranné mezery zachovat, musíme uzavřít řetězec do uvozovek.

Poznámka: Příkaz INPUT nelze použít v dialogovém režimu (tj. bez čísel řádků).

Příklad:

```
10 INPUT A,B,A$,B$    ... vstup čtyř proměnných
20 PRINT A,B,A$,B$    ... zobrazení proměnných
```

Po spuštění tohoto jednoduchého programu si můžeme vyzkoušet všechny možné kombinace pro zadávání vstupních dat. První dvě proměnné jsou numerické a druhé dvě proměnné jsou řetězce.

Při používání příkazu INPUT zjistíte, že byste potřebovali vědět, jakou vstupní proměnnou požaduje Váš program. To lze řešit například použitím příkazu PRINT, před příkazem INPUT.

Příklad: Požadujeme vstup dvou numerických proměnných A, B a chceme, aby se před vstupem zobrazilo označení proměnné. Můžeme to řešit napříkald takto:

```
10 PRINT "A=";        ... vytiskneme A=, musíme použít
20 INPUT A                znak ";" aby vstuppokračoval
                          za vytištěným textem (nápovědou)
30 PRINT "B=";
40 INPUT B            ... totéž pro další vstup
50 PRINT "A=";A,"B=";B ... zobrazení na displeji vstupních dat s označením
```

Při spuštění tohoto programu se na obrazovce zobrazí nápověda pro vstup čísla A a po zadání dat se zobrazí nápověda, že se požaduje vstup čísla B. Po zadání se

zobrazí obě čísla pomocí příkazu PRINT na obrazovce.

Jistě jste si všimli, že příkaz INPUT měl uvedeny dva formáty. Zatím jsem se zabývali prvním formátem. Nyní si vysvětlíme druhý formát. Ten je právě určen pro předcházející případ, tj. když potřebujeme indikovat nápovědný text na obrazovce. Formát druhého příkazu INPUT je:

```
INPUT "text"; seznam
```

Tento formát v sobě slučuje příkaz PRINT pro text a INPUT pro seznam. Takže výše uvedený příklad můžeme zapsat:

```
10 INPUT "A=";A     ... zobrazí se A= a čeká na vstup dat
20 INPUT "B=";B     ... totéž pro B
30 PRINT "A=";A,"B=";B  ... tisk vstupních dat
```

Příkaz INPUT v Basicu V5.0 však umožňuje použití více textů v příkazu INPUT (pozor,ne všechny Basicy umožňují vícenásobné použití textů!!), a tak se náš příklad ještě více zjednoduší:

```
10 INPUT "A=";A,"B=";B  ... zobrazení nápovědy a vstup
                            proměnných
20 PRINT "A=";A,"B=";B  ... zobrazení vstupních dat
```

V tomto případě se nám zobrazí A=? a očekává se vstup čísla, které ukončíme znakem CR, pak se nám zobrazí B=? a zadáme druhé číslo.

Pochopitelně můžeme místo jednoduché proměnné použít seznam pro více prvků (např. stejná skupina dat) a pak při vstupu oddělujeme jednotlivé proměnné čárkami (nebo i CR). Příklad: Výše uvedený příklad můžeme zapsat i tímto způsobem:

```
10 INPUT "VSTUP A,B=";A,B
```

Pak víme, že počítač požaduje dvě čísla A a B, které můžeme zadat najednou. Meze pro využití příkazu INPUT s textem se nekladou. Pokud byste za textem použili čárku, pak se text a vstup čísla přesune na další tabelační pozici.

RUN

Povel RUN spustí program v Basicu, který se začne provádět. Pod pojmem program chápeme logickou posloupnost příkazů, která má splnit určitou funkci. V Basicu jsou jako program prováděny příkazové řádky s čísly řádek. Basic provádí příkazové řádky vzestupně podle čísel řádků, pokud neprovádí příkaz, který přerušuje normální posloupnost příkazů (např. skoky, přepínače atd.) a nebo nenajde příkazy pro ukončení programu (END,STOP) a nebo že vyčerpal všechny příkazové řádky.

Formát:

RUN

RUN číslo řádku

Povel RUN má dva formáty, bez čísla řádku a s číslem řádku. Není-li uvedeno číslo řádku, pak začne provádět program od nejnižšího čísla řádku. Je-li číslo řádku uvedeno, začne program provádět od uvedeného čísla řádku.

E N D

Příkaz END ukončuje hlavní program a tvoří fyzický konec hlavního programu a při běhu programu ukončuje provádění programu. Řízení se vrací interpretu jazyka Basic, který ohlásí svoji připravenost zobrazením textu "READY".
Příkaz END se používá v programu pouze jedenkrát.

S T O P

Příkaz STOP ukončuje provádění programu (podobně jako příkaz END). Příkaz STOP může být v programu použit vícekrát pro ukončení programu v různých větvích. Aby uživatel věděl,

který příkaz STOP ukončil provádění programu, vypíše se na obrazovce text

BREAK IN LINE číslo řádku ... kde číslo řádku je číslo řádku s příkazem STOP

Tento příkaz s výhodou používáme pro ladění programu a potřebujeme vědět, ve které větvi programu se nacházíme. Po příkazu STOP můžeme pokračovat povelem CONT (continue - pokračování). Formát obou příkazů je jednoduchý bez parametrů.

Formát:

END

STOP

C O N T

Povel CONT je povelem pro pokračování provádění programu, který byl zastaven příkazem STOP nebo přerušen stisknutím tlačítka CTRL C z klávesnice počítače. Byl-li program zastaven příkazem STOP, pak program pokračuje následujícím příkazem za příkazem pro zastavení. Pozor, byl-li program ukončen příkazem END, nelze pokračovat v programu příkazem CONT. Byl-li program přerušen tlačítkem CTRL C, pak program pokračuje od místa přerušení.

Poznámka: Je-li program přerušen v cyklu, pak u příkazu CONT dojde k chybě (nesprávně nastavená řídící proměnná).

Formát:

CONT

Povel nemá žádné parametry.

N E W

Povel NEW se používá pro vymazání programu a všech proměnných z paměti počítače . Povel používáme před zadáváním nového programu. Provede se i výmaz obrazovky a ohlášení se Basicu s celou hlavičkou.

Formát:

NEW

Povel NEW nemá žádné parametry.

Rádi bychom Vás upozornili na častou chybu začátečníků při vstupu dat, kterou dělají tím, že zapomínají oddělovat vstupní data čárkou a oddělují je mezerami.

Příklad:

```
10 INPUT "A,B,C=";A,B,C   ... vstup tří numerických čísel
20 PRINT A,B,C            ... zobrazení vstupů
```

Po spuštění příkazem RUN se zobrazí:

```
A,B,C=? 1 2 3 (CR)   ... zadali jsme čísla 1,2,3 oddělené
?4 ?5                    mezerami a ukončili řádek znakem
                         CR. Na obrazovce se znovu objeví
                         znak ?, který signalizuje, že
                         Basic očekává vstup znaků. Zadáme
                         znak 4 a opět se zobrazí "?"
                         a zadáme číslo 5

123    4    5        ... příkaz PRINT zobrazí čísla
                         123,4,5. Číslo 123 vzniklo z čísel
                         1,2,3 když se vynechaly mezery,
                         které Basic ignoruje.
```

Nyní INPUT opustíme a opět se k němu vrátíme, až si budeme vysvětlovat práci se soubory, stejně jako příkaz PRINT.

ŘÍZENÍ PROGRAMU

Zatím jsme se seznámili s výstupem znaků na obrazovku displeje, jak vstupovat s daty z klávesnice a jak přiřadit proměnným hodnoty. To je ovšem pro program velmi málo a my potřebujeme v programu testovat určité podmínky a je-li tato podmínka splněna, provést určitou činnost a pokud podmínka splněna není, pak provést zcela jinou činnost. Zatím naše znalosti na větvení nestačí a "umíme" sestavit pouze lineární program. V mnohých případech potřebujeme určitou činnost opakovat, dokud není splněna podmínka pro ukončení opakování činnosti. Takovému opakování říkáme cykly. Dále potřebujeme přerušit lineární posloupnost programu podle čísel řádků a skočit na jiné číslo řádku a tam pokračovat v provádění programu. Při programování často zjistíte, že část programu potřebujete opakovat, podle toho, ve kterém místě programu jste nebo podle hodnot proměnných. V tomto případě by bylo pracné a paměťově náročné neustále opakovat stejné posloupnosti příkazů. Proto Basic umožňuje vyvolávat určené sekvence příkazových řádků a po provedení se vrátit zpět do místa vyvolání. Této sekvenci příkazů pak

říkáme podprogram. Dále se seznámíme s přepínači, které nám umožňují skoky na příkazové řádky podle hodnoty řídící proměnné.

Příkazy cyklů

Příkazy cyklů umožňují opakovat posloupnost příkazů uzavřenou mezi klíčovými slovy cyklů. V Basicu V5.0 jsou tři typy cyklů. První FOR - NEXT je standardním typem cyklu ve všech modifikacích jazyka Basic. Další dva typy WHILE - WEND a REPEAT - UNTIL jsou pouze ve verzi V5.0 pro mikropočítač ONDRA a tyto konstrukce jsou převzaty z jazyka Pascal.

Příkaz cyklu FOR - NEXT

Příkazová konstrukce cyklu FOR - NEXT umožňuje provádění smyčky pro zadaný počet. Základem je tzv. řídící proměnná cyklu. Při vstupu do cyklu se řídící proměnné přiřadí počáteční hodnota, která je při každém průchodu cyklem zvýšená (resp. snížená) o hodnotu kroku cyklu. Po zvýšení řídící proměnné se provádí test, byla-li překročena zadaná horní mez. Pokud nebyla, pak se pokračuje v provádění cyklu, byla-li překročena, pak se provádění

smyčky ukončí a program pokračuje následujícím příkazem za ukončovacím příkazem cyklu NEXT.

Formát:

```
FOR řídící proměnná = poč. hod. TO horní mez
```

```
FOR řídící proměnná = poč. hod. TO horní mez STEP krok
```

Uvedená posloupnost tvoří vstupní bod cyklu. Do cyklu lze vstupovat pouze přes vstupní bod, a nelze vstupovat přímo do těla cyklu. Řídící proměnná je libovolná numerická proměnná, které se přiřadí počáteční hodnota. Horní mez určuje, až po kterou hodnotu řídící proměnné se má cyklus opakovat. Krok určuje přírůstek řídící proměnné při jednotlivých průchodech cyklem. Klíčová slova pro vstupní bod cyklu jsou FOR TO STEP, které můžeme česky vysvětlit takto: Pro řídící proměnnou s přiřazenou počáteční hodnotou až do horní meze s krokem cyklu "krok" dělej. Klíčové slovo STEP a hodnota kroku se nemusí uvádět. V tomto případě se uvažuje krok automaticky +1. Pokud požadujeme jiný přírůstek, pak klíčové slovo STEP musíme uvést.
Vzhledem k tomu, že řídící proměnná se porovnává s horní mezí až po průchodu cyklem, tělo cyklu se provede alespoň jednou.

Cyklus se normálně provádí vzestupně, tj. s kladným krokem. V některých případech potřebujeme provádět cyklus opačně, tj. sestupně. Příkazy cyklu FOR - NEXT umožňují i tento způsob práce. V tomto případě je počáteční hodnota vyšší než koncová (horní) mez a krok uvedeme záporný.

Aby Basic věděl, co považujeme za tělo cyklu, které se má opakovat, je pro označení konce cyklu rezervováno klíčové slovo NEXT, které označuje konec cyklu. Výše popsaná činnost zvyšování řídicí proměnné a testování horní meze se provádí, jakmile se narazí na klíčové slovo NEXT.

Formát:

```
NEXT
NEXT seznam
```

Seznam obsahuje řídicí proměnnou (resp. řídicí proměnné) cyklu(ů). U jednoduchého cyklu nemusíme řídicí proměnnou uvádět. Je-li více cyklů vnořených do sebe, je nutno uvádět řídicí proměnnou. Končí-li více cyklů ve společném bodě, pak je nutno uvést seznam řídicích proměnných, uspořádaných od vnitřních cyklů k vnějším (nejblíže ke klíčovému slovu NEXT musí být nejvnitřnější cyklus).

Znázorníme si cyklus:

```
10 FOR I=1 TO 5        ... vstupní bod cyklu, řídicí proměnná I má počáteční hodnotu 1,
        .                  horní mez je 5 a implicitní krok
        .                  má hodnotu +1
        .
  tělo cyklu
20 PRINT I             ... pro ilustraci tělo cyklu u nás představuje příkaz PRINT pro řídicí proměnnou I.
30 NEXT I              ... ukončení cyklu
40 PRINT "KONEC I=";I  ... příkaz za cyklem, který nám signalizuje ukončení cyklu.
```

Po spuštění se nám zobrazí posloupnost

```
 1
 2
 3                     ... tisk řídicí proměnné při jednotlivých průchodech cyklem
 4
 5
KONEC I=6              ...příkaz provedený po průchodu cyklem (řídicí proměnná má již větší hodnotu než je horní mez cyklu
```

Znázorníme si jednoduchý příklad pro vkládání cyklů do sebe, například zobrazení malé násobilky od 1 do 10.

```
10 FOR I=1 TO 10
20 PRINT "NASOBKY CISLA ";I
30 FOR J=1 TO 10
40 PRINT I*J
50 NEXT J,I
```

U tohoto příkladu si povšimněte, že jsou použity dva cykly.

Vnější pro proměnnou I a vnitřní pro proměnnou J. Tělo vnějšího cyklu tvoří příkazy 20, 30, 40 a tělo vnitřního cyklu tvoří příkazy 40. Oba cykly jsou ukončeny ve společném výstupním bodě, tj. řádek 50, a proto musí být uveden seznam řídících proměnných a prvně se uvádí vnitřní, tj. řídící proměnná "J" a pak vnější, tj. "I".

Při vkládání cyklů platí zásada, že cykly musí být vnořeny. Vnitřní cyklus nesmí mít koncový bod cyklu za koncovým bodem vnějšího cyklu.

Příklad:

```
10 FOR I=1 TO 10
 .
 .
50 FOR J=2 TO 6
 .
 .
100 FOR K=1 TO 3
 .
 .
150 NEXT K,J
 .
 .
200 NEXT I
```

DOBŘE

```
10 FOR I=1 TO 10
 .
 .
50 FOR J=2 TO 6
 .
 .
100 FOR K=1 TO 3
 .
 ..
150 NEXT I,J
 .
 .
200 NEXT K
```

CHYBNĚ

U příkladu Dobře jsou cykly správně ukončeny (vnitřní dříve než vnější). Vnější cyklus má řídící proměnnou I,

vnitřní J a nejvnitřnější K. U příkladu Chybně je nejdříve ukončen vnější cyklus, pak vnitřní a nejvnitřnější cyklus je ukončen jako poslední.

Poznámka:

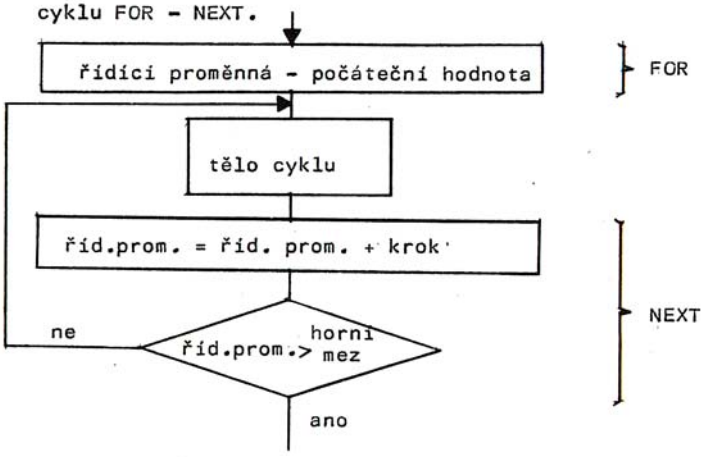
Počáteční hodnota, horní mez a krok cyklu nemusí být konstantní, ale mohou se použít i proměnné, které si nastavíme při chodu programu před vstupem do cyklu.

Např.

```
10 INPUT "POC.HOD,HORNI MEZ,KROK=";A,B,C
20 FOR J=A TO B STEP C
          .
          .
        tělo cyklu
          .
          .
100 NEXT I
```

V některých případech potřebujeme ukončit cyklus dříve než jsme vyčerpali všechny možnosti řídící proměnné (např. je splněna druhá **podmínka** a nepotřebujeme dále provádět smyčku). V tomto případě doporučujeme zvětšit řídící proměnnou nad horní mez a provést skok na výstupní bod cyklu, tj. na příkaz NEXT. Pokud byste použili nepodmíněný skok, může dojít k destrukci programu. Do cyklu se vstupuje přes vstupní bod (tj. přes FOR) a z cyklu vystupujeme zásadně přes výstupní bod, tj. přes NEXT.

Pro lepší porozumění si uvedeme schéma konstrukce příkazu cyklu FOR - NEXT.



Příkaz cyklu REPEAT - UNTIL

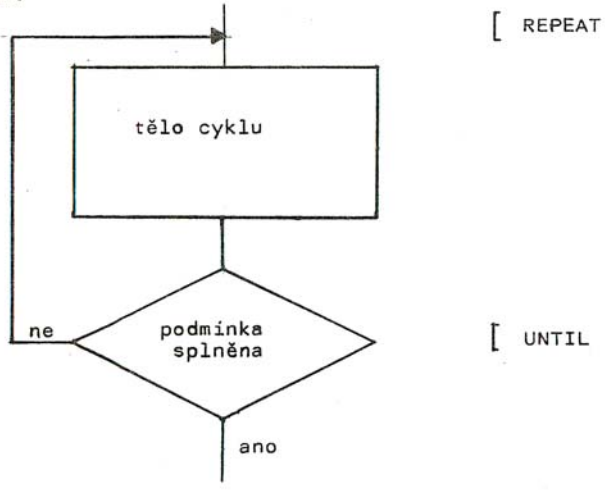
Příkaz cyklu REPEAT - UNTIL není v množině příkazů Basicu a pro BASIC EXP V5.0 byl doplněn. Tato struktura je převzata z jazyka Pascal. Pokud chcete přenášet svoje programy na jiné mikropočítače, které tuto **konstrukci** nemají, pak tento příkaz cyklu nepoužívejte.

Formát:

```
REPEAT
tělo cyklu
UNTIL podmínka
```

Klíčové slovo REPEAT (opakuj) tvoří vstupní bod do cyklu. Klíčové slovo UNTIL (dokud) ukončuje cyklus a vyhodnocuje podmínku uvedenou za klíčovým slovem. Cyklus se provádí tak dlouho , dokud není splněna podmínka. Je-li podmínka splněna, pak se pokračuje následujícím příkazem za UNTIL. Povšimněte si, že zde řídicí proměnná je definována až u slova UNTIL, ale na rozdíl od FOR se automaticky neprovádí zvyšování řídicí proměnné. Zvyšování řídicí proměnné si musí zajistit uživatel v těle cyklu. Podmínka se vyhodnocuje až po průchodu cyklem a z toho vyplývá, že tělo cyklu se u tohoto příkazu provede alespoň jedenkrát.

Pro názornost si opět uvedeme blokové schéma konstrukce příkazu:



Pozor! Zapomeneme-li nastavovat řídící proměnnou v těle cyklu, pak se cyklus bude provádět do nekonečna (nekonečná smyčka) a z cyklu se nedostanete, jedině přerušením programu.

Tato konstrukce cyklu je vhodná, není-li pravidelný krok pro cyklus nebo závisí-li provádění cyklu na vnějších podmínkách.

Příklad:

V programu si zadáme znak použitý na klávesnici a chceme, aby se našel znak, který je uveden v programu a program počítá vaše pokusy o nalezení znaku.

```
10 B$="!" : I =0
20 REPEAT
30 INPUT "ZADEJ ZNAK";A$
40 I = I +1
50 UNTIL A$ = B$
60 PRINT "ZNAK";B$;" JSI NASEL NA ";I;"POKUSU!"
```

Řádek 10 nastavuje počáteční podmínky, tj. hledaný znak a vynuluje počítadlo pokusů. Řádky 20 až 50 jsou cyklus. Vstupní bod cyklu je řádek 20, řádek 30 naplní řídící proměnnou znakem z klávesnice, řádek 40 zvětší počítadlo pokusů o další pokus a řádek 50 je výstupním bodem cyklu, je-li splněna podmínka, tj. je-li vstupní znak z klávesni-

ce roven zadanému znaku. Pokud se znaky nerovnají, cyklus se opakuje. Při nalezení správného znaku se ukončí cyklus a vypíše se počet pokusů, kterých bylo zapotřebí pro nalezení správného znaku.

Příkaz cyklu WHILE - DO - WEND

Příkaz cyklu konstrukce WHILE-DO-WEND není v množině příkazů běžných jazyků Basic, i když se s touto konstrukcí v mnohých implementacích můžete setkat. V Basicu V5.0 je tato konstrukce opět převzata z jazyka Pascal, podobně jako příkaz cyklu REPEAT-UNTIL. Obě konstrukce Vám zjednodušují programování smyček, které nelze vytvořit pomocí konstrukce FOR-NEXT. V jiných implementacích tyto konstrukce obcházíme pomocí podmíněných a nepodmíněných skoků. Takové programy jsou méně přehledné a srozumitelné než programy, které využívají těchto nových konstrukcí cyklů.

Formát:

```
WHILE podmínka DO
      .
      .
tělo cyklu
      .
      .
WEND
```

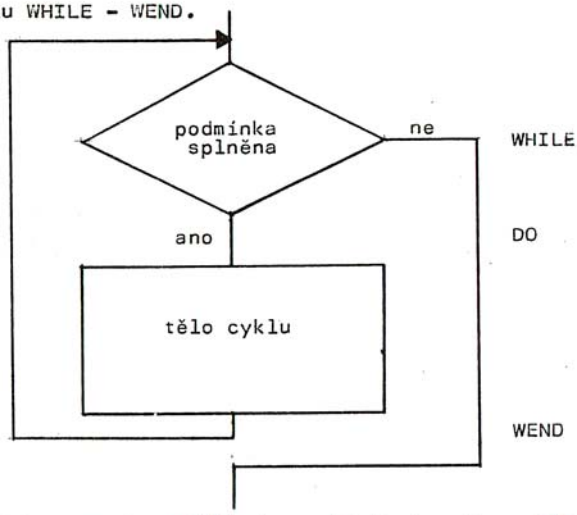
Klíčové slovo WHILE (dokud) tvoří vstupní bod cyklu. Při provádění této konstrukce se vyhodnotí podmínka a je-li splněna, provede se tělo cyklu, které je uvedeno klíčovým slovem DO. Aby Basic mohl rozeznat konec těla cyklu, musí být označen konec cyklu klíčovým slovem WEND (WHILE-END). Nebylo možné použít slovo END (tak jako v Pascalu), protože příkaz END označuje fyzický konec hlavního programu.

Všimněte si, že řídící proměnná, stejně jako v příkazu REPEAT je definována v podmínce, ale příkaz cyklu tuto řídící proměnnou automaticky nezvyšuje. To si musí uživatel zajistit sám v těle cyklu. Pokud nebudeme měnit řídící proměnnou, můžeme vytvořit nekonečný cyklus.

Podmínka se vyhodnocuje před vstupem do cyklu a není-li splněna, tělo cyklu se neprovede a program pokračuje příkazem za koncovým bodem cyklu WEND. Z toho vyplývá, že u cyklu WHILE-END se tělo cyklu nemusí provést ani jednou.

Tento typ cyklu je vhodný v případech, že potřebujeme čekat na nějakou vnější událost, nebo v případech, kdy se krok cyklu mění podle potřeby a v případech, kdy se tělo cyklu nemá provést ani jednou, není-li podmínka splněna (ostatní konstrukce cyklů tělo cyklu provedou minimálně jedenkrát).

Pomocí blokového schématu si znázorníme konstrukci příkazu cyklu WHILE - WEND.



Použijeme stejný příklad pro hledání znaku z klávesnice, který jsme použili u cyklu REPEAT. Příklad pouze vylepšíme zadáním hledané klávesy.

```
10 INPUT B$: I=0
20 CLS                    ! výmaz obrazovky
30 WHILE A$<>B$ DO
40 INPUT A$
50 I=I+1
60 WEND
70 PRINT "ZNAK";B$;" JSI NASEL NA ";I;"POKUSU!"
```

Řádek 10 nastaví hledaný znak (zadaný z klávesnice) a vynuluje počítadlo pokusů. Řádek 20 je příkaz pro výmaz

obrazovky a seznámíme se s ním později. Řádky 30 až 60 jsou příkazy cyklu. Cyklus se ukončuje vnější podmínkou, že byl z klávesnice zadán správný znak.
Řádek 70 se tiskne po ukončení cyklu.

Podmíněné větvení programu IF THEN ELSE

Příkaz IF umožňuje větvení programů nebo provedení posloupnosti příkazu v závislosti na podmínce, která se určuje pomocí relačních výrazů.

V běžném životě se neustále rozhodujeme, co budeme dělat, když nastane nějaký jev. Například, Když nebude pršet, pak půjdu ven, jinak zůstanu doma. Když nebude pršet - je podmínka a je-li splněna (tj. že neprší), půjdu ven a není-li splněna (tj. že venku prší), zůstávám doma.

Podobně se rozhodujeme i při programování. Některou posloupnost příkazů potřebujeme provést pouze tehdy, je-li splněna určitá podmínka a jinou posloupnost, není-li podmínka splněna. Podmínka může být složitý logický výraz, jehož hodnota je "pravda" nebo "nepravda". Podmínka je výraz s relačními operátory (jako rovná se, menší, větší atd.).

V Basicu V5.0 jsou dva typy příkazů IF - jednoduché a složené. Jednoduché příkazy obsahují pouze příkaz THEN nebo GOTO. Složené příkazy se větví pro obě možnosti vyhodnocení podmínky THEN a ELSE. Složené příkazy pro větvení nejsou implementovány ve všech verzích jazyka Basic. Kromě toho Basic V5.0 umožňuje vyhodnocování více příkazů za klíčovými slovy THEN a ELSE.

Formát:

```
IF podmínka GOTO čís. řád.
IF podmínka THEN čís. řád.
IF podmínka THEN příkaz
```

Nejdříve si probereme jednoduché příkazy pro větvení. Podmínka je relační výraz, který může nabývat pouze dvou hodnot - pravda a nepravda. Je-li hodnota "pravda", pak se provede skok na číslo řádku nebo příkaz (resp. posloupnost příkazů). Je-li hodnota "nepravda", pak se pokračuje na následujícím řádku za příkazem IF. To znamená, že můžeme vyhodnotit pouze posloupnost příkazů, která je uvedena na jednom příkazovém řádku. První dva typy jsou funkčně shodné a provedou skok na uvedené číslo řádku.

Příklad:

```
100 IF A=1 GOTO 150
```
Je-li proměnná A rovna 1, pak se provede skok na řádek s číslem 150.

```
110 IF A=2 THEN 200
```
Podobně jako předcházející případ, je-li A rovno 2, pak se provede skok na řádek 200.

Některé implementace nepřipouští první tvar s klíčovými slovy GOTO a musí se použít vždy klíčové slovo THEN a pak příkaz GOTO.

```
100 IF A=1 THEN GOTO 150
```
Stejné jako první případ, pouze s tím rozdílem, že za THEN je použit příkaz nepodmíněného skoku.

```
130 IF A=1 THEN SUMA=SUMA+1: A=2
```
Ukázka použití příkazů za klíčovým slovem THEN.

Uvedeme si formát složeného příkazu IF pro větvení, je-li podmínka pravda a nepravda.

Formát:

IF podmínka THEN čís.řád. ELSE čís.řád.

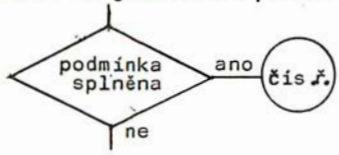
IF podmínka THEN příkaz ELSE příkaz

Klíčová slova IF,THEN,ELSE si můžeme jednoduše přeložit do češtiny jako jestliže, pak, jinak a zapsaný

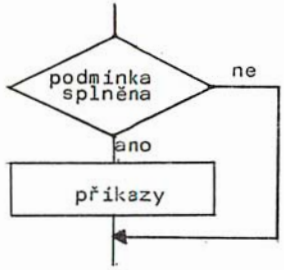
příkaz přečteme jestliže je splněna podmínka, pak proveď příkazy za THEN, jinak proveď příkazy za ELSE.

Opět můžeme použít buď číslo řádku nebo příkaz, resp. posloupnost příkazů. Je tady určité omezení, a to že příkazy musí končit na jednom příkazovém řádku, kde je příkaz IF a druhé omezení, že se v příkazech nesmí použít další podmíněný příkaz IF.

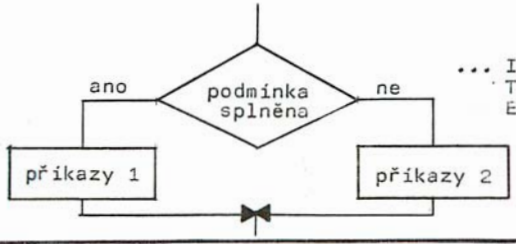
Znázorníme si jednotlivé podmíněné příkazy:



...typ IF pod.GOTO č.ř.
IF pod.THEN č.ř.



...typ IF pod. THEN
příkazy



... IF pod.
THEN p1
ELSE p2

Podmínky mohou pracovat jak s numerickými, tak i s řetězcovými proměnnými. Například jako podmínka může být A<=5 nebo B<>C, A$="JMENO" či A$=B$. To jsou jednoduché relační výrazy, kdy proměnné srovnáváme s konstantami nebo proměnnými. Podmínky mohou být i složené relační výrazy (pomocí logických operátorů). A=1 OR B=2, výraz bude pravdivý, bude-li hodnota proměnné A rovna jedné nebo hodnota proměnné B rovna dvěma. Pochopitelně můžeme použít i další logické operátory, např. AND, NOT, XOR. Lze použít i aritmetické operátory, které se vyhodnocují jako první, pak se vyhodnocují relační a nakonec logické operátory.

Příklad:

(A<5*I) OR (B=3) AND (C=A-1) nejdříve se provedou výrazy v závorkách, tj. nastaví se logické hodnoty pro výrazy. Je-li hodnota A menší než je pětinásobek hodnoty I, pak závorka má hodnotu pravda, druhá a třetí závorka jsou pravda, je-li B=3 a C rovno hodnotě proměnné A zmenšené o jedničku. A nyní se složený výraz vyhodnocuje podle pravidel Boolovské algebry, tj. nejdříve se vyhodnotí výraz AND a potom OR. Aby výraz AND byl pravdivý, musí být pravdivé oba operandy a u operátoru OR musí být pravdivý alespoň jeden operand. Takže složený výraz bude

pravdivý, bude-li A menší než 5*I a nebo bude-li B=3 a současně C bude rovno proměnné A zmenšené o 1.

V mnoha případech omezení uvedená u příkazů IF, tj. že příkazy mohou být pouze na řádku, nebo že nelze vnořovat do sebe podmíněné výrazy, jsou na závadu a vedou ke složitějším a méně přehledným programům v Basicu. Právě malou přehlednost vytýkají odpůrci tomuto jazyku. Proto byl Basic V5.0 doplněn o příkazy bloků z jiných vyšších jazyků (Pascal, Algol, ADA a další). Na blok pak lze pohlížet jako na jeden příkaz. To nám umožní psát víceřádkové příkazy a vnořovat podmíněné příkazy do sebe. Pro blok jsou rezervována dvě klíčová slova BEG a BEND (v jazyku Pascal jsou to BEGIN a END). Příkaz END nelze v Basicu použít, protože je rezervován pro fyzický konec hlavního programu.

Formát:

```
BEG

BEND
```

Klíčové slovo BEG uvádí začátek bloku a na příkazy uvedené za tímto příkazem lze pohlížet jako na jeden příkaz. Klíčové slovo BEND ukončuje blok příkazů.

Doporučujeme uvádět klíčová slova na zvláštních programových řádcích, aby byl přehled o použití bloků. Každému otevíracímu příkazu BEG musí odpovídat uzavírací příkaz BEND.

Příklad použití:

```
10 INPUT A
20 IF A=1 THEN
30 BEG
40 PRINT "++++++++++++++++"
50 PRINT "CISLO =1"
60 PRINT "++++++++++++++++"
70 BEND
80 ELSE
90 BEG
100 PRINT "CISLO JE RUZNE OD 1"
110 BEND
```

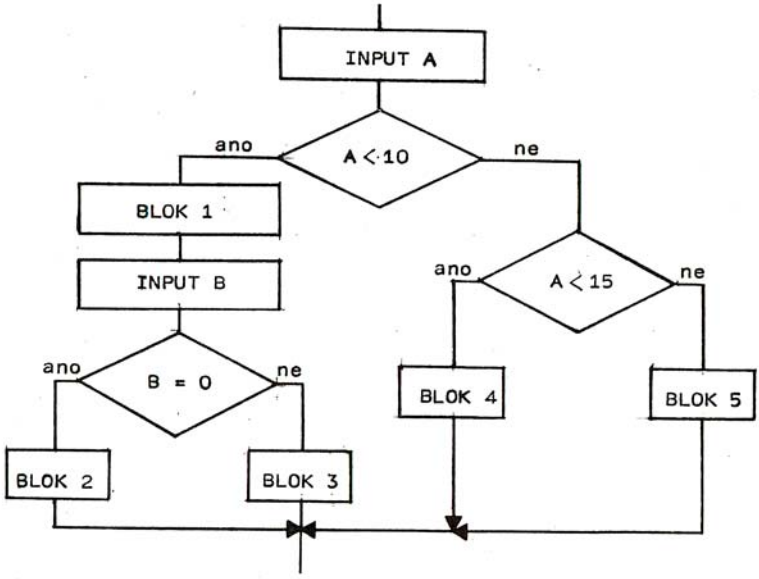
V příkladu řádky 30 až 70 tvoří první blok a řádky 90 až 110 druhý blok (ten by nemusel být, protože jsme použili pouze jeden příkaz.

Nyní si ukážeme vnořování podmíněných příkazů s použitím bloků:

```
10 INPUT A
20 IF A< 10 THEN
30 BEG              ... hlavní podmíněný příkaz
40 PRINT "BLOK 1" .. 1. blok
50 INPUT"B=";B
```

```
60 IF B=0 THEN          ... vnořený podmíněný příkaz
70 BEG
80 PRINT "BLOK 2"
90 PRINT "B=0"
100 BEND
110 ELSE
120 BEG
130 PRINT "BLOK 3"
140 PRINT "B<>0"
150 BEND
155 BEND
160 ELSE
170 BEG
180 IF A<15 THEN        ... vnořený podmíněný příkaz
190 BEG
200 PRINT "BLOK 4"
210 BEND
220 ELSE
230 BEG
240 PRINT "BLOK 5"
250 BEND
260 BEND
```

Uvedený příklad si znázorníme jako vývojový diagram:



Jistě mnozí z Vás viděli program zapsaný v jazyce Pascal, který umožňuje při zápisu vkládat levostranné mezery a tím zvýrazňuje opticky blokovou strukturu programu. Basic V5.0 toto rovněž převzal z Pascalu a při zápisu programu již nedělá automaticky kompresi levostranných mezer a program ukládá do paměti tím, jak je zapsán z klávesnice.

Příklad použití:

```
10 IF A=1 THEN          ... hlavní program
20 BEG
30 tělo programu pro THEN
```

```
40 IF B=0 THEN          ... vnořený test
50  BEG
60 tělo programu        ... tělo vnořeného programu
70  BEND                ... ukončení vnitřního těla
80 BEND                 ... ukončení těla pro hlavní
                            program"THEN"
90 C=A                  ... pokračování hlavního programu
```

Vidíte, že program s levostrannými mezerami je z hlediska bloků podstatně přehlednější a okamžitě poznáte, jak se nám program skládá a jestli máme všechny bloky řádně ukončeny příkazem BEND.

Příkazy bloků jsou specialitou Basicu EXP V5.0 a nejsou ve všech implementacích jazyka Basic. Proto pozor při přenášení programů pod jiný interpret jazyka Basic.

Poznámka:

Místo podmínky ve výrazu IF může být i logická proměnná, která má hodnotu pravda nebo nepravda. To znamená, že podmínku si můžeme nejdříve vyhodnotit a přiřadit logické proměnné (numerická proměnná) a tu pak použít v podmíněných příkazech. S výhodou toho můžeme použít při složitých relačních výrazech, které se nám vícekrát opakují. Pak provádění programu je rychlejší, protože podmínka se nemusí vždy znovu vyhodnocovat.

GOTO - nepodmíněný skok

Při programování mnohdy zjistíte, že byste potřebovali přerušit provádění programu podle vzestupných čísel řádků a že potřebujete pokračovat ve zcela jiném místě v programu. Obvykle se Vám to stane, když si rozvětvíte program do několika větví pomocí podmíněných skoků typu IF, a pak potřebujete větvě opět slučovat. Nové moderní programovací jazyky typu Pascal, ADA, které jsou založeny na principech strukturovaného programování, nepodmíněný skok používají zcela minimálně a použití programátorům záměrně znesnadňují.

Je to proto, že programy, které "hýří" příkazy skoků GOTO jsou zcela nepřehledné a snadno u těchto programů dochází k chybám. **Proto příkazy skoků používejte** pokud možno co nejméně. Když si provedete řádnou analýzu problému, často zjistíte, že tam, kde jste dříve používali skoky, se snadno obejdete bez těchto příkazů (s využitím cyklu typu WHILE nebo REPEAT a podprogramů).

Formát:

```
GOTO číslo řádku
```

```
GO TO číslo řádku
```

Oba zápisy jsou shodné, protože interpret jazyka Basic ignoruje mezery jako nevýznamné znaky. Pozor však při volbě identifikátorů, GO a TO jsou samostatná klíčová slova, která se nesmí použít v identifikátorech. Číslo řádku v příkazu skoku udává číslo programového řádku, na kterém má program pokračovat.

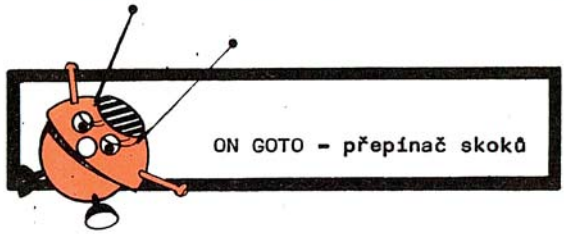
Příklad:

```
10 INPUT A
20 IF A=-1 THEN
30 BEG
40 PRINT "KONEC"
50 GOTO 100        ... skok na konec programu
60 BEND
70 PRINT A         ... tisk vstupní číslice
80 GOTO 10         ... skok na začátek programu
100 END
```

Jednoduchý program, který vám bude neustále vypisovat zadaná vstupní čísla do té doby, dokud nezadáte číslo "-1".

Poznámka:

V příkazu GOTO může být pouze číslo řádku, nemůže zde být proměnná místo čísla řádku! Basic V5.0 nepodporuje nepodmíněné skoky. Pokud potřebujeme větvit skoky, můžeme použít přepínače.



Podle hodnoty řídicí proměnné lze řídit skoky na jednotlivé příkazové řádky.

Formát:

```
ON říd.proměnná GOTO č.ř.1,č.ř.2,... č.ř.N   (seznam čísel řádků)

ON říd.proměnná GOTO seznam čísel řádků ELSE č.řádku
```

Podle hodnoty řídicí proměnné se provede skok na číslo řádku. Je-li hodnota "1", pak se skočí na první příkazový řádek, jehož číslo je uvedeno v seznamu (č.ř.1). Je-li hodnota řídicí proměnné "2", pak se skočí na číslo řádku podle druhého prvku seznamu, atd.

Je-li číslo mimo rozsah prvků seznamu, pak se přejde na další příkaz za příkazem přepínače. Pokud použijeme druhý tvar, tj. použijeme klíčové slovo ELSE, pak se pro všechny hodnoty mimo seznam provede skok na uvedené číslo řádku.

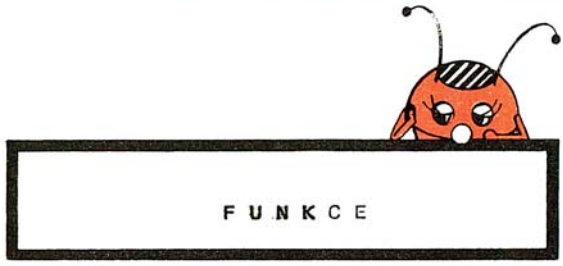
Uvedeme si jednoduchý příklad, který si můžete vyzkoušet a tak nejlépe pochopíte funkci přepínače.

Příklad:

```
10 INPUT A
20 ON A GOTO 50,100,150     ... přepínač pro tři hodnoty
                                1,2,3
30 STOP                     ... indikace zastavení pro
                                jiné hodnoty
50 PRINT A:GOTO 10          ... příkazy pro přepínač=1
100 PRINT A:GOTO 10         ... příkazy pro přepínač=2
150 PRINT A:GOTO 10         ... příkazy pro přepínač=3
```

Po spuštění programu při zadání čísla se provede skok na příslušný tisk, je-li číslo v rozmezí 1 až 3 a pro jiné číslo se pokračuje na řádku 30 STOP, který vypíše BREAK ON LINE 30. Příklad můžeme pozměnit na ukázku s ELSE, když změníme řádek 20 a doplníme řádek 200

```
20 ON A GOTO 50,100,150 ELSE 200  ... přepínač
..
200 PRINT "KONEC - NESPRAVNE CISLO" .. ukončení pro
                                       ostatní čísla
```



FUNKCE

V matematice se často setkáváme s pojmem funkce, která představuje složitý aritmetický výraz. Tento výraz se vyhodnocuje pro určitou hodnotu a výsledkem vyhodnocení je opět konkrétní hodnota (pokud výraz má pro danou hodnotu řešení). Protože jazyk Basic vznikl pro výuku programování, ale i pro běžné používání a řešení úloh technicko-ekonomického charakteru nejen programátory specialisty (kteří dnes používají obvykle jiné programovací prostředky), ale i běžnými uživateli, kteří mají pouze základní znalosti z výpočetní techniky. Proto jsou v interpretu jazyka Basic zabudovány nejčastěji používané funkce z běžné praxe. Dále Basic umožňuje uživatelům vytvářet si vlastní funkce v programu a ty pak používat místo neustálého vypisování (rozvoje) funkce. Funkce se podobá podprogramům, od kterých se liší tím, že jsou volány jménem funkce (která představuje ve výrazu proměnnou) a po provedení funkce vrací hodnotu.

Kromě numerických funkcí má Basic zabudovány příkazy - funkce pro práci s řetězcovými proměnnými, které umocňují využitelnost jazyka Basic i pro oblast zpracování textů. Provedeme si stručné rozdělení funkcí u Basicu V5.0:

a) zabudované numerické funkce
b) zabudované řetězcové funkce
c) jednořádkové uživatelem definované funkce (numerické)
d) víceřádkové uživatelem definované funkce (numerické i řetězcové)

U funkcí rozsliňujeme dvě fáze: definice funkce, aby překladač znal uvedenou funkci a mohl ji vyhodnotit (pro body a, b jsou již součástí překladače a nemusíme je definovat). Funkce uvedené pod body c, d se musí před použitím nadefinovat.

Druhá fáze je použití pro vyhodnocení funkce pro danou hodnotu, které se provede voláním funkce. Volání se provede uvedením jména funkce na pravé straně přiřazovacího příkazu.

Standardní numerické funkce:

ABS(N) ... absolutní hodnota numerické proměnné N (převede na kladné číslo)

INT(N) ... celočíselná část čísla N (odřezání desetinné části)

SGN(N) ... znaménko čísla N (-1 pro A<0, 0 pro A=0 a +1 pro A>0)

SQR(N) ... druhá odmocnina čísla N (číslo N musí být kladné)

LOG(N) ... přirozený logaritmus čísla N (v matematice označení ln N; N - kladné)

EXP(N) ... exponenciální funkce (přirozené číslo e na N-tou)

SIN(N) ... trigonometrické funkce sinus (N v radiánech, lze použít i stupně podle nastavení příkazem DEG)

COS(N) ... trigonometrická funkce cosinus

TAN(N) ... trigonometrická funkce tangens

ATN(N) ... trigonometrická funkce arctangens

RND(N) ... generátor pseudonáhodných čísel v intervalu od 0 do 1. Je-li N záporné, pak se provádí nastavení generátoru na výchozí pozici a generuje se stejné číslo. Lze využít pro ladění programů.

FRE(0) ... délka volné paměti v Bytech

FRE(A$) ... délka volné paměti pro řetězcové proměnné
Pozn. Pokud se při funkci FRE dosadí záporný výsledek, pak máme k dispozici více než 32 7678 paměti. Celkovou kapacitu si vypočítáme ze vztahu 65535 + FRE(0).

Příklady:

ABS(5)=5

ABS(12.5)=12.5

ABS(-10)=10

INT(10.125)=10

INT(-12.2)=13

INT(2.9999)=2

Pokud bychom chtěli zbytek, pak ho vypočítáme jako A-INT(A)

2.9999-INT(2.9999)=0.9999

SGN(-25)=-1

SGN(0)=0

SGN(125)=+1

Pro trigonometrické funkce je možno příkazem DEG ON nastavit parametr funkce ve stupních. Příkaz DEG OFF přepne na radiány. Implicitní nastavení je na radiány.

DEG ON: SIN(30) = 0.5 ... zadání úhlu ve stupních

DEG OFF: SIN(30) =-.988032 ... zadání úhlu v radiánech

V Basicu není zabudována funkce pro zaokrouhlování. Pokud potřebujeme číslo zaokrouhlit, pak použijeme výraz INT(N+0.5).

Funkce pracující s řetězci:

Rozeznáváme dva typy funkcí pracujících s řetězci. První typ dává výsledek opět jako řetězec a funkce obsahuje znak řetězce za jménem ("$") a druhý typ pracuje s řetězci jako operandy a výsledek je numerická hodnota. Tyto funkce nepoužívají označení $ za jménem.

LEN(A$) ... funkce vrátí číselnou hodnotu, udávající počet znaků v řetězci A$

VAL(A$) ... funkce vrací číselnou hodnotu řetězce A$. První nenumerický znak (kromě mezery a tečky) ukončuje číslo. Používá se pro převod čísel ve znakovém tvaru do vnitřního tvaru pro čísla, která můžeme používat pro výpočty.

ASC(A$) ... funkce vrací dekadickou hodnotu prvního znaku z řetězce A$. Je to inverzní funkce k funkci CHR$.

STR$(N) ... funkce převádí číslo N na řetězec číselných znaků (hodnota čísla ve znakovém tvaru). Levostranná mezera místo znaménka plus je zachována, pravostranná mezera se ignoruje. Funkce STR$ je inverzní funkcí k funkci VAL.

CHR$(N) ... převod číselné proměnné N na odpovídající znak v kódu ASCII

LEFT$(A$,N) .. funkce vytváří nový řetězec z řetězce A$ tak, že přenese do nového řetězce N znaků zleva.

RIGHT$(A$,N) . funkce vytváří nový řetězec z řetězce A$ tak, že přenese do nového řetězce N znaků počítaných od pravého kraje řetězce (pozor, řetězec neotáčí!)

MID$(A$,N) ... funkce vytváří nový řetězec z řetězce A$ tak, že přenese zbytek řetězce od pozice dané číslem N

MID$(A$,N1,N2) ... funkce vytváří nový řetězec z řetězce A$ tak, že přenese N2 znaků od pozice zadané číslem N1.

Pro vyzkoušení a pochopení těchto řetězcových funkcí si napište jednoduchý program:

```
10 INPUT "N,A$";N,A$
20 PRINT LEFT$(A$,N)
30 PRINT RIGHT$(A$,N)
40 PRINT MID$(A$,N)
50 PRINT MID$(A$,N,2)
```

Při spuštění programu si zadáte číslo N (např. 3 a řetězec ABCDEF) a na obrazovce se zobrazí:

```
ABC    ... tři první znaky zleva
DEF    ... poslední tři znaky
CDEF   ... zbytek řetězce od třetí pozice
CD     ... část řetězce od třetí pozice, dva znaky
```

Ostatní funkce a kombinace si snadno vyzkoušíte sami. Jenom malá poznámka: Basic V5.0 pracuje se znaky ASCII v sedmibitové kombinaci (nevyužívá se pro znaky 8. bit, který Basic využívá pro svoje účely), takže můžeme použít dekadickou hodnotu až 127.

Definice a volání jednořádkových numerických funkcí

Pro definici uživatelských numerických jednořádkových funkcí je klíčové slovo DEF následované písmeny FN a jménem funkce, pro které platí stejná pravidla jako identifikátory proměnných (tj. musí začínat písmenem a významné jsou pouze první dva znaky!). Za jménem je v kulatých závorkách uvedeno jméno formálního parametru. Vlastní formální parametr se však ve vlastní funkci nemusí vyskytovat.

Formát:

```
DEF FN jméno (parametr) = výraz
```

Příklady:

```
DEF FNA(X)=X*X + X + 1
DEF FNAB(X) = 3*X + 3
DEF FNC(X) = X +FNA(4)      ... pozn. Funkce FNA musí být
                                již definována dříve
DEF FNK(X) = 3.14159        ... definování konstanty (tu
                                ovšem můžeme definovat i
                                jinak) a zde slouží jako
                                ilustrace, že formální pa-
                                rametr se nemusí ve výrazu
                                vyskytovat
DEF FNR(RA) = 3.14159/180*RA ... převod stupňů na
                                radiány
```

Volání funkce:

Volání funkce se provede uvedením jména funkce ve výrazu a za formální parametr se dosadí skutečná hodnota parametru v době volání.

Příklad: (použijeme výše uvedené příklady definice):

```
100 PRINT FNA(2)          ... vytiskne se hodnota 7
150 PRINT FNC(3)          ... vytiskne se hodnota
                              3 + 21, tj. 24
```

Definice a volání víceřádkových funkcí a parametru

Protože Basic V4.0 a mnohé jiné Basicy neumožňují definování uživatelských řetězcových funkcí a proto, že v mnoha případech nevystačíme pouze s definicí funkce na jednom řádku, byl Basic V5.0 rozšířen o možnost definování uživatelských vícepříkazových funkcí, které mohou pracovat jak s numerickými, tak i řetězcovými proměnnými a konstantami. Definice umožňuje použití více formálních parametrů. V definičním těle popisu funkce musí být jménu funkce přiřazena určitá hodnota. Definice funkce je ukončena příkazem ENDF.

Formát:

```
FUNC jméno (par1,par2,...par N)
        .
        .
        .
    tělo funkce
        .
        .
        .
jméno = výsledek
ENDF
```

Tato víceřádková definice funkcí plně nahradí výše uvedenou jednořádkovou definici funkce, která je zde zachována z důvodů kompatibility s jinými verzemi jazyka Basic. Volání funkce se provádí stejným způsobem jako u jednořádkových funkcí, tj. uvedením jména funkce. Aby překladač poznal, že se jedná o volání víceřádkové funkce, je jméno uvedeno znakem "&". Počet skutečných parametrů funkce musí souhlasit s počtem formálních parametrů u definice funkce.

Příklad: Sečtení tří úhlů X1,X2,X3

```
10 FUNC UHEL(X1,X2,X3)
20 X=X1 + X2 + X3
30 UHEL = X
40 ENDF
  .
  .
  .
```

... bylo možné zapsat rovnou UHEL=X1+X2+X3, tento je pouze demonstrační, že lze psát na více řádků a že je nakonec nutno do jména přiřadit výsledek!

```
200 BETA=&UHEL(ALFA,GAMA,0.57) ... volání funkce, kde formální parametry X1-X3
                                   byly nahraženy hodnotami
                                   proměnných ALFA,GAMA a
                                   konstantou 0.57. Výsledek
                                   funkce se dosadí do pro-
                                   měnné BETA.
```

Podprogramy (procedury)

Podprogramy tvoří velmi důležitou část programování a jsou implementovány na všech vyšších jazycích, a proto ani Basic netvoří výjimku. Podprogramy tvoří základní kameny strukturovaného programování, zkracují programy, programy jsou přehlednější a lineárnější a navíc šetří místo v paměti počítače. To jsou hlavní přednosti používání podprogramů. Ve srovnání programů bez podprogramů s podprogramy, jsou tyto o něco rychlejší, protože se neprovádí úschova místa volání. Ovšem tuto malou výhodu v běžném programování snadno pomineme a budeme využívat výhod, které podprogramy přináší.

Jistě jste si sami všimli, že se v programech opakují stejné posloupnosti příkazů, které potřebujeme v různých místech programu. Tento případ můžeme řešit několika způsoby. Nejméně elegantní je, že posloupnost příkazů znovu napíšeme. Jinou, nám již známou možností, je použití ne-

podmíněných skoků a nastavení indikátoru, abychom věděli, odkud jsme skok provedli a zpět se pak vraceli podle indikátoru pomocí přepínače. Sami uznáte, že toto řešení také není zcela bez kazů, protože již musíme vědět, kolikrát se podprogram bude vyvolávat a kam se musí vracet. Navíc nám neumožní vrátit se na příkazy uprostřed víceříkazové řádky a sami si musíme hlídat nastavení. Proto si jistě řeknete, že to by si měl pohlídat sám Basic a on to také umí! Jazyk Basic má zabudované příkazy pro vytváření a volání podprogramů. Naopak vytváření podprogramů je v Basicu tak snadné, že to je až na závadu, protože uživatelé pak snadno zapomínají na správné programátorské zásady, které nám říkají, že problém si rozdělíme na přesně specifikované činnosti, které mají jeden vstupní bod a přesně specifikované výstupy (těch už může být více).

V Basicu může být podprogram libovolná po sobě jdoucí posloupnost příkazových řádků, která je ukončena příkazem pro návrat z podprogramu - klíčovým slovem RETURN. A tak můžete vlastně do podprogramu vstupovat libovolným řádkem - a to je právě ta nectnost jazyka Basic. Zvykněte si vstupovat do podprogramu pouze přes jeden vstupní řádek - vstupní bod podprogramu. Ve vyšších jazycích je tento princip plně respektován, a proto si nezvykejte

na špatné návyky hned od začátku.

Basic V5.0 má doplněnou strukturu podprogramů (procedur), která se definuje před použitím (stejně jako víceřádková funkce) a volá se před použitím jménem procedury. Tyto doplňky jsou převzaty z jazyka Pascal a umožňují dodržovat zásady strukturovaného programování a činí programy přehlednější.

Nyní se seznámíme se zápisem podprogramu a jeho voláním:

Formát:

podprogram	... posloupnost příkazových řádků
RETURN	... klíčové slovo ukončující podprogram

Příkaz RETURN ukončí provádění posloupnosti příkazových řádků jako podprogramu a provede návrat na další příkaz, který je uveden za příkazem volání podprogramu.

Formát:

GOSUB čís. řádku
GO SUB číslo řádku

Uvedené příkazy slouží k volání podprogramu. Překladač Basicu si na tento příkaz uloží potřebné údaje o místě volání do zásobníku a provede skok na uvedené číslo řádku. Posloupnost příkazových řádků provádí jako podprogram a po nalezení klíčového slova RETURN, provede návrat do místa, odkud byl volán. K tomu, aby věděl, kam se má vrátit, slouží uschované údaje v zásobníku.

Příklad:

```
 10 PRINT "CHCES NAPOVEDU?"
 20 GOSUB 200
 30 IF A$="A" THEN 50
    ..
 80 PRINT "ODPOVED A/N"
 90 GOSUB 200
    ..
200 INPUT A$
210 IF A$="A" OR A$="N" THEN 230
220 PRINT "ODPOVEZ A-ANO, N-NE!": GOTO 200
230 RETURN
```

V našem jednoduchém případě používáme podprogram pro vstup znaku z klávesnice a test, je-li zadaný znak přípustný (A,N),jinak opět čeká na správný vstup.

Podprogram je volán (řádek 20 a 90), když požadujeme vstup z klávesnice a odpověď Ano, Ne. Praktic-

ké použití podprogramů si sami jistě vyzkoušíte a budete je s výhodou používat. Nyní se seznámíme s doplněním Basicu V5.0, tj. s procedurami. Úmyslně zde používáme slovo procedura (i když znamená podprogram), abychom ji odlišili od jednoduchého podprogramu, který je běžně zaveden ve všech implementacích jazyka Basic. Procedury jsou převzaty z jazyka Pascal (přesto se od procedur v Pascalu trochu liší, a to je způsobeno tím, že Basic nezná pojem "lokální" proměnná). V Basicu jsou všechny proměnné kdykoliv přístupné a nemusí se definovat (deklarovat) a definice se provádí automaticky při prvním výskytu proměnné v programu. Nově zavedená procedura se musí v programu deklarovat, uvést její jméno, parametry a tělo procedury - posloupnost příkazových řádků. Procedura se ukončuje příkazem ENDP. Procedura je označena jménem a volání této procedury se opět provádí jménem procedury a nikoliv číslem řádku. Proto se do procedury vstupuje pouze přes vstupní bod procedury a nelze vstupovat do libovolného místa! Nejlépe je uvádět procedury a víceřádkové funkce na začátku programu za deklaracemi polí, popř. i proměnných. Při příkazu RUN se definice procedur a funkcí neprovádí jako hlavní program (pouze se vytvoří tabulka odkazů na jména procedur a funkcí), na rozdíl od "normálních" podprogramů, které je

nejlépe umisťovat za tělo hlavního programu (tj. za příkaz END).

Formát:

```
PROČ jméno (par1, par2,par3,..par N)
      .
      .
tělo procedury
      .
      .
ENDP
```

Deklarace procedury se uvádí klíčovým slovem PROC, za kterým následuje jméno procedury, které se tvoří podle zásad pro jména proměnných (začíná písmenem a platné jsou pouze první dva znaky). Za jménem procedury následují v kulatých závorkách formální parametry. Procedura může být i bez formálních parametrů.
V parametrech procedury může být použito klíčové slovo VAR, které způsobí, že výsledek se dosadí do uvedené proměnné.

Poznámky k procedurám a funkcím:

Jména funkcí a procedur se vytváří podle zásad pro jména proměnných a mohou mít libovolný počet znaků. Jména mohou být shodná se jmény proměnných a polí. Počet pa-

rametrů v procedurách a funkcích není omezen a při volání musí počet parametrů odpovídat zadanému počtu parametrů při deklaraci. Funkce a procedury deklarujte na počátku programu. Pozor na formální parametry, které se stávají součástí seznamu proměnných, a proto identifikátory (jména) proměnných použitá jako formální parametry funkcí a procedur nepoužívejte v hlavním programu pro jiné proměnné, protože by došlo ke ztotožnění proměnných a tím k jejich přepsání při vyvolání procedury, resp. funkce. Volání procedury se provádí standardním postupem, tj. použitím příkazu GOSUB.

Formát:

```
GOSUB jméno (par1, par2,...,par N)
```

kde jméno je název procedury zavedený při deklaraci a par jsou skutečné parametry volání procedury místo formálních parametrů. Počet musí souhlasit s deklarací procedury.

Příklad:

```
10 PROC PRENOS          ! deklarace procedury
20 A=1                  ! tělo procedury
    .
    .
50 ENDP                 ! ukončení proc.
    .
    .
```

```
hlavní program
500 GOSUB PRENOS          ! volání procedury PRENOS
      .
      .
550 IF X=1 THEN GOSUB PRENOS    ! další volání procedury
      .
      .
800 END
```

Příklad použití VAR u parametru, tj. předání výsledku do proměnné:

```
10PROC ABS (X, VAR Y)
20 Y=Y+X                        ! tělo procedury
30 ENDP                         ! hlavní program
40 A=1
50 GOSUB ABC(2,A)               ! volání procedury
```

Po provedení procedury bude v proměnné A hodnota 3, protože VAR způsobí přenos z formální proměnné do skutečné proměnné.

Poznámka:

Použijeme-li klíčové slovo VAR, pak za tento formální parametr nelze dosazovat konstantu, ale vždy to musí být proměnná. VAR uvádí proměnnou, do které se má po ukončení procedury přenést hodnota.

ON GOSUB - přepínač podprogramů

Podobně jako pro nepodmíněné skoky je v jazyku Basic zaveden přepínač pro skoky do podprogramů. Po provedení podprogramu se provede návrat za příkaz přepínače.

Formát:

```
ON řídící proměnná GOSUB seznam
```

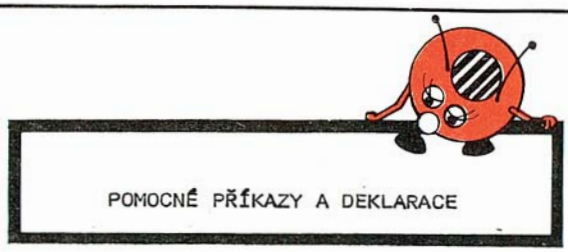
```
ON řídící proměnná GOSUB seznam ELSE podprogram
```

kde řídící proměnná může nabývat hodnot od 1 do N (N je počet prvků seznamu podprogramů). Seznam je posloupnost čísel řádků začátků podprogramů pro "normální" podprogramy a nebo jmen procedur. Pokud jsou uvedeny jména procedur s parametry, musí být uvedeny i skutečné parametry procedur.

Pro druhý tvar přepínače s ELSE se provede podprogram (tj. skok na číslo řádku, nebo podle jména procedury) pro hodnoty řídící proměnné, které jsou mimo povolený rozsah (tj. menší nebo rovné nule a větší než je počet prvků seznamu).

U přepínačů lze kombinovat čísla řádků a jména procedur. V tomto případě pozor při použití povelu RENUM na pře-

číslování čísel řádků i příkazu ON GOSUB, ale pouze u těch čísel, která jsou uvedena před jménem procedury. Čísla řádků podprogramů, která jsou uvedena za jménem, se nepřečíslují!!! Proto Vám nedoporučujeme používat kombinovaný způsob u přepínačů, i když to překladač umožňuje.



POMOCNÉ PŘÍKAZY A DEKLARACE

Zatím jsme se zabývali pouze jednoduchými proměnnými, ale často se vyskytují případy, kdy potřebujeme pracovat s množinou uspořádaných prvků, které mají vzájemnou souvislost - prvky polí. Pole má pro všechny prvky stejné jméno a na jednotlivé prvky se přistupuje pomocí indexů, které jsou uzavřeny v kulatých závorkách. Pole mohou být jednorozměrná nebo vícerozměrná. V Basicu V5.0 je možno použít až osmirozměrné pole, ale zároveň jsme omezeni použitelnou kapacitou operační paměti. Některé jiné jazyky Basic mají počet dimenzí polí omezen nižším počtem než V5.0. Dolní mez, tj. první prvek pole má hodnotu indexu "0" a v deklaraci pole se neuvádí. Implicitní horní index pole je 10 (pole o jedenácti prvcích). Abychom mohli změnit počet prvků v poli změnou horní hranice, je v jazyku Basic zaveden příkaz DIM. Pole mohou být jak pro numerické, tak i pro řetězcové proměnné. U řetězcových proměnných si musíme příkazem CLEAR rezervovat dostatečný prostor pro řetězcové pole.

Formát:

```
DIM jméno pole (horní index)
```
... pro jednorozměrné pole

```
DIM jm.pole (h.i.1, h.i.2,...h.i.N)
```
... pro vícerozměrné pole

Jméno pole se tvoří podle stejných zásad jako pro jednoduché proměnné a může se s nimi shodovat. Horní index ukazuje na poslední použitelný prvek pole. Je-li vícerozměrné pole, pak horní indexy jednotlivých dimenzí oddělujeme čárkami. Příkaz DIM má ještě třetí tvar pro definici více polí současně.

Formát:

```
DIM jm1(seznam indexů),jm2(seznam indexů),...,
    jm N(seznam indexů)
```

Jednotlivá jména polí s mezemi jsou od sebe oddělována čárkami, Řetězcová pole mají za jménem znak pro pole "$".

Příklad:

```
10 DIM A(4),B(23),D$(5)     ! vymezení pole A o 5ti prv-
                              cích s indexy 0 až 4, pole
                              B o 24 prvcích a řetězcové
                              pole o 6ti prvcích
```

Poznámka:

Doporučujeme vždy používat příkaz DIM pro vymezení pole v jazyce Basic, i když je zavedena implicitní deklarace polí. Deklarace má několik předností - máme přehled o použitých polích, šetří se paměť, protože pole dimenzujeme podle skutečných počtů prvků a to má podstatný vliv u vícerozměrných polí, kdy požadavky na paměť rostou mocninou počtu dimenzí a zrychlí se provádění programů. Zrychlení dochází proto, že pole se vymezí na začátku programu a ne až v průběhu chodu programu, kdy se musí znovu organizovat rozdělení paměti. Tento důvod je jedním z důvodů, proč do Basicu V5.0 byl zaveden příkaz i pro deklaraci proměnných VAR.

Formát:

```
VAR seznam proměnných
```

kde seznam proměnných obsahuje jména numerických a řetězcových proměnných, oddělených od sebe čárkou. Důvody, proč je tento příkaz zaveden, již byly uvedeny. V Basicu platí implicitní deklarace výskytem jména proměnných. Tato im-

plicitní deklarace mnohdy vede k těžko odhalitelným chybám, kdy se přepíšeme ve jméně proměnné a Basic okamžitě vytvoří novou proměnnou a neupozorní na chybu tak, jak to udělají jiné jazyky (Pascal, Algol, Fortran), kde není implicitní deklarace. Hlavní důvod je zase v přehlednosti programů a jeho zrychlení. Příkaz VAR má ještě druhou funkci, se kterou jsme se již setkali u procedur, kdy je příkaz VAR uveden před formálním parametrem, pak se hodnota dosadí do skutečné proměnné.

Pomocné příkazy

Pomocné příkazy ulehčují práci programátorům, činí program a zobrazování čitelnější.

Příkazy pro poznámky v programu:

Formát:

```
REM text
```

je-li použito klíčové slovo REM na počátku příkazové řádky, pak se celá řádka bere jako komentář a znaky se nevyhodnocují. Je-li použito za oddělovačem příkazů ":", pak zbytek řádku je brán jako komentář (poznámka). Pomocí příkazů REM můžeme program učinit čitelnější tím, že si oddě-

líme jednotlivé funkční bloky a můžeme si zapsat vysvětlující text k činnosti programu. Pozor **-text** za REM se ukládá do paměti počítače a zabírá tak část paměti. Pro poznámky je ještě jeden zkrácený tvar.

Formát:

```
! text
```

znak "!" (vykřičník) nahrazuje klíčové slovo REM. Činnost je shodná s REM.

Příkaz pro přepínání ze stupňů na radiány a naopak:

Formát:

DEG ON	... proměnné u trigonometrických funkcí jsou ve stupních
DEG OFF	... proměnné jsou v radiánech

Implicitní nastavení je pro radiány. Při vyhodnocení příkazu DEG se testuje další slovo, je-li OFF, pak provede přepnutí na rad. Jinak přepne na stupně.
Příklad je uveden u trigonometrických funkcí (pro sin 30).

Příkaz pro předání dekadické hodnoty tlačítka klávesnice:

Formát:

```
proměnná = INKEY
```

INKEY je vlastně funkce, která předá dekadickou hodnotu poslední stisklé klávesy během provádění programu. Není-li stisknuta žádná klávesa, pak funkce INKEY předá hodnotu "0".

Příklad:

```
10 IF INKEY )0 THEN 100 ELSE 10   ! čekání na stisknutí
                                    libovolné klávesy
```

Příkaz pro převod hexadecimálního čísla na dekadické:

Tento příkaz je opět zvláštní funkcí a je určen hlavně pro práci v Basicu s assemblerem.

Formát:

```
HEX (hexa.číslo)
```

Hexa číslo je číslo v šestnáctkové soustavě - hexadecimální číslo v rozmezí 0H až FFFFH.
Pozor, místo hexačísla nelze použít proměnnou.

Příkaz pro zvukovou signalizaci:

Příkaz umožňuje na mikropočítači nastavit jeden ze sedmi tónů a délku jeho trvání.

Formát:

```
BEEP
BEEP tón
BEEP tón,délka
BEEP nast 1; nast 2 ...;nast N
```

Příkaz BEEP má několik tvarů, při základním se provede signalizace nejvyšším tónem jednotkovou délkou. Tón je možno nastavit čísly 0 až 7 přičemž 1 je nejhlubší a 7 nejvyšší tón. Číslo 0 je bez zvukového výstupu. Délka je počet jednotek trvání (20 ms) a může být v rozsahu 0 - 255. Délka a tón tvoří nastavení zvukového signálu. Pokud chceme v jednom příkazu BEEP nastavit více zvukových signálů, pak jednotlivá nastavení oddělujeme znakem ";" (středník).

Poznámka:
Příkaz BEEP můžeme použít pro generování časové prodlevy, je-li vypnut zvuk (tón=0).

Příkazy pro práci s kurzorem a obrazovkou:

Formát:

```
CURS X,Y
```

Příkaz nastaví kurzor na zadané místo na obrazovce dané souřadnicemi X a Y (v alfanumerickém režimu), tj. sloupec a řádek. Souřadnice X udává sloupec a může být v rozsahu 0 až 39 a Y 0 až 23.

Pozor, při příkazech PRINT s funkcí TAB, kdy funkce TAB nemusí správně pracovat.

Příkaz pro nastavení módu zobrazení:

Formát:

```
MODE N
```

kde N je číslo, udávající mód pro zobrazení a umožňující paralelní připojení tiskárny k zobrazování (co se zobrazuje, to se i tiskne - pozor, nemáte-li připojenou tiskárnu, pak se program nastaví v čekací smyčce na připravenost tiskárny). Bit 0 nastavuje připojení tiskárny, tj. "0" - odpojena "1" - připojena (dekadická hodnota

0,1). Bit 7 určuje způsob zobrazení "0" normální (bílé na černém pozadí),"1" - inverzní (černé na bílém pozadí).

Takže dostáváme 4 kombinace 0 - normální, bez tisku
1 - normální, s tiskem
128 - inverzní, bez tisku
129 - inverzní, s tiskem

Příkaz ke smazání obrazovky:

Formát:

CLS

Příkaz nemá žádné parametry. Provede výmaz aktuálního "okna" (není-li zadáno okno, pak celé obrazovky) a provede nastavení ukazatele na základní postavení (první řádek v levém horním rohu - "HOME"). Implicitní nastavení okna je celá obrazovka.

Příkazy pro nastavení "okna":

Pro nastavení "okna" je příkaz WND (window - okno) a to znamená rezervování určité části obrazovky, ve které se provádí vlastní zobrazování textů.

Okno si můžeme představit jako malou obrazovku na obrazovce displeje, na kterou se provádí veškeré výstupy.

Plocha obrazovky jakoby neexistovala pro výstup znaků (je chráněna). V okně se provádí "rolování textů". Rolování textu je posun textů o řádek nahoru a horní řádek se ze zobrazení ztrácí.

Formát:

WND počet řádků
WND souřadnice
WND #

První tvar příkazu rozděluje obrazovku na dvě části (horní - rezervovanou, dolní přístupnou pro zobrazování). Tento příkaz je dílčím tvarem druhého tvaru se souřadnicemi a je uveden samostatně, protože byl zaveden v Basicu **EX V4.0**.

Souřadnice se uvádí v posloupnosti Y_0, X_0, Y_1, X_1 a jednotlivé souřadnice se oddělují čárkami. Nemusíme uvádět vždy všechny čtyři souřadnice, ale musíme dodržet uvedenou posloupnost souřadnic.

Příklad:

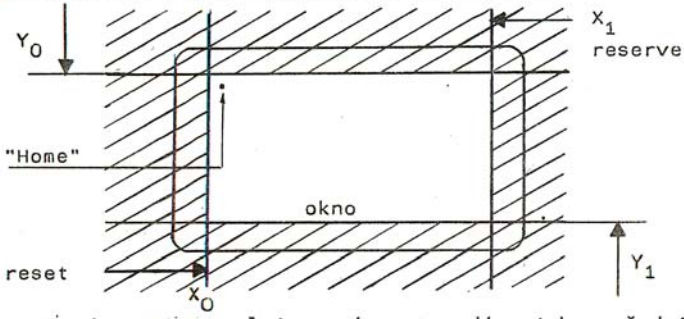
```
WND 5          ! rezervuje 5 řádků shora, zbytek je
                 přístupný
```

```
WND 4,8        ! rezervuje 4 řádky shora a 8 sloupců
                 zleva, vytváří okno v pravém dolním rohu

WND 4,8,15     ! vytváří pás na obrazovce (5. až 14. řádek
                 a zleva začínající na 9. pozici)

WND 4,8,15,25  ! totéž okno jsme ořízli i zprava od
                 25. sloupce
```

Význam jednotlivých souřadnic je na obrázku.



Basic nekontroluje správnost zadávaných souřadnic okna a nesprávné zadání může mít zcela nedefinované chování systému. V systému jsou zavedena dvě okna a právě třetí formát příkazu WND# provádí přepnutí mezi oběma systémovými okny.

Poznámka:

Okna jsou součástí Monitoru a Basic je využívá. Pomocí služeb monitoru si můžeme vytvořit i uživatelská okna, která zaměňujeme se systémovými. Bližší informace jsou uvedeny v příručce pro práci v assembleru TOOL ASM 80.

Pomocí příkazu CURS se můžete dostat i mimo vymezené okno! Proto pozor při používání tohoto příkazu, máme-li nastavena okna. Upozorňujeme, že příkaz CLS provádí výmaz okna a nastavení ukazatele (kurzoru) do výchozí pozice v rámci okna (první sloupec a první řádek v okně, nazývaný u displejů "HOME").

PŘIŘAZOVÁNÍ KONSTANT PROMĚNNÝCH

Často v průběhu chodu programu potřebujeme přiřadit proměnným předem určenou hodnotu, ať numerickou nebo řetězcovou. Zatím jsme se seznámili s jedním způsobem, a to pomocí přiřazovacího příkazu LET. Forma přiřazení je velmi jednoduchá a snadno použitelná. Stačí napsat příkaz LET proměnná = konstanta.
Například: LET X=5 nebo LET A$="DEN",
přičemž slovo LET je nepovinné a nemusí se uvádět.

Ovšem takové přiřazování má jednu nevýhodu, že stejná konstanta se vyskytuje na mnoha místech v programu a pokud konstanta představuje nějaký parametr, který bychom za čas potřebovali změnit, pak je pracné vyhledávat všechny konstanty v programu a pak je měnit. Zejména když se k programu vracíme po delší době a prakticky už ani nevíme, jak jsme ho naprogramovali. Ukazuje se, že by bylo vhodné mít konstanty uspořádané na jednom místě v programu a v případě potřeby si konstanty "přečíst" a uložit je do proměnných. Basic je vybaven takovým aparátem - příkazy,

které nám umožní vytvořit v paměti počítače "zásobník konstant" a v případě potřeby je přiřadit proměnným. Jsou to příkazy DATA, READ a RESTORE.

Příkaz DATA umožní vytvořit paměť konstant, příkaz READ je umí přečíst a přiřadit proměnným a příkaz RESTORE umožňuje nové použití paměti konstant.

Formát:

DATA seznam konstant

kde seznam konstant obsahuje jednotlivé konstanty - numerické nebo řetězcové oddělené mezi sebou znakem "," (čárka). Konstanty mohou být mezi sebou promíchány, tj. nemusí být stejného typu. Pouze při příkazu pro čtení je musíme správně přiřazovat.

Poznámka:
Na řádku DATA se nesmí použít poznámka (REM nebo "!").

Řetězcové konstanty se v příkazu DATA uzavírají do uvozovek stejně jako v přiřazovacím příkazu LET.

Čtení konstant se provádí příkazem READ. Předtím, než si uvedeme formát příkazu READ si vysvětlíme, jak příkaz pracuje. V překladači Basicu je ukazovátko, které

ukazuje na aktuální konstantu. Při spuštění programu ukazuje na první (tj. nejnižší řádek a příkazy DATA) řádek a první uvedenou konstantu. Při použití příkazu se ukazatel automaticky přesune na další konstantu v příkazu DATA a je-li příkaz vyčerpán, pak se přesune na další řádek s příkazem DATA. Příkaz READ může přiřadit pouze odpovídající konstantu podle typu proměnné. Nemůže do numerické proměnné přiřadit řetězcovou konstantu a naopak do řetězcové proměnné číslo ve vnitřním zobrazení. Příkaz READ umožňuje přiřazení více proměnných najednou.

Formát:

READ proměnná
READ seznam

kde seznam je posloupnost proměnných oddělených čárkami. První tvar je dílčím tvarem druhého typu, kde seznam má pouze jeden prvek.

Příklad:

```
10 DATA "PONDELI", "UTERY", "STREDA", "CTVRTEK",
20 DATA "PATEK", "SOBOTA", "NEDELE"
30 FOR I=1 to 7
40 READ A$
50 PRINT A$
60 NEXT
```

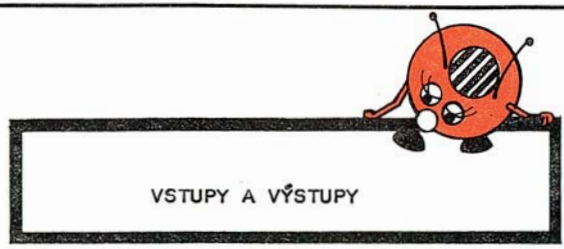
Po spuštění programu se nám na jednotlivých řádcích bude vypisovat řetězcová proměnná A$, které jsou v cyklu postupně přiřazovány konstanty - jména dnů v týdnu. Všimněte si, že konstanty jsou napsány na dvou řádcích a ukazatel dat plynule přechází z jednoho příkazu DATA na druhý. A nyní když se zamyslíte nad naším příkladem, pak pro další dny 8, 9, ... bude nutné znovu vypisovat další, již jednou zavedené konstanty. Abychom to nemuseli provádět a mohli jsme znovu využít zavedené konstanty, to nám umožní příkaz RESTORE.

Formát:

```
RESTORE
RESTORE číslo řádku
```

Příkaz má dva tvary. První bez parametrů, nám nastaví ukazatel na první konstantu jako při spuštění programu. Druhý tvar nám umožní přesunout ukazatel na řádek s konstantami, který je uveden za příkazem RESTORE. Doplníme si náš příklad o další příkazové řádky tak, aby se nám seznam vypsal dvakrát. Doplníme řádky:

```
25 FOR J=1 TO 2   ! cyklus dvakrát pro výpis
60 NEXT 1         ! doplníme proměnnou u konce 1.cyklu
70 RESTORE        ! nastavíme ukazatel na PONDELI
80 NEXT J         ! ukončíme vnější cyklus
```



VSTUPY A VÝSTUPY

Vstupní a výstupní operace a práce se soubory tvoří důležitou část každého programu. V předchozí části jsme se seznámili s výstupy na obrazovku displeje a se vstupy z klávesnice pomocí příkazů PRINT a INPUT a také jsme si řekli, že s dalšími funkcemi těchto příkazů se seznámíme později.

Mikropočítač ONDRA používá jako vnější paměť kazetový mangetofon. Data se zaznamenávají na kazetu sekvenčně, tzn. postupně za sebou. Protože používáme tu nejjednodušší paměť, která se používá u počítačů a která nemá dálkové ovládání všech funkcí, jako je nastavení pro čtení, zápis dat, převíjení a zastavování, musíme tyto činnosti vykonávat my ve spolupráci s počítačem. Počítač nám oznámí na obrazovce displeje, jakou činnost od nás požaduje (nastavení magnetofonu) a po nastavení požadované funkce mu stisknutím tlačítka na klávesnici oznámíme, že činnost byla provedena a že může pokračovat.

Data na kazetě jsou organizována do souborů, tj. ucelené logické celky dat, které mají svoje označení - jméno souboru a pak obsahují vlastní data a konec souboru je označen speciální značkou. První části souboru říkáme hlavička souboru. Na pásce tvoří první blok. Kromě jména jsou v hlavičce uloženy i poznámky. Protože obvykle předem neznáme velikost souboru, členíme data na menší celky - bloky, se kterými pak pracujeme. Činnosti související s organizací a prací se soubory vykonává systémový program, který je u Ondry nazván Mikos, a interpret jazyka Basic spolupracuje s tímto systémovým programem.

Princip práce Mikosu je takový, že po zkontrolování hlavičky přečte blok dat do vyrovnávací paměti (bufferu) a Basicu pak předává data z paměti. Když vyčerpá všechna data z vyrovnávací paměti, pak automaticky načte další blok dat. Obdobně je tomu i při zápisu dat na kazetu. Data vystupující z Basicu se nejdříve ukládají do vyrovnávací paměti a po naplnění vyrovnávací paměti se celý blok zapíše na kazetu a vyrovnávací paměť se uvolní pro vstup (zápis) dalších dat. Po ukončení zápisu musíme systému oznámit, že chceme zápis dat ukončit a že může soubor uzavřít. Práci se soubory si můžeme proto rozdělit na tři základní části:

- Otevření souboru:pro čtení požadovaného souboru, pro zápis nalezení místa, kam výstupní soubor uložit a zápis hlavičky souboru. Pro tyto činnosti máme v Basicu dva příkazy FIND a OPEN.
- Čtení a zápis dat na kazetu; pro tuto činnost máme čtyři příkazy: GET, INPUT, PUT a PRINT.
- Ukončení souboru;příkaz CLOSE.

Nyní ještě několik poznámek k práci s magnetofonem. Celý systém Mikos je založen na principu dálkového ovládání spouštění a zastavování motorku mangetofonu. Protože předem nevíme, jak často budeme vystupovat nebo vstupovat s blokem dat na nebo z magnetofonou, provádí ovládání motorku operační systém podle potřeby. Proto bloky dat na kazetě musí být od sebe odděleny mezerou (mezibloková mezera) tak, aby magnetofon v mezeře stačil zastavit a znovu se rozběhnout. Vzhledem k tomu, že zastavování a rozběh se pokaždé od sebe trochu liší (třeba již tím, jaká je kazeta, kolik pásky je na každé straně kazety atd.), nemůžeme zapsat blok na stejné místo! Proto při práci se soubory se nepřepisuje blok, ale vytváří se nový soubor!! Tak to provádí i velké počítače, pokud pracují s magnetickou páskou. Protože kazeta je magnetické médium, u-

možňuje zrušení starých souborů a přepsání novými - mnohdy i když to nechceme a provedeme špatnou manipulaci s magnetofonem. Proto se systém Mikos vás neustále dotazuje a Vám napovídá tak, aby takových nežádoucích manipulací bylo co nejméně.

Po tomto stručném úvodu přejdeme k příkazům jazyka Basic. Formát dat a způsob práce Mikosu je uveden v příručce Návod k obsluze.

Otevření souboru

Protože máme pouze jeden magnetofon připojený k mikropočítači, můžeme mít otevřen v daném okamžiku pouze jeden vstupní nebo výstupní soubor.

Otevření souboru pro čtení

Formát:

```
FIND
FIND "jméno souboru"
```

Příkaz pro otevření vstupního souboru má dva tvary. První tvar bez parametrů se používá pro zavedení souboru

dotazovým postupem Mikosu. Ve druhém případě zadáváme řetězec se jménem požadovaného souboru. Po nastavení na čtení se začne vyhledávat soubor na kazetě se zadaným jménem. Pro jméno platí konvence Mikosu, tj. můžeme použít hvězdičkovou konvenci (pak se systém po nalezení hlavičky zeptá, zda-li nalezený soubor požadujeme. Pokud ano, provede zavedení, v opačném případě pokračuje v hledání dalšího souboru) nebo zadat i prázdný řetězec a pak se zavádí první nalezený soubor.

Příkaz FIND má i druhou funkci, že nám uzavírá vstupní soubor. To v případě, že máme otevřen jeden vstupní soubor a nyní chceme první ukončit a otevřít nový vstupní soubor. Příkaz FIND tuto činnost provede. Byl-li otevřen vstupní soubor, pak ho uzavře a otevře nový. Jinak vstupní soubory se nemusí uzavírat.

Otevření souboru pro zápis

Formát:

```
OPEN
OPEN "jméno souboru"
OPEN "@jméno souboru"
```

Příkaz pro otevření výstupního souboru má tři typy příkazu. Předtím než si rozebereme jednotlivé typy, připomeneme si činnost související se zápisem souboru. Můžeme ji rozdělit na dva kroky - první, kdy musíme vyhledat místo, kam budeme zapisovat (tj. najít soubor, za který budeme zapisovat - funkce čtení). V některých případech již máme kazetu nastavenou na požadované místo a chceme soubor přidat (append). Druhým krokem je vlastní zápis (nastavení magnetofonu na zápis).

První tvar předává řízení systému Mikos a systém pracuje v dotazovacím režimu. Druhý způsob pracuje v dotazovacím režimu pro nalezení místa a po nastavení magnetofonu na zápis provede zápis souboru. Třetí tvar je pro "append" - přidání a v tomto případě se nevyhledává a pouze nastavuje magnetofon na zápis. Jméno souboru se uvádí jako řetězec. U funkce append je doplněn řídící znak "@", který se ovšem do jména nepřenáší a informuje Basic o požadované funkci přidání.

Poznámka:
Jméno v systému Mikos může obsahovat až 11 znaků. Dále je možno uvést poznámku, až do celkového počtu znaků 64 vč. jména. Jméno souboru je ukončeno první mezerou nebo počtem znaků (11). Při vyhledávání souboru podle jména se porovná-

vá pouze zadaný počet znaků v hledaném jméně, nikoliv celé jméno souboru, tj. 11 znaků. Proto soubory na pásce BAS1 a BAS2 při hledání jména BAS se vybere soubor, na který se narazí buď BAS1 nebo BAS2.

Tím umíme otevřít vstupní nebo výstupní soubor na kazetě.

Zápis dat na pásku

Pro zápis dat na kazetu ve znakovém tvaru se používá příkaz PRINT.

Formát:

```
PRINT #P seznam
```

kde znaky "#P" označují výstup na kazetu a seznam je posloupnost proměnných a konstant, jak numerických, tak i řetězcových. Jednotlivé prvky jsou odděleny oddělovači "," - čárka a nebo ";" středník a lze použít i funkce používané u příkazu PRINT. Výstup je ve znakovém tvaru stejně jako na obrazovce displeje.

Pozor! Pokud takto pořízenou pásku chceme opět zpracovat Basicem, pak musíme dodržet určitá pravidla. Pokud seznam obsahuje pouze jeden prvek, je na pásce oddělen znaky "CR" a "LF" a můžeme ho přečíst z pásky. Pokud je v seznamu více

prvků, pak musíme na pásku doplnit oddělovač ","- čárku tak, aby při vstupu byl nalezen oddělovač vstupních dat.

Příklad:

```
10 OPEN "DATA"        ! výstupní soubor se jménem DATA
20 FOR I=1 to 5
30 PRINT#P I          ! v cyklu uložení čísel 1 až 5
40 NEXT
50 CLOSE              ! uzavření souboru
```

Uvedený příklad nám uloží posloupnost čísel 1 až 5 vzájemně oddělené znaky CR a LF. Tento soubor pak můžeme přečíst.

Upravme příklad pro výstup dvou proměnných:

```
30 PRINT#P I, I*I
```

V tomto případě nám budou na pásku vystupovat dvě proměnné I a I*I, které jsou od sebe oddělený pouze mezerami. Např. 1 1 CR LF 2 4 CR LF ....
Pak při čtení dat z pásky načteme místo dvou prvků 1 1 prvky 11 a 24 a budou nám chybět vstupní data.
Proto musíme řádek 30 upravit tak, abychom na pásku zaznamenali oddělovač dat.

```
30 PRINT#P I;",";I*I
```

V tomto případě se na pásku zaznamenají data ve tvaru 1 , 1 CR LF 2 , 4 CR LF ... a při vstupu dat z pásky budou správně přečteny.

Nyní se seznámíme s použitím příkazu PRINT i pro výstup na jiná zařízení než na kazetu. Říkali jsme si, že k mikropočítači ONDRA je možno připojit i tiskárnu.

Zápis dat na výstupní zařízení

U mikropočítače ONDRA máme jako výstupní zařízení obrazovku displeje, kazetový magnetofon a tiskárnu. Na první dvě zařízení již umíme vystupovat (zapisovat). S tiskárnou můžeme pracovat dvojím způsobem, buď jako se samostatnou tiskárnou a nebo jako "hard copy" tak, že nám současně zobrazuje na obrazovce a zapisuje na tiskárně. Dále pro ladění programů je doplněno "pseudovýstupní" zařízení - nulové zařízení (odpadkový koš - bucket). Toto pseudozařízení nikam nezapisuje a slouží nám pro odladění programu, abychom zbytečně například netiskli nebo nepsali na pásku. Formát příkazů je stejný a liší se pouze označením výstupního zařízení.

Formát:

```
PRINT #zař. seznam
```

kde zař. je označení zařízení a může být H - pro "hard copy", tj. tiskárna a obrazovka současně, P pro výstup na

kazetu, L pro výstup pouze na tiskárnu a B pro nulové zařízení.

Příklad:

```
PRINT # H  A,B,C     ! výstup proměnných A,B,C na tiskárnu
                       a obrazovku
PRINT # P  A         ! zápis na kazetu
PRINT # L A,B,C      ! výstup proměnných A,B,C pouze na
                       tiskárnu
PRINT # B A,B,C      ! pseudovýstup proměnných A,B,C -
                       nikam se nezapisují
```

Poznámky:

U příkazu PRINT můžeme použít všechny funkce pro PRINT včetně příkazu USING. Při příkazu PRINT# H lze použít i znaků české abecedy, u příkazu PRINT # L pouze znaky latinské abecedy (tj. bez diakritických znamének - háčků, čárek).
Tím jsme si ukončili výklad příkazu PRINT.

Vstup dat z pásky

V předcházející části jsme se naučili zapisovat data na kazetu a nyní se seznámíme s příkazem, který nám umožní čtení dat z pásky.

Formát:

```
INPUT# R seznam
```

S příkazem INPUT jsme se již seznámili při vstupu dat z klávesnice a stejný příkaz je i pro vstup dat z kazety. Aby Basic věděl, odkud má data číst, má pro kazetu v příkazu označení " R". Jinak platí stejná pravidla jako pro čtení z klávesnice. Pozor pouze na to, abychom proměnným přiřazovali správný typ proměnných. Dále si uvědomte připomínku uvedenou u příkazu PRINT o oddělovačích dat.

Uvedeme si příklad, který nám umožní přečíst data z příkladu u příkazu PRINT.

Příklad:

```
100 FIND "DATA"          ! otevření vstupního souboru
110 FOR I = 1 TO 5
120 INPUT# R J           ! čtení dat z kazety do proměnné J
130 PRINT J              ! zobrazení přečtených dat
140 NEXT I
150 STOP
```

Na obrazovce se nám zobrazí přečtená data od 1 do 5. Nyní si upravíme řádek 120 a 130 pro vstup dvou prvků seznamu, tak, abychom mohli přečíst úpravy uvedené u příkladu.

```
120 INPUT# R J,K          ! vstup dvou prvků
130 PRINT J,K             ! zobrazení na obrazovce
```

Tím jsme si ukončili výklad příkazu INPUT:

Basic umožňuje čtení a zápis dat na kazetu po Bytech. Uživateli jsou předávány všechny Byty uvedené na kazetě, tj. včetně oddělovačů, mezer a CR LF. Tato data se musí přiřadit do numerických proměnných a pak je můžeme dále zpracovávat. Tyto příkazy jsou určeny pro speciální použití a při běžném způsobu práce v Basicu je ani příliš nevyužijeme. Proto se o nich zmíníme pouze krátce.

Vstupy Byte z kazety

Formát:

```
proměnná = GET
```

Příkaz přiřadí Byte z kazety do numerické proměnné.

Výstup Byte na kazetu

Formát:

```
PUT proměnná
```

Příkaz PUT uloží hodnotu proměnné (Byte) na kazetu.

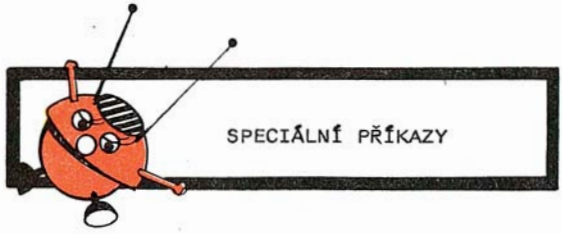
Uzavření výstupního souboru

Jak jsme si již říkali o principu činnosti systému Mikos, data při zápisu jsou ukládána do vyrovnávací paměti a po zaplnění se automaticky zapisují na kazetu. Proto při ukončení zápisu je nutné uzavřít data ve vyrovnávací paměti a vyrovnávací paměť zapsat na pásku. Poslední blok na pásce má speciální označení konce souboru. Proto musíme vždy výstupní soubor uzavírat, aby nedošlo ke ztrátě dat a soubor byl správně ukončen.

Formát:

```
CLOSE
```

Příkaz pro uzavření souboru nemá žádné parametry.



Je to skupina příkazů jazyka Basic, které umožňují práci s podprogramy v assembleru, přímé Vstup/výstupní operace a obsluhu přerušovacího systému. Vzhledem ke konstrukci mikropočítače ONDRA nedoporučujeme příkazy pro obsluhu přerušovacího systému a přímé vstup/výstupní operace používat. Jejich použití může vést k destrukci základního programového vybavení a tím ke ztrátě programu v paměti počítače. Zde jsou uvedeny pouze pro zkušené programátory, kteří by na Ondrovi si chtěli připravovat programy pro jiné mikropočítačové systémy, např. pro systém SAPI 1. Proto tyto příkazy probereme velice stručně.

a) přímé vstup/výstupní operace

- funkce pro čtení Byte paměti:

  Formát:

  ```
  proměnná = PEEK (A)
  ```

  kde parametr A je numerická proměnná udávající adresu místa paměti, jehož obsah se má přiřadit do proměnné.

- příkaz pro zápis Byte do paměti

  Formát:

  ```
  POKE A, Byte
  ```

  kde Byte je numerická proměnná (Byte - 0 až 255), jehož hodnota se uloží na adresu v paměti počítače, jejíž adresa je v proměnné A.

- příkaz pro čtení Byte z portu

  Formát:

  ```
  INP (A)
  ```

  kde A je numerická proměnná, jejíž obsah udává adresu portu (0 - 255). Pozor!!! Tento příkaz nepoužívejte u Ondry!!!

- příkaz pro výstup Byte na výstupní port

  Formát:

  ```
  OUT A, Byte
  ```

  kde A je numerická proměnná, udávající adresu výstupního portu (0 - 255) a Byte je numerická proměnná obsahující výstupní Byte, který se pošle na výstupní port. Tento příkaz lze u Ondry použít, ale používání

nedoporučujeme. Adresy portů jsou uvedeny v příručce pro práci v assembleru (TOOL).

- příkaz čekání na zadání data ze vstupního portu

Formát:

```
WAIT A,M
```

```
WAIT A,M,Z
```

Příkaz má dva formáty, kde A je numerická proměnná - adresa portu, M - je proměnná s maskou a Z je očekávaný znak. Příkazu odpovídá sekvence příkazů v assembleru:

```
WAIT: IN A       ; čti z portu A
      ANI M      ; maska
      CPI M      ; shoda?
      JNZ WAIT   ; ne čekej dál

WAIT1: IN A
       ANI M
       CPI Z
       JNZ WAIT1
```

Příkaz WAIT nelze použít u mikropočítače ONDRA.

- příkaz pro určení adresy proměnné

Formát:

```
PTR (proměnná)
```

Funkce PTR vrací adresu proměnné. Jméno proměnné je uvedeno v kulatých závorkách. U numerické proměnné ukazuje adresa přímo na proměnnou a u řetězcové proměnné na 4 - B tabulku.

Do speciálních funkcí patří i příkaz HEX, který byl popsán již dříve.

b) Obsluha přerušení v Basicu V5.0

U mikropočítače Ondra tyto příkazy nepoužívejte.

Inicializace obsluhy přerušení

Formát:

INTRP číslo řádku, A

příkaz inicializuje obsluhu přerušení v Basicu. Po vykonání každého příkazu Basicu se textuje obsah paměťového místa, jehož adresa je uvedena v proměnné (A). Je-li povoleno přerušení příkazem "ON", potom se přeruší program v Basicu a spustí se podprogram, který začíná na řádku, jehož číslo je uvedeno v příkaze (číslo řádku). Zároveň se přepíše obsah "kliče" (tj. obsah na adrese A) na hodnotu OFFH, která zakazuje přerušení. Uživatel si

musí sám nastavovat tento klič na hodnotu "0",po zpracování přerušení. Podprogram obsluhy přerušení musí končit příkazem "RETURN", aby se mohlo pokračovat v hlavním programu.

Formát:

INTRP ON	... povolení přerušení
INTRP OFF	... zákaz přerušení

Příkaz INTRP ON povoluje přerušení a příkaz **INTRPOFF** zakazuje přerušení.

c) Spolupráce s podprogramy v assembleru

V Basicu V5.0 jsou zabudovány příkazy pro použití podprogramů napsaných v assembleru v Basicu. Jsou dva způsoby práce a podprogramy, jeden ve kterém strojový kód je přímo v jazyku Basic na příkazových řádcích a druhý způsob, kdy strojový kód je nezávislý na programu v jazyce Basic a lze ho do paměti zavést samostatně.

Formát:

```
CODE adresa; hexdata 1, hexdata 2,,..hexdata N
```

kde adresa je počátek vymezeného úseku paměti RAM, do kterého se ukládají hexdata. Hexdata jsou hexadecimální číslice, které se ukládají do paměti a mohou se provádět jako program v assembleru. Adresa je od dat oddělena středníkem a hexadecimální data mezi sebou čárkou.

Příklad:

```
100 CODE HEX(40F0);3E,20,21,0,0,C9
```

Tento program odpovídá programu v assembleru 8080.

```
40F0  MVI A,' '
      LXI H,0
      RET        ;příkaz RET je povinný - návrat z pod-
                  programu
```

Pózor! Podprogramy v assembleru musí končit příkazem RET, pokud požadujeme pokračování programu v Basicu nebo skokem do Monitoru (ukončení činnosti).

Druhou skupinu tvoří příkazy pro volání podprogramů v Basicu:

Formát:

proměnná = CALL (adresa)

proměnná = CALL (adresa, HL)

```
proměnná = CALL (adresa, HL, DE)
```

Příkaz provede volání podprogramu na adrese, která je uvedena jako parametr volání. Příkaz umožňuje nastavit 16ti bitový parametr do registrového páru HL a DE. Platí, že obsah registru L se uloží dále do reg. C a A. Po návratu se obsah registrového páru HL uloží dále do proměnné.

Formát:

```
proměnná = USR (adresa)
```

```
proměnná = USR (adresa, C)
```

Příkaz provede volání podprogramu na adrese, uvedené v příkazu. Jinak se příkaz chová obdobně jako příkaz CALL. Rozdíl je pouze v tom, že se zadávají 8mi bitové proměnné, které se ukládají do reg. C. Po návratu z podprogramu se registr "A" uloží do proměnné.

GRAFICKÉ PŘÍKAZY

Před uvedením grafických příkazů Basicu V5.0/G si nejdříve vysvětlíme některé pojmy a způsob, jak se provádí kreslení.

Mikropočítač ONDRA má vymezenou část paměti RAM jako paměť video. Obsah této paměti je průběžně zobrazován na obrazovce monitoru. Jednotlivým bitům této paměti odpovídají příslušné body na obrazovce. Obrazovka je rozdělená v rastru 320 x 240 bodů, což je zobrazovaných 76 800 bodů. Paměť počítače je organizována po Bytech, jeden Byte je složen z 8 bitů. Bit je nejmenší jednotkou informace a může nabývat hodnot "0" nebo "1". Podle hodnoty bitu je buď bod na obrazovce rozsvícen ("1") nebo zhasnut ("0").

Pro zobrazování bodů v uvedeném rastru je zapotřebí 9600 Byte. Počítač ONDRA má celkovou kapacitu paměti RAM 64 kB, z toho je pro videopaměť rezervováno 10 kB. Na programy nám tedy zbývá 54 kB.

Do videopaměti se ukládají všechny znaky, které mají být zobrazeny na obrazovce monitoru. Způsob ukládání a převod znaků do bodového rastru je řízen systémovými programy.

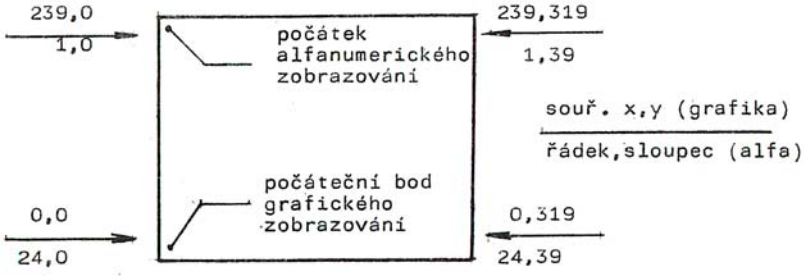
Basic V5.0/G má dva režimy zobrazování. První je zobrazování textů, tj. zobrazování alfanumerických znaků. Například veškerý dialog s počítačem probíhá v tomto režimu, výpis výsledků a zpráv při řešení úloh (pokud nepoužíváte grafické příkazy) je řešen jako výstup alfanumerických znaků. Zobrazování je řízeno programem Monitor. Místo zápisu na obrazovce je signalizováno speciálním znakem - ukazovátkem (nebo se často používá název kurzor). Na jeden řádek je možno v tomto režimu zapsat 40 znaků. Počet řádků na obrazovce je 24, což odpovídá kapacitě 960 znaků.

Po dosažení konce řádku kurzor automaticky přechází na další řádek (pozor - ale sám nedává znak konce řádku). Pro řízení kurzoru máme na klávesnici speciální tlačítka - je to znak "CR" - návrat vozu (na klávesnici je označen ↵), který do paměti uloží znaky pro návrat ODH a znak nového řádku OAH. Dále jsou to znaky, které jsou na klávesnici označeny šipkami (vlevo, vpravo, nahoru, dolů). Tyto znaky generují řídicí znaky, jimiž se řídí pohyb

kurzoru na obrazovce, a současně i ukazatel ukládání do videopaměti.

Alfanumerické zobrazování začíná v levém horním rohu obrazovky a postupně zaplňuje obrazovku shora dolů. Po dosažení pravého dolního rohu se provede "rolování" textu - tj. celý zobrazovaný text je posunut o jeden řádek nahoru.

Druhý způsob zobrazování je grafický režim, kdy obrazovka je rozdělena na bodový rastr. Počátek v grafickém režimu není v levém horním rohu, ale v levém dolním rohu tak, jak je zvykem při kreslení na papír. Grafický režim má vlastní kurzor, který se na obrazovce nezobrazuje (aby na obrazovce nerušil kreslené obrázky).
Pro pohyb kurzoru platí speciální příkazy v Basicu V5.0/G.
Na obrázku je zobrazeno rozdělení obrazovky v alfanumerickém a grafickém režimu.



Basic V5.0/G umožňuje rozdělení obrazovky i do jiného měřítka než je 320 x 240 bodů. K tomu slouží příkaz SCALE, který umožní nastavit měřítko výstupních obrázků podle potřeby, pak parametry většiny příkazů jsou přepočítávány podle měřítka (kromě příkazů, které pracují ve fix módu - pevném měřítku 320 x 240 bodů). To zbavuje uživatele povinnosti, aby sám přepočítával umístění bodů na obrazovce.

Pro kreslení grafů je zaveden příkaz AXES, který vám podle zvoleného měřítka vykreslí osy souřadnic.

Pro pochopení činnosti grafických příkazů je nutno vysvětlit parametr PEN. Tento parametr znamená "pero", způsob, jakým se bude kreslit čára. Je-li hodnota PEN rovna "0", pak jako by pero bylo zvednuté a nekreslí. Při hodnotě PEN rovné "255" se kreslí plná čára. Parametr PEN je bytová proměnná, jejíž obsah určuje kreslení čáry. Po každém nakreslení bodu obsah parametru zarotuje o jeden bit vlevo a podle obsahu bitu 7 se určuje, zda se bit zobrazí. "1" určuje rozsvícení bodu, "0" bod se nezobrazuje. To nám umožňuje kreslit plnou čárou, čerchovanou nebo tečkovanou čárou. Příkaz COLOR nastavuje "mód" zápisu čáry a dá se použít i k nastave-

ní pera. Mód může nabývat hodnot 0 - 3. Pro hodnotu "0" se čára nekreslí (bez ohledu na nastavení PEN), pro hodnotu "1" je zápis realizován funkcí "XOR", tj. provede se inverze bodů čáry - body, které svítily se zhasnou a naopak (bez ohledu na PEN).

Při hodnotě módů "2" je zápis čáry řízen hodnotou pera a při hodnotě "3" se provádí výmaz čáry (bez ohledu na PEN).

Basic V5.0/G má pouze příkazy pro kreslení přímky a nemá příkazy pro kreslení kružnic. Pokud potřebujeme zobrazit křivky, můžeme je zobrazovat pomocí bodů nebo aproximacemi pomocí úseček. Výpočet bodů provádíme pomocí rovnic popisujících danou křivku.

Čára se kreslí vždy definovaným způsobem. Program interpolace pro přepočet do bodového rastru zjistí výchozí a koncový bod, které má spojit přímkou. Z obou bodů vybere podle velikosti úhlu jeden za výchozí. Toto řešení bylo zvoleno proto, aby čáry byly kresleny vždy v definovaném směru a bylo je možno mazat. V případě, že bychom vždy kreslili z výchozího do koncového bodu, mohlo by při zpětném mazání (tj. z bývalého koncového do výchozího bodu) dojít k jiné interpolaci, některé body bychom nesmazali.

Basic V5.0/G je doplněn o příkazy pro grafické tisky textů a uživatelských znaků. Generátor znaků je jiný než v programu Monitor a vychází z rastru 7 x 5 bodů. Uživatel si může v rastru 8 x 5 bodů definovat vlastní grafické značky, které pak může zobrazovat pomocí příkazu LABEL. Pro definici uživatelských znaků se používá příkaz DEF CODE. Výstup textů a uživatelských znaků je řízen měřítkem u příkazu LABEL. Toto měřítko znamená zvětšení základního rastru v ose X a Y. LABEL 2,3;"A" vytiskne písmeno "A" tak, že v ose x bude 10 bodů (každý bod základního rastru je zdvojen) a v ose y bude 21 bodů + 3 body pro oddělení znaků. Zápis bude proveden na místě, kam ukazuje grafický kurzor. Po zápisu bude kurzor umístěn za textem (znakem) na dolní lince písemného rastru.

Basic V5.0/G obsahuje ještě jeden příkaz pro definování uživatelského obrázku, DEF FILL. Tento příkaz obsahuje rovněž měřítko, jako příkaz DEF CODE. Za měřítkem (odděleným znakem ":" - středník) se uvádí hexadecimální kód bodů kreslené linky. Jednotlivé body na sebe bezprostředně navazují. Konec linky je indikován znakem ";". Grafický kurzor se vrátí zpět (v ose x) a v ose y se posune o jednu linku nahoru. Další hexadecimální kódy vykresluje další linku. Posun nahoru o jednu linku se provádí

znakem středník. Celý příkaz DEF FILL je ukončen znakem pro oddělování příkazů, což je znak "CR", končí-li příkazový řádek nebo znak ":", je-li další příkaz na řádku.

Pomocné příkazy GETP, GETX, GETY umožňují programu zjistit stav daného bodu (na který ukazuje grafický kurzor) nebo absolutní souřadnice (ve fix módu) v ose x nebo y.

Příkaz CLSG umožňuje výmaz grafické paměti. Uvede-li se u příkazu CLSG parametr (0-255), provede naplnění paměti touto hodnotou, např. CLSG 255 naplní všechny bity grafické paměti hodnotou "1", tj. rozsvítí všechny body obrazovky.

Pro ovládání kurzoru je příkaz MOVE a všechny kreslicí příkazy. Příkaz MOVE nastaví kurzor do určeného bodu. Kreslicí příkazy začínají z bodu, kde je grafický kurzor a po provedení příkazu nechají kurzor v koncovém bodě.

Pro kreslení čar se používá příkaz PLOT. U příkazu může být uvedeno více bodů, které se mají postupně propojit. Jednotlivé body se oddělují znakem středník.

Grafický Basic EXP V5.0/G obsahuje velice účinný příkaz pro práci ve fix modu, příkaz DRAW. Tento příkaz pracuje bez ohledu na nastavené měřítko. Je v podstatě jed-

noduchým grafickým jazykem pro kreslení čar a obrazců a pomocí příkazu lze zadávat více povelů pro kreslení. Na tento příkaz navazuje příkaz DRAW INPUT, který nám umožní generovat povelové řetězce podle pohybu grafického kurzoru na obrazovce, jehož pohyb se řídí tlačítky z klávesnice.

Příkazy:

Formát:

```
SCALE XMIN,XMAX,YMIN,YMAX
```

Nastavení měřítka pro zobrazování. Hodnoty mezí MIN a MAX souřadnic jsou v rastru X(0-319 bodů) a Y(0-239) bodů. Ve fix módu odpovídá každé souřadnici právě jeden bod. Měřítko může být větší než je základní rastr, ale pak několika hodnotám v měřítku bude odpovídat stejný bod. Měřítko se uplatňuje u příkazů MOVE, PLOT a AXES.

Příklad:

```
SCALE 0,319,0,239    odpovídá základnímu nastavení
                     po RUN

SCALE -1,1,-1,1
```

Formát:

```
CLSG[N]
```

Výmaz grafické paměti. Parametr N umožní zaplnit paměť hodnotou "**N**". Na parametr "N" se díváme jako na osmibitové číslo; bit,který je nastaven na "1" bude rozsvícen a který obsahuje "0" nesvítí. Zadává se jako dekadické číslo v rozmezí 0 až 255.

Příklad:

```
10 CLSG        ! výmaz paměti
11 CLSG 255    ! zaplň paměť - samé jedničky
```

Formát:

```
COLOR N [,M]
```

Nastavení módu zápisu a popřípadě i pera. Příkaz je sice **nazván COLOR, ale to neznamené, že nastavuje "barvu".** Mikropočítač ONDRA je pouze černobílý bez možnosti barevného zobrazování i bez možnosti různých odstínů šedi. Příkaz určuje, jakým způsobem se bude provádět grafické zobrazování.

N .... mód zápisu čáry:

N=0 .... netvoří se čára (pero nahoře)
N=1 .... zápis čáry je realizován funkcí "XOR"
N=2 .... zápis čáry je řízen hodnotou pera "1" ... zápis
"0" ... výmaz

N=3 .... výmaz čáry
M=PEN ... nastav pero, viz MOVE povel

Formát:

```
AXES X,Y [,PEN]
```

vymaluje osy s průsečíkem v zadaných bodech X,Y podle měřítka.

Příklad:

```
AXES 0,0
AXES 20,20,HEX(5F)
```

Formát:

```
GETP
```

Funkce "GETP" dosadí hodnotu bodu (0 nebo -1), na který ukazuje kurzor. Adresa je nastavena operací (MOVE,PLOT, LABEL..). Hodnota "0" je zhasnutý bod.

Příklad:

```
10 MOVE 0,0
20 IF GETP=0 THEN PRINT "volné políčko"
```

Formát:

```
GETX
```

je funkce, která přiřadí do proměnné hodnotu kurzoru v ose X.

Formát:

```
GETY
```

je funkce, která přiřadí do proměnné hodnotu kurzoru v ose Y.

Příklad:

```
PRINT "SOURADNICE JE:";GETX;GETY
```

Poznámka: Hodnoty souřadnic jsou ve fix módu.

Formát:

```
LABEL MX,MY;"TEXT"
```

V daném měřítku (zvětšení) MX a MY se zobrazí text jako v příkazu PRINT. Zobrazuje znaky od "␣" ... "Z" a ignoruje CR, LF. Dále zobrazuje uživatelem definované znaky (příkaz DEF CODE). Pozor! Pro příkaz LABEL obsahuje interpret vlastní generátor znaků v rastru 5 x 7 bodů. V generátoru jsou zavedeny znaky v rozsahu 20H až 5FH, tj. pouze velká písmena! Písmo (levý dolní roh rastru) je zobrazeno v místě grafického kurzoru.

Příklad:

```
LABEL 1,1;"BASIC"
MOVE 30,20:LABEL 2,2;"MIT"
```

vždy postaví hodnotu PEN = 255.

```
10 A$="TESLA"
20 LABEL 2,5;A$
```

Formát:

```
DEF CODE ADR;DATA,DATA ... DATA
```

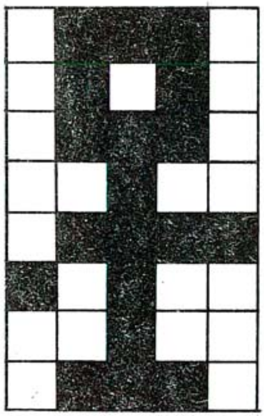
uživatelský deklarovaný znak (5byte=1 písmeno) od kódu 60H - 1 pětice, 61H druhá, 62H třetí ... ADR udává adresu, na kterou si uživatel uloží tabulku svých znaků.

Příklad:

```
  9 DEF CODE HEX(20);7F,3E,1C,C,8,FF,FF,FF,FF,FF
 10 FL$=CHR$ (HEX(61))  ! kód plného políčka
 11 ŠIPKA$=CHR$  (HEX(60))  ! kód šipky
120 LABEL 3,3; "PLNÉ POLIČKO",FL$;"ŠIPKA";CHR$
                                  (HEX(60))
```

Příklad definice uživatelského znaku - panáček

```
10 DEF CODE HEX(4300);4,E9,BF,E9,8
20 LABEL 2,1;CHR$ (96)  ! ZOBRAZENÍ UŽIVATELSKÉHO
                          ZNAKU
```



Formát:

```
FILL MX,MY  ,PEN
```

Vyplní plochu o velikosti MX,MY od kurzoru MX,MY=MAX. 255; 0 odpovídá hodnotě 256. Kurzor posouvá v ose X.

Příklad:

```
FILL 2,2,255

           * *
   ------| * *  |<-- nová poloha kurzoru
```

Formát:

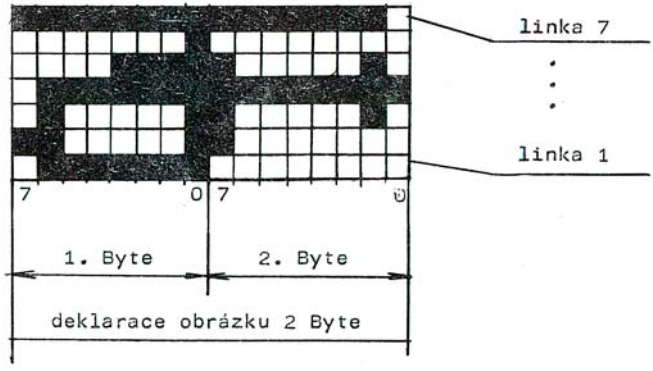
```
DEF FILL MX,MY;HEX-DATA,HEX...;MX,MY;HEX-DATA...;HEX-DATA
```

Vymaluj plochu o šířce N*8 bodů. Délka řádky je určena počtem HEX-byte mezi";". Po dosažení ";" se pokračuje v kreslení o řádek výše. Výška je dána délkou seznamu.
MX,MY-měřítko v ose X,Y (jako u "FILL")

Příklad: nakreslení obrysu vrtulníku

10 DEF FILL 3,2;7F,0;C1,80;41,82;7F,FF;F,82;1,0;FF,FE

Vysvětlivky: V ose x je prodloužení 3x, měřítko v ose y udává, že každá zadaná linka se kreslí 2x.



Formát:

```
PLOT X1,Y1[,PEN1][;X2,Y2[,PEN2]...; XN,YN [,PENN]]
```

Příkaz pro kreslení čáry. Spojí výchozí bod s bodem zadaným souřadnicemi v příkazu. Je možno zadat typ kreslené čáry (parametr PEN). Lze zadávat souřadnice více bodů, které se oddělují znakem ";".

Příklad:

```
MOVE 3,3          ! nastav počátek čáry
PLOT 20,20        ! od polohy kurzoru (3,3) do 20,20 udělá
                    čáru

PLOT 30,30;X+3,X;X-5,30   ! lomená čára
PLOT 10,10,255
```

Formát:

```
MOVE X,Y [,PEN]
```

Nastavení kurzoru od bodu o souřadnicích X,Y. Umožní nastavení pera - typ čáry.

Příklad:

```
MOVE 0,0
MOVE 0,0,255
PEN ... 0 ruší čáru ... nastavení pera
PEN ... 255 plná čára
PEN ... HEX(55) tečkovaná čára
PEN ... HEX(F5) čerchovaná čára
```

Kombinovaný grafický příkaz DRAW

Příkaz umožňuje kreslení v pevném rastru - fix módu. Je to příkaz, který umožňuje slučovat více povelů pro kreslení do jednoho příkazu. Dokonce umožňuje vnořovat povely do sebe.

Formát:

```
DRAW "seznam povelů"
```

Seznam povelů je chápán jako textový řetězec, a proto musí být uzavřen v uvozovkách. Seznam povelů obsahuje jeden nebo více povelů pro kreslení nebo přesun ukazovátka - grafického kurzoru. Povely jsou vzájemně oddělovány znakem ":" (dvojtečka). Povel může mít parametry (souřadnice - X,Y a pero - PEN), které se oddělují znakem "," čárka. Povel P připouští současný zápis více souřadnic, bodů, které jsou vzájemně odděleny od sebe znakem ";" (středník).

Povely:

M X,Y ... MOVE nastavení kurzoru na souřadnici bodu zadanou X,Y

P X,Y;X1,Y1 ... ;XN,YN ... kreslení čáry - spojí místo, kde se nachází grafický kurzor se souřadnicí

uvedenou v povelu. Je-li uvedeno více bodů, pak tyto body propojí přímkami.

S N,M ... relativní MOVE - nastaví kurzor od okamžité polohy o N bodů v ose X a o M bodů v ose Y. Posun se provádí podle znaménka (kladné vlevo, záporné vpravo).

V N,N ... relativní Plot - propojí přímkou body, které jsou dány okamžitou polohou kurzoru a bodem, který je o N bodů posunut v ose a o M v Y. Opět se uplatní znaménko posunů.

R N ... posun o N bodů od polohy kurzoru vpravo

L N ... posun o N bodů od polohy kurzoru vlevo

D N ... posun o N bodů od polohy kurzoru dolů

U N ... posun o N bodů od polohy kurzoru nahoru

C M ... jako příkaz COLOR - nastavení **způsobu kreslení**

Poznámka:

Všechny příkazy mohou být doplněny o parametr PEN, který se uvádí jako poslední oddělený čárkou. Příklad: M X,Y,PEN DRAW "M 0,50,255"

Povel pro vnořené povelové řetězce

X jméno $,,, jméno $ je jméno jiného řetězce, který má být proveden jako povely pro kreslení v příkazu DRAW. Musí být

sestaven podle pravidel pro řetězec povelů v příkazu DRAW. Vnořený povelový řetězec musí být ukončen znakem "!" (vykřičník)!!! Při provádění posloupnosti povelů se při povelu X přeruší normální posloupnost a začne se provádět nová vnořená posloupnost povelů. Po provedení vnořené posloupnosti se pokračuje v provádění původní posloupnosti.

Příklad:

```
100 A$="V 12,12: V-12,0!"
        .
        .
        .
250 DRAW "M 0,23: P 120,-3;5,122 XA$: V 56,-12"
```

Příkaz DRAW nejdříve přesune kurzor do bodu (0,23), pak provede kreslení čar (dvě čáry) a pak začne provádět vnoření řetězce - nakreslí dvě relativní čáry a pak se vrátí zpět do hlavního řetězce a nakreslí jednu relativní čáru.

Příklad pro nakreslení domku:

```
100 DUM$="M10,10:R100:U100:L100:D100:M10,110:V50,66:V50,
                                                    -66!"
110 OKNO$="R10:U15:L10:D15!"
120 DVERE$="R20,U75,L20,D75!"
200 REM KRESLENI
210 DRAW "X DUM$:M 30,50: X OKNO$ : M 50,50:X OKNO$:
                  M75,10: X DVERE$"
```

Příkaz DRAW nejlépe pochopíte, když s ním budete kreslit na obrazovce. Nejdříve začněte pouze s jedním povelem a pak si postupně přidávejte.

Příkaz pro generaci povelového řetězce

Příkaz DRAW INPUT je určen jako generátor povelového řetězce pro příkaz DRAW. Pomocí tlačítek na klávesnici pro ovládání kurzoru (na klávesnici označené šipkami) ovládáme pohyb ukazovátka v osách x a y. Po najetí do požadovaného bodu máme možnost vytvořit povel M nebo V (nastavení bodu kurzoru nebo spojení čarou).

Formát:

```
DRAW INPUT seznam
```

kde seznam je posloupnost řetězcových proměnných, do které se bude ukládat kreslená posloupnost bodů. Do jednoho řetězce lze umístit max. 120 znaků.

Povely - tlačítka pro ovládání příkazu:

šipky ... ovládají pohyb kurzoru ve směru šipek na tlačítkách

S ... příkaz pro zapamatování polohy bodu (generuje povel M)

V ... příkaz pro nakreslení čáry od posledně pamatovaného bodu a místa, kde je umístěn kurzor

CR ... ukončení zápisu do generovaného řetězce

Poznámka:

Máme-li v seznamu více řetězcových proměnných, pak automaticky po naplnění jednoho řetězce pokračuje v plnění dalšího. Protože nevíme, kolik řetězců jsme naplnili, musíme stisknout tlačítko tolikrát, kolik je nevyčerpaných řetězců. Za příkazem DRAW INPUT si zobrazte upozornění, že jste za příkazem.

Příklad:

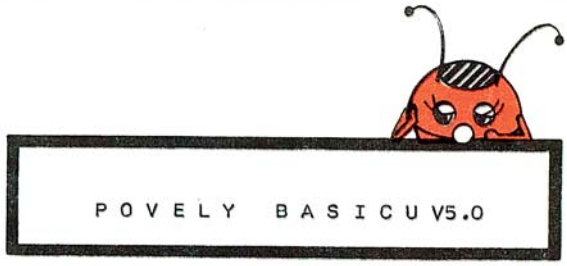
```
10 CLSG: CLEAR 500
20 DRAW INPUT A$, B$   ! budeme plnit dva řetězce
30 DRAW "M100,30:XA$:XB$:M200,60:XA$:XB$"  ! 2x zobrazíme nakreslený obrazec
```

Příkaz DRAW INPUT můžeme využít i pro jiné účely, např. pro zadání polohy pro text. Např. chceme umístit nápis DUM k našemu domečku, který jsme nakreslili příkazem DRAW. Doplníme příklad o příkazy.

```
250 DRAW INPUT P$      ! pomocný řetězec, který nám
                         umožní posunout kurzor
260 LABEL 10,8;"DUM"
```

Poznámka:

Kurzor v příkazu DRAW INPUT je zobrazen jako bod (na černém pozadí jako bílý) a naopak. Může se stát, že ho nemůžete najít, tak s ním začnete pohybovat a určitě ho brzy objevíte.



POVELY BASICU V5.0

Povely slouží k řízení a vymezení pracovních oblastí interpretu jazyka Basic V5.0. Se třemi povely jsme se již setkali v předcházející části a umožnily nám spustit program a vymazat program z paměti. Některé povely můžeme použít jako příkazy v programu.

Povely:

RUN ... povel pro spuštění programu od začátku

RUN N ... spuštění programu od čísla řádku N. Při spuštění se nastavují (nulují některé vnitřní proměnné) a proměnné v programu.

NEW ... povel pro zrušení programu a všech proměnných v paměti. Můžeme zadávat další program.

CLEAR .... povel pro smazání tabulek a rezervaci paměti.

CLEAR délka ... výmaz a rezervace zadané délky pro řetězce

CLEAR délka,RAM ... výmaz a rezervace zadané délky pro řetězce. RAM je délka, o kterou se má snížit horní hranice paměti RAM. Vytvoří se rezervovaná oblast, se kterou normální příkazy a Monitor nepracují. Např. pro programy v assembleru. Psát a číst z této oblasti můžeme příkazy PEEK, POKE a využívat ji pomocí CALL.

Př. CLEAR 255,1000

Basic vypíše na obrazovce text
"END OF RAM 51847"
Příkaz musíme použít pro rezervaci pracovní oblasti pro řetězce, pokud nám nebude stačit implicitní velikost 255B.

CONT ... pokračování v provádění programu, jestliže byl zastaven příkazem STOP a nebo byl přerušen klávesou CTRL C.

LIST ... výpis programu na obrazovce (provádí se výpis celého programu). Výpis lze přerušit klávesou CTRL C.

LIST N ... výpis od zadaného čísla řádku (N)

LIST N, počet ... výpis od zadaného čísla řádku (N) a vypíše se pouze zadaný počet řádků

@ N ... editace řádku N. Není-li uvedeno číslo řádku, pak se implicitně vybere první rádek a po zastavení programu s chybou se automaticky bere chybný řádek.

Editační klávesy: CTRL D - výmaz znaku vlevo od kurzoru

- ← - posun kurzoru o znak vlevo (šipka vlevo)
- → - posun kurzoru o znak vpravo (šipka vpravo)
- ↓ - posun po oddělovačích na řádku (oddělovače mezera, čárka, středník atd.) (šipka dolů)
- ↑ - nastavení kurzoru na druhý okraj řádku (šipka nahoru)

- CR - ukončení editace
- znak - každý jiný znak se ukládá do editovaného textu

Poznámka:
Po zadání povelu se zobrazí řádek, který chceme editovat. Pokud provedeme libovolnou editační funkci, zobrazí se řádek znovu na horním řádku obrazovky a zde vidíme, co a jak editujeme. Příkaz můžeme použít i pro kopírování řádku tak, že přepíšeme číslo řádku na nové číslo. Původní řádek zůstává zachován.

RENUM ... povel pro přečíslování programu (implicitní 10,10) provede přečíslování včetně odkazů GOTO, GOSUB. U GOSUB je jedno omezení, že přečísluje pouze řádky, které jsou uvedeny před jmény procedur. Čísla řádků uvedených za jmény se nepřečíslují!

RENUM nové čís. ... určení počátečního čísla pro nový program.

RENUM nové čís.,staré čís. ... určení počátků přečíslování programů

RENUM nové čís.,staré čís.,krok ... určení počátků přečíslování v novém programu se bude provádět s přírůstkem čísel řádků "krok". Přečíslování se provádí od řádku "staré číslo", kterému se přiděli nové číslo s přírůstkem "krok" až do konce programu.

AUTO ... automatické číslování řádků (implicitní 10,10)

AUTO čís.řád. ... autom. číslování řádků s počátečním číslem prvního řádku "čís.řád."

AUTO čís.řád.,krok ... autom. číslování řádků , další řádky jsou zvětšeny o "krok"

Automatické číslování řádků se ukončí zadáním prázdného řádku, tj. stisknutím tlačítka CR. zadávání lze rovněž ukončit tlačítkem CTRL C.

SEEK text ... vyhledání zadaného textu v programu (příkazových řádcích)

SEEK text,čís.řád. ... vyhledání zadaného textu, vyhledává se od zadaného čísla řádku. Povel vyhledá v programu libovolný text kromě znaku mezera, středník a CR. Pokud není zadané číslo

řádku, pak se provádí vyhledávání od počátku. Je-li znak (text) nalezen, pak se zobrazí nalezený řádek a můžeme použít následující povely:

- "mezera" - další vyhledávání (další výskyt zadaného textu)
- CR - ukončení povelu SEEK
- znak - každý jiný znak provede automaticky volání editoru

DELETE poč.řád.,kon.řád. ... zrušení části programu od počátečního do koncového čísla řádku včetně. Není-li zadáno koncové číslo, zruší se až do konce programu.

BYE ... ukončení práce v Basicu a návrat do monitoru

LLIST ... výpis programu na tiskárnu (viz povel LIST). Pozor, pokud nemáte připojenou tiskárnu, pak nepoužívejte tento povel.

TRON ... start trasování programu - vypisují se čísla řádků, kterými program prochází

TROFF ... konec trasování. Ukončení výpisu Tyto povely můžeme použít v programu.

TRSTEP ... trasování po krocích. Po provedení příkazového řádku se program zastaví a čeká na klávesnici. Tlačítko "mezera" zna-

mená proveď další krok a tlačítko "CR" - konec trasování.

LOAD ... zavedení programu ve znakovém tvaru do paměti. Není-li paměť prázdná (tj. je-li zde jiný program), Basic vypíše text "APPEND (Y/N)" a čeká na odpověď Y - zavede další program, N - zruší starý a zavede nový.

LOAD B ... zavedení programu v binárním tvaru (ve vnitřní předzpracované formě. Tyto programy musely být uloženy povely SAVE B nebo SAVE P). Pozor, tato forma není přenositelná mezi různými verzemi Basicu a počítači (např. ONDRA - SAPI 1)

LOAD G ... zavede do grafické paměti nový obsah

LOAD CODE ... zavede do paměti strojový kód (např. podprogramy v assembleru)

Poznámka:

Všechny příkazy LOAD a SAVE lze použít i ve tvaru se zadaným jménem souboru, podobně jako příka OPEN. Jméno souboru se zapisuje v uvozovkách.

SAVE ... zápis programu v Basicu na vnější paměť (kazetu) ve znakovém tvaru (ASCII).

SAVE B ... zápis programu ve vnitřním tvaru na kazetu

SAVE P ... zápis programu ve vnitřním tvaru na kazetu (stejně jako SAVE B), ale uložený program po zavedení do paměti (LOAD B) je chráněn proti výpisu a změnám. Tzv. "soukromý" program.

SAVE G ... uložení obsahu grafické paměti na kazetu

SAVE CODE ... uložení strojového kódu na kazetu. Po zadání příkazu je nutno zadat počáteční a koncovou adresu stejně jako u příkazu monitoru K-S. Zadání adresy je indikováno znakem "=". Zadává se v hexadecimálním tvaru.

GOTO ... jako povel lze použít i příkaz GOTO pro spuštění programu. V tomto případě se nenulují proměnné jako u příkazu RUN. (Obvykle používáme pro ladění) Podobně můžeme použít příkaz PRINT pro zjištění stavu proměnných.

Poznámky:

Příkaz, před kterým není číslo řádku, se provádí okamžitě. Číslované příkazy se umisťují dle čísel řádku do zdroj. programu.

Maximální počet znaků na instrukčním řádku je 132,vč. čísla řádku a mezer. Maximální číslo řádku ... 64999.

Počet mezer mezi číslem řádku a první instrukcí zůstane uchován

```
5 FOR J=1 to K
10 FOR I=1 to 10
20 X=J*SIN(I):Z=Z/15+X:PRINT "X=";X,"Z=";Z
30 NEXT I
35 NEXT J
```

Názvy proměnných mohou být víceznakové (první vždy písmeno), ale rozhodující jsou však pouze první d v a znaky.

Celá rezervovaná slova se nesmí užívat jako názvy proměnných.

Pro numerické proměnné a stringy je možno použít těchže názvů (např. "B" a "B$")

Přerušení chodu programu (PO "RUN") klávesou "CTRL C" (ETX).

Klíčová slova nebo jejich části nelze používat jako názvy proměnných.

Např. DEL=1 ! nepřípustné, protože je to část klíčového slova DELETE

IF A=TON ... část klíčového slova "TO"

Přerušení programu.

Při chodu programu lze program přerušit tlačítkem NMI (nemaskované přerušení), které je na levém boku skříňky počítače.

Je-li program ve stavu čtení dat z klávesnice, lze program přerušit stiskem kláves "CTRL C".

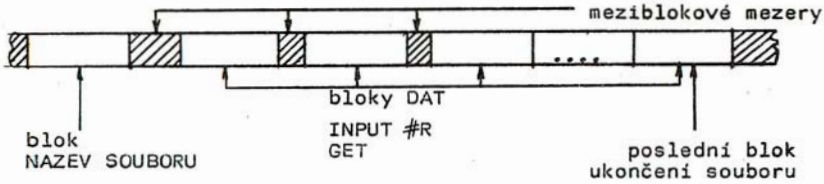
Přerušení se provede po dokončení příkazu.

Basic V5.0/G umožňuje i zobrazování znaků české abecedy v řetězcových konstantách a proměnných. Práce s češtinou je poněkud komplikována, protože interpret používá pouze sedmibitový kód pro znaky a osmý bit využívá pro svoji potřebu. Proto nelze použít přímo znaků KOI-8(čs-2, kde znaky české abecedy mají nastaven paritní bit na jedničku (osmý bit). Basic V5.0 zatím pracuje se znaky české abecedy tak, že je rozkládá na dva znaky. První znak je 7FH a na klávesnici Ondry ho zapíšeme tak, že jsme na číslicové změně a stiskneme tlačítko "G". Následující znak je pak brán jako znak české abecedy. Zde si musíme uvědomit, že při editaci jsou to dva znaky a nikoliv jeden zobrazovaný

znak. Podmínkou pro zobrazení znaků české abecedy je, že jsme do paměti si zavedli generátor českých znaků. Při editaci se znak 7FH zobrazuje jako šipka nahoru, ukazuje-li na něj kurzor. Připravuje se další verze interpretu jazyka Basic, která umožní snazší zadávání znaků české abecedy.

Manipulace se soubory dat na kazetě

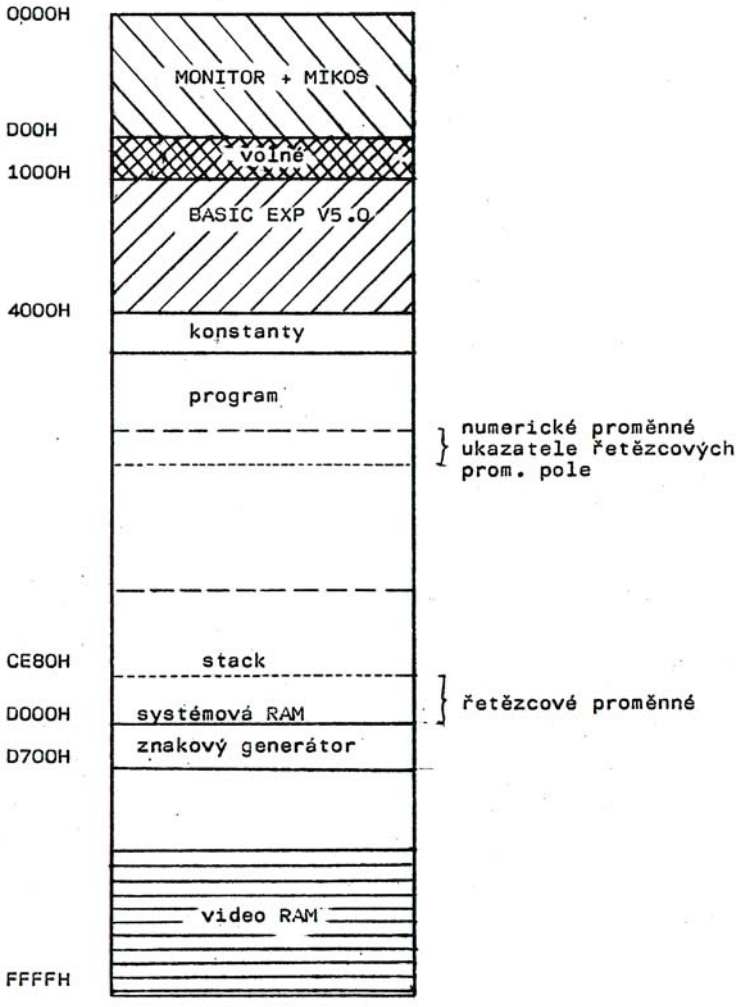
Vstupní soubory jsou takové soubory, za které tiskneme data. Vstupní soubory byly dříve vytvořeny a jsou uloženy na pásce.



Operační systém umožňuje vyhledat začátek souboru na kazetě podle jména nebo použijeme-li hvězdičkovou konvenci - dotazováním, zdali je nalezený soubor hledaným souborem. Po vyhledání souboru lze jednotlivá data zpracovávat programově:

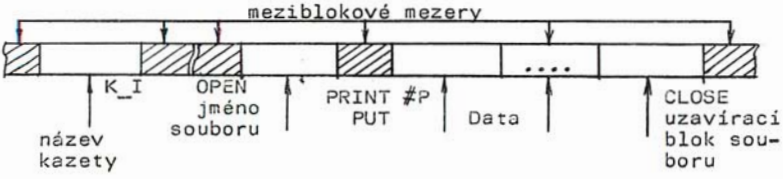
```
FIND            ! povel pro otevření vstupního souboru
INPUT#R,X,Y     ! čtení dat do proměnných X,Y
A=GET           ! čtení Byte
FIND            ! ukončení souboru
```

Rozdělení RAM pro BASIC



Vstupní soubor se uzavírá příkazem FIND, který automaticky otevře nový vstupní soubor. Dotaz na jméno vstupního souboru se provádí až při prvním požadavku na data ze vstupního souboru. Příkaz INPUT čte vstupní data po nalezení oddělovače proměnných v souboru, tj. znak "," (čárka) nebo znak "CR". Levostranné mezery se na vstupu ignorují (pro řetězce) pro čísla i pravostranné mezery.

Výstupní soubory jsou takové soubory, do kterých se zapisují data. Výstupní soubor je právě programem vytvářen. Zápis dat na pásku je sekvenční (postupný, jeden blok za druhým). Do sekvenčního souboru nemůžeme data přidávat ani je měnit. Pokud máme požadavek na změnu nebo doplnění dat, musíme data z požadovaného souboru přečíst, pak změnit nebo doplnit a vytvořit nový výstupní soubor.



Před zápisem na kazetu, tj. před otevřením výstupního souboru, je nutno najít místo na kazetě, na které se bude provádět zápis dat. Obvykle to bývá za poslední vložený soubor, aby systém mohl nalézt správné místo, zadává se jméno posledního souboru. Systém přečte uvedený soubor a nastaví se za jeho konec a je připraven pro zápis souboru. Po nastavení magnetofonu na zápis se provede zápis bloku s názvem souboru. Po zápisu dat se musí výstupní soubor uzavřít příkazem CLOSE, který zapíše ukončovací značku souboru a zapiše data z výstupní vyrovnávací paměti na pásku.

Příklad:

```
OPEN                ! otevření výstupního souboru
PRINT # P,X;",";Y   ! zápis proměnných x,y a oddělovače
PUT X1              ! zápis Byte
PRINT # P "DATA"    ! zápis řetězce, konstanty
CLOSE               ! uzavření výstupního souboru
```

Příkazy pro otevření souboru na kazetě OPEN, FIND a povely LOAD a SAVE umožňují zadání jména souboru v příkazu. Příkazy pro otevírání výstupních souborů umožňují i přidání souboru (append) bez vyhledávání místa na kazetě. Proto v Basicu vystačíme pouze s příkazy a nemusíme využívat služeb Monitoru a Mikosu v režimu Monitor.

Poznámky:

Zápis nového souboru mezi již zapsané soubory zásadně nedoporučujeme, protože se může poškodit začátek dalšího souboru. Proto příkaz OPEN žádá zadání jména posledního souboru, za který můžeme zapisovat nový soubor. Okamžitý zápis dat na kazetu bez vyhledání může při trošce nepozornosti způsobit přepsání Vašeho programu nebo cenných dat.

Hlášení chyb:

Basic EXP V5.0/G při výskytu chyby v prováděném programu provede chybové hlášení a na obrazovce monitoru přeruší provádění programu a nastaví se do příkazového režimu. Při zadání povelu EDIT, tj. "@" se přejde do editoru na řádek, který způsobil chybové hlášení.

Tvar chybového hlášení:

?XX ERROR IN N

kde "XX" je dvouznakové označení chyby dle seznamu a "N" je číslo řádku, kde byla zjištěna chyba.

Seznam chybových hlášení:

NF ... chyba v cyklu "FOR-NEXT", pro NEXT chybí FOR

SN ... syntaktická chyba, špatně zapsaný příkaz

RG ... Basic při provádění nalezl příkaz RETURN a nebylo volání GOSUB

OD ... chybí data v seznamu DATA, příkaz čtení dat READ vyčerpal i seznam a jsou požadována další data

FC ... chyba v zápisu funkce

OV ... přetečení proměnné, hodnota proměnné je mimo dovolený rozsah

OM ... přetečení paměti, nedostatek místa v paměti

US ... neznámý příkaz, odkaz na řádek, který není v programu uveden

BS ... chyba v indexaci - špatný zápis nebo přetečení indexu pole

DD ... vícenásobná deklarace pole téhož jména

/O ... dělení nulou

ID ... nepřípustný výraz

TM ... nesouhlas typů proměnných ve výrazu

OS ... přetečení vyhražené paměti pro řetězcové proměnné (příkaz CLEAR)

LS ... příliš dlouhý řetězec (více než 255 znaků)

ST ... příliš složitý řetězcový výraz

CN ... chyba v návaznosti příkazů

UF ... volání nedefinované funkce

WE ... WEND bez příkazu WHILE DO

UN ... UNTIL bez příkazu REPEAT

BE ... nalezen příkaz BEND bez příkazu BEG

M# ... příliš mnoho znaků "#" ve vzoru u příkazu USING

ND ... nejsou binární data

PA ... chyba v zadání parametrů

PE ... deklarace procedury PROC není ukončena příkazem ENDP

FE ... deklarace funkce FUNC není ukončena příkazem ENDF

Informační hlášení při provádění programu:

EXTRA IGNORED ... přebytečná data ve vstupním seznamu příkazu INPUT jsou vynechána. Na vstupu bylo zadáno více dat než požaduje příkaz INPUT. Jde o ifnormační hlášení, program pokračuje dále v činnosti.

REDO FROM START ... nesprávný vstup pro příkaz z klávesnice. Místo číslice je zadáno písmeno nebo jiný znak. Dosavadní vstup je **ignorován** a program očekává správné zadání čísla.

Slovník příkazů a instrukcí

a) Monitor a Mikos

.B	spuštění programu Basic EXP/G V5.0
.C	studený start monitoru (programový RESET)
.G	skok na zadanou adresu
.M	přesun dat v paměti
.S	prohlížení obsahu paměti,popř. změna
.X	prohlížení obsahu registrů, popř. změna
.K_C	uzavření výstupního souboru na kazetě
.K_D	výpis adresáře kazety
.K_F	vyhledání vstupního souboru
.K_I	označení (inicializace) kazety
.K_J	test a skok na začátek zavedeného programu
.K_L	zavedení binárního souboru do paměti
.K_M	manipulace s kazetou (např. převíjení)
.K_O	otevření výstupního souboru s vyhledáním
.K_A	otevření výstupního souboru bez vyhledávání
.K_S	uložení binárního souboru na kazetu
.K_T	text zavedeného programu v paměti

b) BASIC EXP/G V5.0

ABS	funkce pro výpočet absolutní hodnoty
AND	logická funkce AND ("i") - logický součin
ASC	funkce pro výpočet desítkové hodnoty ze znaku ASCII
ATN	funkce pro výpočet arc tangens
AXES	vykreslení os souřadnic
AUTO	povel pro automatické číslování
BEEP	zvuková signalizace
BEG	označení začátku bloku příkazů (musí být ukončen příkazem BEND)
BEND	ukončení bloku příkazů (musí předcházet BEG)
BYE	návrat do monitoru
CALL	volání podprogramu v assembleru
CLEAR	vymezení prostoru pro řetězcové proměnné
CHR $	funkce převodu dekadické hodnoty na ASCII znak
CLOSE	uzavření výstupního souboru
CLS	výmaz alfanumerické obrazovky
CLSG	výmaz grafické obrazovky
CODE	uložení znaků v hexadecimálním kódu do paměti
COLOR	nastavení grafického módu, popř. pera
CONT	povel k pokračování provádění příkazů, programu
COS	funkce pro výpočet kosinu
CURS	nastavení polohy alfanumerického ukazatele (kurzoru)

DATA	seznam proměnných pro příkaz READ
DEF CODE	definování uživatelského znaku pro příkaz LABEL
DEF FILL	definice a vykreslení uživatelského obrázku
DEF FNx	definice uživatelské numerické funkce
DEG ON,OFF	funkce pro nastavení stupňů nebo radiánů
DELETE	zrušení zadaných řádků v programu
DIM	deklarace rozměrů pole
DO	klíčové slovo z příkazu WHILE
DRAW	kombinovaný příkaz pro grafické kreslení v absolutních souřadnicích
DRAW INPUT	automatické generování povelového řetězce pro příkaz DRAW, pomocí tlačítek klávesnice
ELSE	klíčové slovo z příkazu IF/THEN, ON GOTO, ON GOSUB
END	ukončení programu
ENDF	ukončení deklarace funkce (musí předcházet FUNC)
ENDP	ukončení deklarace procedury (musí předcházet příkaz deklarace procedury PROC)
EXP	výpočet exponenciální funkce
FAST	povolení nebo zákaz zobrazování na obrazovce
FILL	zaplnění zadané oblasti grafické paměti
FIND	otevření vstupního souboru
FOR	příkaz cyklu FOR - TO -STEP
FRE	zobrazení počtu Byte volné paměti
FNx	volání uživatelské numerické funkce
FUNC	deklarace víceřádkové uživatelské funkce (musí být ukončen příkazem ENDF)

GET	čteni Byte z vyrovnávací paměti kazety
GETP	funkce předávající hodnotu bodu grafické paměti
GETX	funkce předávající hodnotu Xové souřadnice bodu
GETY	funkce předávající hodnotu Yové souřadnice bodu
GOSUB	volání podprogramu
GOTO	nepodmíněný skok
HEX	funkce, která převádí hexadecimální číslo na dekadické
IF	klíčové slovo z podmíněného příkazu IF/THEN/ELSE
INKEY	funkce pro vstup znaku z klávesnice
INP	vstup Byte z portu
INPUT	vstup proměnných (z klávesnice, pásky atd.)
INT	funkce zaokrouhlení čísla - převod na celočíselné
INTRP	řízení přerušovacího systému (ON,OFF)
LEFT $	přenos části řetězce zleva
LEN	funkce předávající počet znaků v řetězci
LET	přiřazovací příkaz (klíčové slovo není povinné)
LABEL	grafický příkaz pro zobrazení ASCII znaků a uživatelem definovaných znaků (DEF CODE)
LIST	výpis programu na obrazovce
LLIST	výpis programu na tiskárně
LOAD	zavedení programu v Basicu (ASCII) do paměti
LOAD B	zavedení binárního programu v Basicu do paměti
LOAD CODE	zavedení programu ve strojovém kódu do paměti
LOAD G	zavedení grafické paměti z pásky do paměti

LOG	funkce pro výpočet logaritmu
MID $	funkce vytvoření podřetězce z řetězce
MODE	nastavení modu zobrazování
NEW	počáteční nastavení Basicu - pro vstup nového programu
NEXT	klíčové slovo ukončující cyklus FOR
NOT	logická funkce - negace
MOVE	nastavení grafického ukazatele na zadané souřadnice
OFF	klíč
ON	klíč
ON GOSUB	přepínač volání podprogramů
ON GOTO	přepínač nepodmíněných skoků
OPEN	otevření výstupního souboru
OR	logická funkce ("nebo") - logický součet
OUT	výstup Byte na port
PEEK	funkce, která předá hodnotu Byte v paměti
PLOT	nakreslení čáry na obrazovce
POKE	uložení hodnoty do Byte v paměti
POS	funkce, která předá číslo první volné pozice (místo zobrazení nebo pozice pro tisk znaku na řádku)
PRINT	výstup proměnných, konstant na zadané **výstupní** zařízení (obrazovka, tiskárna, páska atd.)
PROC	definice víceřádkové proceduy (podprogramu). Definice procedury musí být ukončena příkazem ENDP.

PTR	funkce, která předává adresu proměnné v paměti
PUT	zápis Byte do výstupní vyrovnávací paměti (pásky)
READ	čtení proměnných z paměti, ze seznamu DATA
RENUM	povel pro přečíslování řádků programu
RETURN	příkaz pro ukončení programu
RESTORE	nastavení ukazatele v seznamu DATA
REPEAT	začátek cyklu REPEAT - UNTIL, cyklus není-li splněna podmínka
RIGHT$	přenos části řetězce zprava
RND	funkce generující náhodné číslo 0 až 1
REM	poznámka
RUN	spuštění programu v Basicu
SAVE	uložení programu v Basicu (ASCII) na pásku
SAVE B	uložení programu v Basicu v binární formě na pásku
SAVE CODE	uložení strojového kódu (zadaného obsahu paměti) na kazetu
SAVE G	uložení grafické paměti na kazetu
SAVE P	uložení soukromého programu v Basicu na pásku, zavádí se do paměti příkazem LOAD B
SGN	funkce, která předává hodnotu znaménka (-1,0,+1)
SCALE	zadání měřítek pro osy x a y v grafických příkazech
SEEK	povel pro vyhledání zadaného textu ve zdrojovém textu programu
SIN	funkce pro výpočet sinus

SPC	funkce pro výstup mezer (pro příkaz PRINT)
SQR	funkce výpočtu odmocniny
STOP	přerušení programu (lze pokračovat příkazem CONT)
STEP	klíčové slovo z příkazu cyklu FOR-TO-STEP pro nastavení kroku cyklu
STR $	funkce, která vrací řetězec, reprezentující numerickou hodnotu, bez koncové mezery
TAB	funkce pro nastavení pozice výstupu znaků v příkazu PRINT
TAN	funkce, která vrací hodnotu tangens
THEN	klíčové slovo podmíněného příkazu (IF-THEN-ELSE)
TROFF	povel pro vypnutí trasování programu
TRON	povel pro zapnutí trasování programu
TR STEP	trasovací příkaz po krocích
UNTIL	klíčové slovo ukončující cyklus REPEAT
USING	klíčové slovo z příkazu PRINT, umožňující formátovaný výstup čísla podle zadaného vzoru
USR	volání uživatelského programu v assembleru
VAL	funkce, která převádí numerický řetězec na číslo ve vnitřním tvaru
VAR	deklarace proměnných
WAIT	čekání na výskyt definovaného znaku na portu
WEND	příkaz ukončující cyklus WHILE
WHILE	příkaz cyklu, který se provádí, pokud je splněna podmínka

WND	příkaz pro vytvoření a přepnutí "okna"
XOR	logická funkce neekvivalence (exclusive OR)
!	klíčový znak uvozující poznámku
?	klíčový znak pro zkrácení zápisu klíčového slova PRINT. Do paměti se ukládá celé klíčové slovo PRINT
@	vyvolání EDITORu Basicu
&	volání víceřádkové funkce

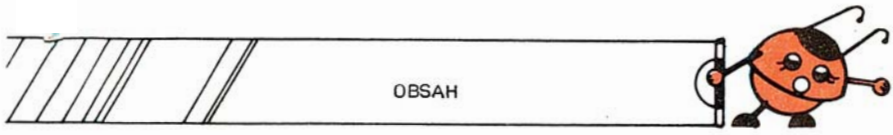
Řízení periferií

PRINT # C ... CONSOLE
L ... LPT
P ... PUNCH
H ... HARDCOPY
B ... BLANK

INPUT # C ... CONSOLE
R ... READER

Povely grafického jazyka DRAW

M ... MOVE
P ... PLOT
S ... SET
V ... VECTOR
L ... LEFT
U ... UP
R ... RIGHT
D ... DOWN
C ... COLOR
X ... externí povel



PŘÍLOHA

Basic EXP V5.0/G umožňuje zadávání vybraných klíčových slov zjednodušeným způsobem - stisknutím dvou tlačítek se vygeneruje klíčové slovo.

klávesa	klíčové slovo	funkce
A	AUTO	automatické číslování řádků
B	BEG	začátek bloku
C	CLOSE	uzavření výstupního souboru
D	DATA	definice konstant pro READ
E	ELSE	klíčové slovo v příkazech IF,ON GOTO
F	FOR	příkaz cyklu
G	GOSUB	volání podprogramu
H	SEEK	povel pro vyhledání textu v programu
I	INPUT	vstup dat ze vstupního zařízení
J	RUN	spuštění programu
K	DELETE	zrušení programových řádků
L	LIST	výpis programu na obrazovce
M	MOVE	grafický příkaz pro nastavení kurzoru
N	NEXT	klíčové slovo cyklu FOR
O	OPEN	otevření výstupního souboru
P	PROC	definice procedury
Q	BYE	ukonč.činnosti Basicu a návrat do Monitoru
R	RETURN	klíčové slovo pro návrat z podprogramu

klávesa	klíčové slovo	funkce
S	STEP	klíčové slovo cyklu, trasování
T	THEN	klíčové slovo podmíněného skoku
U	USING	formátování čísel při výstupu v PRINT
V	FIND	otevření vstupního souboru
X	DRAW	kombinovaný grafický příkaz
Y	LABEL	grafický příkaz pro výstup textu
Z	PLOT	grafický příkaz pro kreslení čáry

Řády

HEX	X	X0	X00	X000
0	0	0	0	0
1	1	16	256	4096
2	2	32	512	8192
3	3	48	768	12288
4	4	64	1024	16384
5	5	80	1280	20480
6	6	96	1536	24576
7	7	112	1792	28672
8	8	128	2048	32768
9	9	144	2304	36864
A	10	160	2560	40960
B	11	176	2816	45056
C	12	192	3072	49152
D	13	208	3328	53248
E	14	224	3584	57344
F	15	240	3840	61440

Převodní tabulka hexadecimálního čísla na dekadické číslo

Řády

DEC	X	X0	X00	X000	X0000
0	0	0	0	0	0
1	1	A	64	3E8	2710
2	2	14	C8	7D0	4E20
3	3	1E	12C	BB8	7530
4	4	28	190	FA0	9C40
5	5	32	1F4	1388	C350
6	6	3C	258	1770	EA60
7	7	46	2BC	1B58	11170
8	8	50	320	1F40	13880
9	9	5A	384	2328	15F90

Převodní tabulka dekadických čísel na hexadecimální čísla

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Převodní tabulka hexadecimálních a dekadických čísel

Použití: převod hexa na dekadické- řádky jsou pro horní půlbyte, sloupce pro dolní.
Př. A6H ... vybereme řádek "A" a sloupec "6" a najdeme odpovídající číslo 166 /dekadicky/.
Převod dekadických čísel na hexa - v tabulce nalezneme dekadické číslo a řádek nám udává horní půlbyte a sloupec dolní: Př. číslo 138 = 8AH.

	písmena				ČS - písmena				znaky			čísla			CTRL
	V		M		V		M								
	40H		60H		E0H		C0H								
K	H	D	Z	D	Z	D	Z	D	Z	H	D	Z	H	D	H
A	01	65	a	97	Á	225	á	193	-	2D	45	-	2D	45	01
B	02	66	b	98					?	3F	63	?	3F	63	02
C	03	67	c	99	Č	227	č	195	:	3A	58	:	3A	58	03
D	04	68	d	100	Ď	228	ď	196	=	3D	61	=	3D	61	04
E	05	69	e	101	É	229	é	197	#	23	35	3	33	51	05
F	06	70	f	102	Ř	230	ř	198	^	5E	94	~	7E	126	06
G	07	71	g	103					_	5F	95	DEL	7F	127	07
H	08	72	h	104	Ů	232	ů	200	<	3C	60	<	3C	60	08
I	09	73	i	105	Í	233	í	201	(	28	40	8	38	56	09
J	0A	74	j	106	Ů	234	ů	202	>	3E	62	>	3E	62	0A
K	0B	75	k	107	Ľ	235	ľ	203	[	5B	91	{	7B	123	0B
L	0C	76	l	108	Ľ	236	ľ	204	]	5D	93	}	7D	125	0C
M	0D	77	m	109	Ö	237	ö	205	.	2E	46	.	2E	46	0D
N	0E	78	n	110	Ň	238	ň	206	,	2C	44	,	2C	44	0E
O	0F	79	o	111	Ó	239	ó	207	)	29	41	9	39	57	0F
	50H		70H		F0H		D0H								
P	00	80	p	112	Ô	240	ô	208	@	40	64	0	30	48	10
Q	01	81	q	113	Ä	241	ä	209	!	21	33	1	31	49	11
R	02	82	r	114	Ř	242	ř	210	$	24	36	4	34	52	12
S	03	83	s	115	Š	243	š	211	+	2B	43	+	2B	43	13
T	04	84	t	116	Ť	244	ť	212	%	25	37	5	35	53	14
U	05	85	u	117	Ú	245	ú	213	'	27	39	7	37	55	15
V	06	86	v	118					;	3B	59	;	3B	59	16
W	07	87	w	119	É	247	é	215	"	22	34	2	32	50	17
X	08	88	x	120	À	248	à	216	/	2F	47	/	2F	47	18
Y	09	89	y	121	Ý	249	ý	217	&	26	38	6	36	54	19
Z	0A	90	z	122	Ž	250	ž	218	*	2A	42	*	2A	42	1A
⎵	20	32													
↵	0D	13													1D
↓	0A	10													1F
↑	1A	26													
←	08	08													
→	18	24													

Vysvětlivky: V,M ... velká,malá písmena
K... klávesnice, H... hexačíslo, D... dekadické číslo, Z ... znak
Hexadec. kód znaku dostaneme sečtením hexačísla /2sl./ s číslem v záhlaví. Př. kód písmene "U" =05 +50=55H

Tabulka přiřazení znaků na klávesnici.

	DEC	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
BIN	DEC	HEX	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0000	00	00	NUL	DLE	SP	0	@	P	`	p					´	ô	´	Ô
0001	01	01	SOH	DC1	!	1	A	Q	a	q					á	ă	Á	Ă
0010	02	02	STX	DC2	"	2	B	R	b	r						ř		Ř
0011	03	03	ETX	DC3	#	3	C	S	c	s					č	š	Č	Š
0100	04	04	EOT	DC4	¤	4	D	T	d	t					ď	ť	Ď	Ť
0101	05	05	ENQ	NAK	%	5	E	U	e	u					ě	ú	Ě	Ú
0110	06	06	ACK	SYN	&	6	F	V	f	v					ŕ		Ŕ	
0111	07	07	BEL	ETB	'	7	G	W	g	w						é		É
1000	08	08	BS	CAN	(	8	H	X	h	x					ű	à	Ű	À
1001	09	09	HT	EM	)	9	I	Y	i	y					í	ý	Í	Ý
1010	10	0A	LF	SUB	*	:	J	Z	j	z					ů	ž	Ů	Ž
1011	11	0B	VT	ESC	+	;	K	[	k	{					ĺ		Ĺ	
1100	12	0C	FF	FS	,	<	L	\	l	|					ľ	ˇ	Ľ	
1101	13	0D	CR	GS	-	=	M	]	m	}					ő		Ő	˙
1110	14	0E	SO	RS	.	>	N	^	n	~					ň		Ň	
1111	15	0F	SI	US	/	?	O	_	o	DEL					ó		Ó	

Kódová tabulka KOI-8čs-2