ATARI

TURBO BASIC

SVAZARM HODONÍN
ZO
VÝPOČETNÍ TECHNIKY

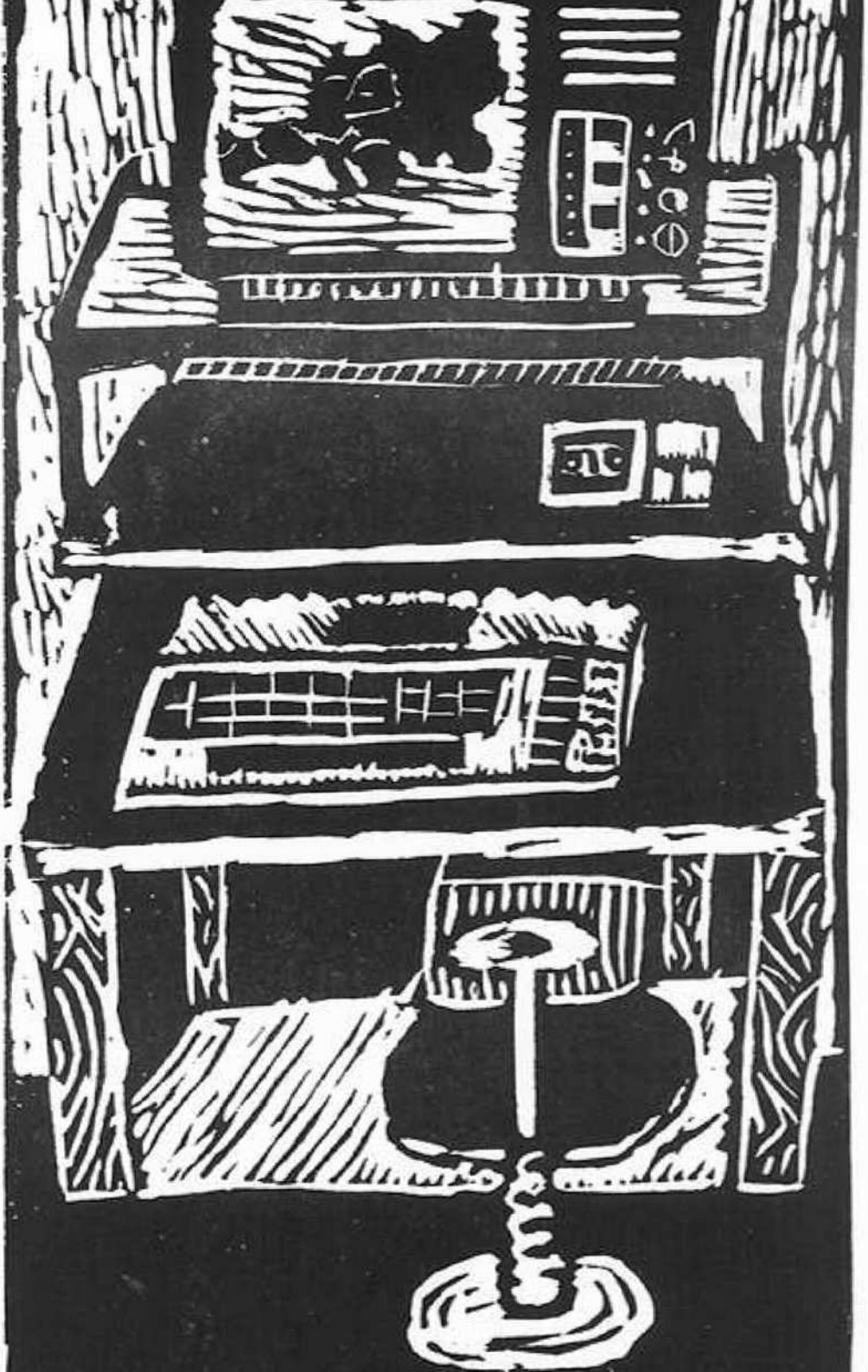
TURBO BASIC

ISBN 80– 900047– 1–7

OBSAH

OBSAH 2
PŘEDMLUVA 5
ÚVOD 6
1. 1 Program a programovací jazyk 6
1. 2 Zápis syntaxe 7
1. 3 Dialog s počítačem 7
1. 5 Organizace programu 10
1. 6 Zkratky 10
2 ZÁKLADNÍ POJMY 11
2. 1 Abeceda jazyka 11
2. 2 Identifikátory 11
2. 3 Konstanty 12
2. 4 Návěští 14
2. 5 Klíčová slova 15
3 DEKLARACE PROMĚNNÝCH 16
3. 1 Aritmetické proměnné 16
3. 2 Indexované proměnné 16
3. 3 Deklarace znakových proměnných 18
3. 4 Rušení deklarací 18
4 VÝRAZY 19
4. 1 Aritmetické výrazy 19
4. 2 Relační výrazy 22
4. 3 Znakové výrazy 24
4. 4 Funkce 25
5 JEDNODUCHÉ PŘÍKAZY 28
5. 1 Přiřazovací příkaz 29
5. 2 Komentáře 31
5. 3 Příkazy zastavení chodu programu 31
5. 4 Adresové příkazy 32
6 STRUKTUROVANÉ PŘÍKAZY 34
6. 1 Složený příkaz 35
6. 2 Podmíněné příkazy 35
6. 3 Příkazy cyklu 38
6. 4 Podprogramy 42
7 SKOKOVÉ PŘÍKAZY 44
7. 1 Nepodmíněné skoky 45
7. 2 Podmíněné skoky 45
7. 3 Skoky z podprogramů 46
8 PŘÍKAZY VSTUPU/VÝSTUPU 47
8. 1 Soubory 48
8. 2 Příkazy V/V řízené seznamem 50
8. 3 Bytově orientované V/V příkazy 53
8. 4 Vnitřní soubor 56
8. 5 Speciální V/V příkazy 58

9 PŘÍKAZY GRAFIKY 60
9. 1 Základní pojmy 60
9. 2 Příkaz GRAPHICS 62
9. 3 Přípravné příkazy 66
9. 4 Výkonné příkazy grafiky 69
10 ZVUKOVÉ PŘÍKAZY 72
11 SYSTÉMOVÉ PŘÍKAZY 73
11. 1 Příkazy V/V programů 74
11. 2 Příkazy tvorby programu 76
11. 3 Výkonné příkazy systému 78
11. 4 Speciální příkazy systému 79
11. 5 Diskové příkazy systému 81
11. 6 Systémové klávesy 82
12 CHYBÁŘ 83



Vážení přátelé,

dostáváte do rukou příručku, která Vás povede popisem programovacího jazyka Turbo Basic, ve verzi pro osobní počítače ATARI 800/130/XL/XE.

Příručka Vám může pomoci naučit se dobře používat Turbo Basic během krátké doby intenzivního studia. Sestavování vlastních programů, které je mnohem obtížnější, než získání znalostí o programovacím jazyku, však vyžaduje dlouhodobější praktickou činnost. Tato publikace Vám je tedy dalším příspěvkem naší ZO Výpočetní technika Svazarm Hodonín k rozšíření Vašich znalostí a ke zkvalitnění využití Vašeho počítače.

Ing. Josef Januška

1 ÚVOD

Tato příručka je určena těm uživatelům počítače ATARI, kteří se chtějí zdokonalit v programování a tím i lépe využít možnosti svého počítače. Příručka vám umožní nejen zvládnout programovací jazyk Turbo Basic, ale i hlouběji pochopit základní principy programování. Není tedy chápána jen jako pomůcka popisující Turbo Basic, ale i jako návod k tvorbě programů.

Turbo Basic odstraňuje největší nedostatek počítačů ATARI řady 800/130, kterým je implementovaný interpretor Basic. Atari Basic svými vlastnostmi neumožňuje efektivně využít možností vašeho počítače. Turbo Basic patří k moderním interpretorům jazyka Basic, které tvoří přechod mezi Basicem a vyššími programovacími jazyky. Přitom rychlost provádění příkazů je několikanásobně vyšší než u Atari Basicu.

Příkazy Atari Basic jsou podmnožinou množiny příkazů Turbo Basicu. Proto programy vytvořené v Atari Basicu je možné spouštět i v Turbo Basicu.

1. 1 Program a programovací jazyk

Program a programovací jazyk jsou spojeny s řešením problémů na počítači. Klíčovým pojmem řešení problémů na počítači je algoritmus. Algoritmus určuje způsob zpracování vstupních informací na výstupní. Algoritmus popisuje takovou posloupnost akcí (příkazů), kterým se zpracování informací provádí.

Program je převod algoritmu do formy, kterou je schopen počítač realizovat. Každý počítač je vybaven sadou strojových instrukcí, které lze zapsat do programu ve strojovém kódu. Zápis i velmi jednoduchých algoritmů do programu ve strojovém kódu je velmi obtížný a formulace příkazů je vzdálená od akcí, které chceme vykonat. Proto většina programů je zapsána v programovacích jazycích, jejichž příkazy překladače nebo interpretory převedou do strojového kódu. Basic patří mezi nejjednodušší programovací jazyky, které umožňují začátečníkům poměrně rychle zvládnout základy programování.

Turbo Basic, na rozdíl od jiných interpretorů jazyka Basic, umožňuje psát programy obdobným způsobem jako programy ve vyšších programovacích jazycích. Především možnost tvorby programových struktur je velkou výhodou Turbo Basicu a jeho zvládnutí vám podstatně usnadní přechod k programování ve vyšších programovacích jazycích (zejména v jazyku PASCAL).

Turbo Basic je tedy vhodným programovacím jazykem pro začátečníky, neboť jim umožňuje získat návyky a zvládnout příkazy, které mohou využít v další programátorské praxi.

1. 2 Zápis syntaxe

Každý jazyk, který národy světa používají, má stanovena pravidla pro jeho zápis. Jsou to mluvnická pravidla, se kterými se každý z nás setkává od prvé třídy. Programovací jazyky se řídí přísně při zápisu programu obdobnými pravidly, kterými je programovací jazyk definován. Souboru těchto pravidel se říká SYNTAX jazyka.

SYNTAX jazyka je množina pravidel určující přípustné způsoby a formy zápisu programu.

V příručce se budeme držet pravidel, kterými je popisována syntax vyšších programovacích jazyků. Většina učebnic používá k popisu syntaxe programovacího jazyka tzv. Backus–Naurovu formu (BNF), která používá tato významová pravidla:

<xxx> specifikovaný (určený) symbol jazyka.
Např.: <abeceda>, <identifikátor>

| svislá čára v zápisu syntaxe informuje o tom, že je možno při zápisu programu volit z několika možných alternativ.
Např.: |<písmeno>|<číslice>|<speciální znak>|

[ ] příkazy a parametry uvedené v hranatých závorkách jsou při zápisu programu nepovinné, ale upřesňují nebo rozšiřují možnosti.
Např.: LIST [od,do]

... tečky v zápisu syntaxe vyjadřují posloupnost nebo opakování.
Např.: |0|1|...|9|

(≠) Z tiskových důvodů se v příručce tímto znakem nahrazuje #

1. 3 Dialog s počítačem

Práce uživatele s moderními počítači se provádí dialogovým způsobem tzn. uživatel zadává z klávesnice příkazy a počítač je vykonává. Dialog s ATARI 800/130 se realizuje prostřednictvím rezidentního souboru, který jako vstupní zařízení používá klávesnici a výstupní zařízení TV obrazovku (monitor).

Příkazy píšeme do dialogového řádku, tj. do řádku, ve kterém se nachází kurzor. Přitom si musíme uvědomit, že délka dialogového řádku není totožná s délkou řádku na TV obrazovce. Do řádku na TV se může zobrazit max. 40 znaků, ale do dialogového řádku až 120 znaků. To znamená, že dialogový řádek se může zobrazit až do tří TV řádků.

Dialogový řádek ukončíme stlačením klávesy RETURN. Po stlačení této klávesy počítač provede nejprve vyhodnocení začátku dialogového řádku. Jestliže na začátku dialogového řádku je posloupnost číslic, pokládá tento řádek za programový a uloží jej do uživatelské oblasti paměti počítače. Není-li na začátku dialogového řádku číslo, vyhodnotí řádek jako příkazový a postupně vykonává příkazy, které jsou v něm uvedeny.

syntax:
<dialogový řádek>:=|<příkazový řádek:>|<programový řádek:>|
<příkazový řádek:>:=<příkaz>[:<příkaz>...]

Příklad: READY
PRINT "tento radek je prikazovy":A=5:PRINT A (RETURN)
tento radek je prikazovy
5
READY
120 PRINT "tento radek je programovy":A=5:PRINT A
130 PRINT "dalsi radek programu":INPUT "X=";X
RUN (spuštění programu)

Po provedení příkazového řádku, tedy příkazů, které byly v dialogovém řádku uvedeny, vypíše počítač READY a očekává další příkazy. V případě chybného zadání příkazů odpoví ERROR a inverzně zobrazí místo, kde nalezl chybu.

Po zapsání programového řádku a jeho uložení do paměti pouze nastaví kurzor na nový řádek a očekává další příkazy. To znamená, že odpověď READY dává počítač pouze po vykonání konkretních příkazů.

Poznámka:
Pokud jsou v dalším textu uvedeny příklady, je možné si je pomocí příkazového řádku ověřit. Musíme však dodržet pravidlo, že na začátku příkazového řádku musí být klíčové slovo výkonného příkazu nebo identifikátor proměnné.

Příklad: READY
A=1.5+1.7+0.3-0.8:B=A*2:PRINT A,B,B+3
2.7 5.4 8.4
READY
FOR I=1 TO 3:PRINT I*I:NEXT I
1
4
9
READY

Z příkladů je vidět, že příkazový řádek nám umožňuje zápis jednoduchých algoritmů a ověřování správné funkce příkazů.

1. 4 Organizace programu

I když na řešení stejného algoritmu můžeme sestavit více programů, každý z nich musí být organizován v určitém smyslu pevným způsobem, který je dán syntaxí jazyka. Program v Turbo Basicu je vhodné organizovat následujícím způsobem:

1) Komentář obsahující základní informace o programu.
2) Definice konstant.
3) Deklarace proměnných.
4) Přiřazení počátečních hodnot proměnným.
5) Příkazová část hlavního programu.
6) Deklarace procedur
7) Vnitřní soubor DATA

```
  0 REM *PGM=PRIKLAD,DATUM=10.01.89*
  5 REM ------------------------------------
  6 MAX=5: STO=100:REM definice konstant
 11 REM deklarace promennych
 12 DIM MATICE(MAX,MAX),VEKTOR(MAX+1)
 20 REM ------------------------------------
 22 RESTORE ≠POC.HODNOTY
 23 FOR I=1 TO MAX+1
 24      READ POM: VEKTOR(I)=POM
 25 NEXT I
 30 REM ------------------------------------
 31 ≠HLAVNI.PROGRAM .
    ...
 99 END
100 REM ------------------------------------
101 PROC PRUMER: REM deklarace procedury
    ...
150 ENDPROC
200 ≠POC.HODNOTY:REM vnitrni soubor DATA
201 DATA 1.5,-1.3,12,0,7.5,22
```

Příkazy programu jsou prováděny postupně jeden za druhým tak, jak byly zapsány do programových řádků, pokud pořadí není některým příkazem změněno. Změnu pořadí provádění příkazů realizují strukturované a skokové příkazy.

1. 4. 1 Programový řádek

Programový řádek je základním stavebním kamenem každého programu v Turbo Basicu. Program je tvořen posloupností programových řádků. Přitom číslo programového řádku určuje jeho místo v programu. Programový řádek se skládá z čísla řádku a příkazové části, která je tvořena jedním nebo více příkazy oddělenými dvojtečkami.

syntax:

<programový řádek>:=<číslo řádku><příkazová část>
<číslo řádku>:=|0|1|...|32767|
<příkazová část>:=<příkaz>[:<příkaz>...]

Příklad:

```
130  X=5
140  Y=X*5:Z=Y+X*2: PRINT X,Y,Z
```

Poznámka:
Program v Turbo Basicu lze zapisovat i malými nebo inverzními písmeny. Turbo Basic automaticky tato písmena převede na velká (s vyjímkou znakových konstant).

Příklad:

```
READY
120  a=5:print a, "znakova konstanta"
130  b=a*9
list
```

1. 5 Zkratky

Při popisu jazyka a zápisu jeho syntaxe se často opakují stejné symboly. Proto k zápisu nejčastěji používaných symbolů budeme používat následující zkratky.

: čř:	– číslo řádku
: id:	– identifikátor
: ids:	– identifikátor souboru
: ap:	– aritmetická proměnná
: ip:	– indexovaná proměnná
: $p:	– znaková proměnná
: ac:	– aritmetická konstanta
: $c:	– znaková konstanta
: av:	– aritmetický výraz
: rv:	– relační výraz
: $v:	– znakový výraz
: af:	– aritmetická funkce
: $f:	– znaková funkce

: adr:	– adresa paměti
: zn:	– znaménko |+|–|
: vvz:	– V/V zařízení
V/V	– vstupní nebo výstupní
TB	– Turbo Basic
≠	– z tiskových důvodů je znak (mříž) nahrazen tímto znakem.

2 ZÁKLADNÍ POJMY

2. 1 Abeceda jazyka

Každý programovací jazyk, tedy i TB, využívá množinu písmen, číslic a speciálních znaků, které tvoří jeho abecedu. Oproti jiným programovacím jazykům je abeceda TB rozšířena o grafické znaky, které má počítač Atari k dispozici. Celkem tvoří abecedu jazyka 256 znaků, které jsou definovány kódem ATASCII.

syntax:
<abeceda>:=|<písmeno>|<číslice>|<spec. znak>|<graf. znak>|

Na tomto použití pravidel BNF je vidět, že definujeme symbol :abeceda:, který se skládá ze čtyř podmnožin. Následující zápis vám osvětlí použití dalších pravidel zápisu syntaxe pomocí BNF.

syntax:
<písmeno>:=|A|B|...|Z|a|b|...|z|–|
<číslice>:=|0|1|...|9|
<spec. znak>:=|+|–|*|/|"|.|,|;|(|)|...

Poznámka:
Počítač Atari umožňuje definovat vlastní abecedu. Znaky této vlastní abecedy se mohou od standartních lišit při zobrazení, ale jejich kód zůstává stejný, jako kód standartních znaků.

2. 2 Identifikátory (: id:)

Při tvorbě programu dáváme určitým objektům (nejčastěji proměnným) určitá jména. Identifikátor v TB je tvořen posloupností písmen a číslic, přičemž první musí být písmeno.

syntax:
< identifikátor > := < písmeno > '| < písmeno > | < číslice > |...

Příklad:

```
130   X=5: PRUMER=Y/X
140   A99=0: UHEL.ALFA=360-N*72
150   RETEZ $="text"
```

Ze zápisu syntaxe je vidět, že k identifikaci objektu postačuje jedno písmeno. Další písmena a číslice jsou nepovinné. Přesto je však vhodné používat takové : id:, které jasně komentují, o jaký objekt jde. TB dokonce umožňuje používat i víceslovné : id:, pomocí spojovacího znaku „.“
Příklad:
UHEL.ALFA,PRUMER.TRIDY

Poznámka:
Pokud při prvním použití : id: (deklaraci) v programu počítač oznámí chybu, zkontrolujte, zda posloupnost znaků na začátku : id: není totožná s některým z klíčových slov. Chybu odstraníte změnou posloupnosti znaků na začátku : id:.
Příklad:
DELKA = 10 – začátek : id: je totožný s klíčovým slovem DEL

2. 3 Konstanty

Konstantami rozumíme v TB číselné hodnoty a znakové řetězce, které zapisujeme přímo do programu. Konstanty jsou tedy objekty, které během chodu programu nemění svoji hodnotu.
syntax:
<konstanta>:=|<aritmetická konstanta>|<znaková konstanta>|

2. 3. 1 Aritmetické konstanty (: ac:)

Aritmetická (číselná) konstanta, je číslo které je zapsáno v programu. Zápis čísla lze realizovat čtyřmi různými způsoby.
syntax:
<aritmetická konstanta>:=|<číslo>|<předdefinovaná konst>|
<číslo>:=|<celé číslo>|<racionální číslo>|<hexadec. číslo>|
|<číslo v semilogaritmickém tvaru>|
<předdefinovaná ar. konstanta>:=|%0|%1|%2|%3|

Při zápisu : ac: i při vstupu číselných hodnot si musíme uvědomit, s jakým rozsahem číselných hodnot může TB počítat a jaká je přesnost při vyhodnocení číselných informací.

Rozsah aritmetických hodnot si znázorníme na číslené ose:

– 9.99999999*10^97 0 9.99999999*10^97

–1*10^ – 98 1*10^ – 98

syntax:
<rozsah hodnot>:=|0|<zn>10^ – 98|...|<zn>9.99999999*10^97|

Přesností rozumíme maximální počet platných číslic ar. hodnoty. TB má maximálně v ar. hodnotě 9 platných číslic. Pokud ar. hodnota má větší počet číslic, provádí TB zaokrouhlení na 9 číslic. Způsob zaokrouhlování je vidět z příkladu:

Příklad:
A=1.234567850: B=1.234567849: PRINT A,B
1.23456785 1.23456784

A) Celé číslo – je tvořeno znaménkem a posloupností číslic. U kladného čísla není třeba znaménko uvádět.

syntax:
<celé číslo>:=[<zn>]<číslice>[<číslice>...]
<znaménko>:=|[+]|–|
<rozsah celých čísel>:=|–999999999|...|000000000|

Příklad:
125, –9876, 0, 1989, –912345678

B) Racionální číslo – je tvořeno znaménkem a posloupností číslic, mezi kterými je desetinná tečka.

syntax:
<racionální číslo>:=[<zn>]<celé číslo>.<celé číslo>
<rozsah racionálních čísel>:=|–99999999.9|...|99999999.9|

Příklad:
123.45, –0.00023, –12345678.9, 9.87654321

C) Hexadecimální číslo – je speciální zápis : ac: používající šestnáctkovou číselnou soustavu. Hexadec. čísla nelze používat v příkazech vstupu; při výstupu se hexadec. číslo převede na decimální.

syntax:
<hexadec.číslo>:=[<zn>]$<hex.číslice>[<hex.číslice>...]
<hexadec. číslice>:=|0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|

Příklad:
$A1C6, $BF9 (pouze celá čísla)

D) Číslo v semilogaritmickém tvaru – je univerzálním způsobem zápisu číselné hodnoty, kterým je možno zapsat jakékoliv číslo z rozsahu aritmetických hodnot. Zápis čísla v semilogaritmickém tvaru je tvořen mantisou, E a exponentem (charakteristikou).

syntax:
<číslo semilog.tvaru>:=[<zn>]<mantisa>E[<zn>]<exponent>
<mantisa>:=<racionální číslo>
<exponent>:=<celé číslo>
–kde rozsah čísel exponentu je –98 ... 97

Vyhodnocení čísla v semilogaritmickém tvaru:
<mantisa>*10^<exponent>

Příklad:
– 123.456789E23, 0.123E – 3, 675E – 12, 12.5E5

2. 3. 2 Znakové konstanty (: $c:)

Znaková konstanta je tvořena posloupností znaků, která je ohraničena uvozovkami. Znakové konstanty tvoří tzv. znakové řetězce. Znak je dán kterýmkoliv prvkem z abecedy počítače s vyjímkou uvozovek. Délka znakové konstanty je omezena délkou programového řádku, ve kterém je konstanta zapsána.

syntax:
<znaková konstanta>:="<znak>[<znak>...]"
<znak>:=<prvek abecedy>

Příklad:
"Jmeno Prijmeni 100%","XO=", "60.STUPNU,70.STUPNU"

Poznámka:
Pokud potřebujeme do znakové konstanty zařadit uvozovky, potom musíme do znakového řetězce konstanty vložit tzv. zdvojené uvozovky.
Př.: "tak zvaná " "silná ...."

2. 3. 3 Definice konstant

Jestliže se v programu často vyskytuje nějaká konstanta, je vhodné ji definovat na začátku programu. Definice konstanty v TB se provádí přiřazením konstanty do proměnné. Používání proměnné místo konstanty zvyšuje srozumitelnost a modifikovatelnost programu. Potřebujeme-li změnit hodnotu konstanty, která se v programu často vyskytuje, stačí tuto změnu provést jen jednou při její definici a nemusíme vyhledávat a měnit tuto konstantu v mnoha místech programu. V TB jsou již předdefinovány konstanty: %0, %1, %2, %3, které není třeba definovat.

Příklad:
```
10 MAX=5: PI=3.1415 : REM definice konstant
20 DIM VEKTOR(MAX),MATICE;MAX,MAX+1)
30 FOR I=%1 TO MAX: VEKTOR(I)=%0:NEXT I
```

2. 4 Návěští

Často se stává, že potřebujeme některý programový řádek označit (pojmenovat), abychom se na něj mohli jednoznačně odvolávat ve skokových příkazech. Deklarace návěští je přidělení : id: programovému řádku. Tento : id: se zapíše na začátek příslušného programového řádku.

syntax:
<návěští>:=<čř>≠<: id:>

Příklad:
125 ≠ZACATEK.PROGRAMU

170 ≠CTENI: INPUT "x=";X

Možností používat návěští odstraňuje TB jednu z největších potíží při psaní programu v Atari Basicu, kdy jsme si museli pamatovat : čř: programových řádků, na které byly prováděny skokové příkazy GOTO, GOSUB. V TB se význam : čř: redukuje pouze na informaci, udávající místo programového řádku v programu. Důsledné používání návěští současně doplňuje komentář a ulehčuje práci se skokovými příkazy.

2. 5 Klíčová slova

Klíčová slova jsou : id:, kterými jsou označeny příkazy jazyka TB. Jsou to tedy : id:, které jsou v syntaxu jazyka vyhrazeny jednotlivým příkazům. Proto není vhodné tyto : id: používat jako : id: proměnných (dochází k chybám).

ABS	DATA	FCOLOR	NEW	RAD	TEXT
ADR	DEC	FILLTO	NEXT	RAND	THEN
AND	DEG	FOR	NOTN	READ	TIME
ASC	DEL	FRAC	NOTE	READY	TIME$
ATN	DELETE	FRE		REM	TRACE
	DIM	GET	ON	RENAME	TRAP
BGET	DIR	GO	OPEN	RENUM	TRUNC
BLOAD	DIV	GOSUB	OR	REPEAT	
BPUT	DO	GOTO		RESTORE	UINSTR
BRUN	DOS	GRAPHICS	PADDLE	RETURN	UNLOCK
BYE	DPEEK		PAINT	RND	UNTIL
	DPOKE	HEX$	PAUSE	RUN	USR
CHR$	DRAWTO		PEEK		
CIRCLE	DSOUND	IF	PLOT	SAVE	VAL
CLOAD	DUMP	INKEY$	POINT	SETCOLOR	
CLOG		INPUT	POKE	SGN	WEND
CLOSE	ELSE	INSTR	POP	SIN	WHILE
CLR	END	INT	POSITION	SOUND	
CLS	ENDIF		PRINT	SQR	XIO
COLOR	ENDPROC	LEN	PROC	STATUS	
COM	ENTER	LET	PTRIG	STICK	%GET
CONT	ERL	LIST	PUT	STOP	%PUT
COS	ERR	LOAD		STRIG	%0 %1 %2
CSAVE	ERROR	LOCATE		STR$	%3
	EXEC	LOCK			*B *F *L
	EXIT	LOG			-- =
	EXOR	LOOP			&
	EXP	LPRINT	MOD	MOVE	! $? "

3. DEKLARACE PROMĚNNÝCH

Deklarace proměnných je pojmenování určitého místa paměti počítače. To znamená, že při použití : id: proměnné v programu počítač provede automaticky výpočet adresy paměti, odkud se hodnota čte nebo kam se zapisuje.

Pod pojmem proměnná tedy rozumíme údajový objekt, který může během realizace programu nabývat různé hodnoty. Nejde však o libovolné hodnoty; každá proměnná může nabývat hodnot určitého typu. Říkáme, že proměnné jsou určitého typu. V TB rozlišujeme tři typy proměnných.

syntax:
<proměnná>:=|<aritmetická proměnná>|<indexovaná proměnná>|
|<znaková proměnná>|

- aritmetická proměnná – obsahuje číselné hodnoty
- indexovaná proměnná – obsahuje číslené hodnoty
- znaková proměnná – obsahuje znakové řetězce

3. 1 Aritmetické proměnné (: ap:)

Aritmetické proměnné jsou takové proměnné, které během chodu programu mohou nabývat číselné hodnoty. Takovýmto proměnným také říkáme jednoduché ar. proměnné.

Identifikátor : ap: označuje jedno paměťové místo, které obsahuje 6 Bytů paměti. Na toto paměťové místo zapisujeme nebo z něj čteme číselné hodnoty. TB používá k deklaraci : ap: tzv. implicitní způsob deklarace proměnné. To znamená, že deklarace : ap: se provede při prvním výskytu jejího : id: (jména) v programu.

syntax:
<deklarace ar. proměnné>:=<impicitní deklarace>

Příklad:
```
10 SUMA =0 : REM prvni pouziti v programu
...
50 SUMA=SUMA+5
```

3. 2 Indexované proměnné (: ip:)

Často potřebujeme pracovat s vektory a maticemi, tj. uspořádanými skupinami proměnných. Takovýmto objektům obecně říkáme pole. To

znamená, že při deklaraci indexované proměnné neprovádíme pojmenování pouze jednoho místa v paměti, ale deklarovaný : id: proměnné je společný pro několik míst paměti. K tomu, abychom mohli pracovat s jednotlivými prvky pole, je k dispozici index, kterým provádíme upřesnění konkretního místa paměti.

Indexovaná proměnná (pole) je tvořena uspořádanou množinou : ap: a index určuje, se kterou proměnnou z této množiny chceme pracovat. TB umožňuje pracovat s jednorozměrnými poli (vektory) a dvojrozměrnými poli (maticemi). Deklaraci : ip: provádíme příkazy DIM nebo COM.

syntax:
<deklarace indexované prom.>:=|DIM|COM|<Seznam proměnných>
<seznam proměnných>:=<id>(<rozměr1>[,<rozměr2>]),...
<index>:=1 ... <rozměr>
<rozměr>:=|1|2|...|5460|

<rozměr> – je : av:, jehož hodnota určuje maximální velikost (tj. počet prvků), kterou může příslušný index nabývat. Určuje tedy dimenzi indexu.
<index> – je : av:, jehož hodnota určuje, s kterým prvkem pole budeme pracovat

Příklad:
```
10 MAX=10
20 DIM X(MAX*4),A(MAX,6)
   ...
90 A(MAX-2,1)=100: X(I*2)=A(5-I,I)
```
– kde X(MAX*4) – vektor o 40 prvcích
A(MAX,6) – matice o 10 řádcích a 6 sloupcích

Při dimenzování velkých polí musíme kontrolovat, zda nám k jejich deklaraci postačuje volná paměť. Velikost volné paměti zjistíme příkazem PRINT FRE(0). Musíme si při tom uvědomit, že velikost paměti, kterou zabere : ip:, vypočítáme podle vztahu : VELIKOST=6*ROZMER1*ROZMER2, protože jeden prvek : ip: zabere v paměti 6 Bytů.

Pokud překročíme deklarací proměnných velikost volné paměti, potom po provedení příkazů DIM nebo COM v programu dojde k havárii TB, neboť příkazy při deklaraci současně provádějí i nulování paměti, kterou vyčleňují pro : ip:. Tuto chybu TB neošetřuje! Pokud k ní dojde, musíme TB znovu nahrát.

Příklad:
```
10 DIM A(5000),B$(30000)
RUN
```

3. 3 Deklarace znakových proměnných (: $p:)

Práce s řetězci, tedy používání znakových (stringových) proměnných, patří mezi nejčastější úkony programátora. Většina informací, které se dnes na počítačích zpracovávají, mají charakter znakových řetězců. Deklarací znakové proměnné určujeme, kolik znaků (tj. jak dlouhý řetězec) můžeme do znakové proměnné uložit.

syntax:
<deklarace znakové proměnné>:=|DIM|COM|<id$>(<délka>),...
<délka>:=|1|2|...|32767|

<délka> – je : av:, jehož hodnota určuje počet znaků řetězce, které můžeme do proměnné uložit.

Příklad:
10 COM ZNAK$(1),RADEK(40)

Na rozdíl od deklarace pole, které se skládá z n – prvků, je znaková proměnná chápána jako jednoduchá proměnná, t.j. tvoří jeden celek. Délka : $p: současně udává, kolik Bytů paměti proměnná zabere (1 znak zabere 1 Byt).

Doporučení:
Vzhledem k tomu, že TB má k dispozici pro deklaraci dva příkazy vykonávající stejnou funkci, je vhodné příkaz DIM používat k deklaraci polí (: ip:) a příkaz COM k deklaraci znakových proměnných (: $p:).

Příklad:
```
10  DIM VEKTOR(20),MATICE(5,6)
20  COM ZNAK$(1),RADEK$(40)
```

Deklarace proměnných pomocí příkazů DIM a COM se nazývá explicitní deklarace a je vhodné ji provádět na začátku programu (po definici konstant).

3. 4 Rušení deklarací

Pokud v programu pracujeme s velkými poli nebo dlouhými znakovými proměnnými (nebo je velký program), může dojít k situaci, že se nám další deklarované proměnné již nevejdou do paměti. Potom, pokud to algoritmus dovoluje, můžeme provést příkazem CLR zrušení všech deklarací a příkazy DIM a COM provést deklaraci novou. K uchování výsledků předchozí části programu musíme použít vnější soubor na MGF nebo disku.

syntax:
<rušení deklarací>:=CLR

Příkaz CLR ruší všechny deklarace, což se u : ap: jeví jako jejich vynulování.

Příklad:

```
10  DIM A(20),B(3)
20  COM ZNAK$(1)
30  SUMA=99:DUMP "S: ":GET Q
    ...
90  CLR:DUMP "S: ":GET Q
95  DIM A(5,6):REM nova deklarace
```

4 VÝRAZY

Pod pojmem výraz rozumíme předpis, kterým stanovujeme, jak získáme určitou hodnotu. Výraz se skládá z proměnných, konstant, funkcí, operátorů a závorek. Podle toho, jaký charakter má hodnota, kterou po vyhodnocení získáme, rozdělujeme výrazy v TB do tří typů.

Aritmetické výrazy: jejich výsledkem je aritmetická hodnota, tj. číslo.
Př.: A*B–7.3

Relační výrazy: Jejich výsledkem je logická pravda (TRUE) nebo nepravda (FALSE). Př.: A:B OR B$="N"

Znakové výrazy: jejich výsledkem je znakový řetězec.

Př.: A$(5,7)

syntax:
<výraz>:=|<aritmetický výraz>|<relační výraz>|<znakový výraz>|

4. 1 Aritmetické výrazy (: av:)

Aritmetický výraz je pravidlo pro výpočet číselné hodnoty. Aritmetické výrazy se v TB tvoří podobně jako v matematice z aritmetických a indexovaných proměnných, konstant a funkcí, které jsou spojeny ar. operátory. Z hlediska konstrukce lze ar. výrazy rozdělit na prosté ar. výrazy, jednoduché a složené aritmetické výrazy.

syntax:
<ar. výraz>:=|<prostý ar. výraz>|<jednoduchý ar. výraz>|
|<složený ar. výraz>|
<ar. operátor>:=|^|*|/|DIV|MOD|+|–|
<prvek ar. výrazu>:=|<ap>|<ip>|<ac>|<af>|

Poznámka:

Operátor DIV je operátor celočíselného dělení. Výsledkem operace MOD je zbytek po celočíselném dělení.

Příklad:

```
A=7.3:C=A DIV 3:Z=A MOD 3: PRINT A,C,Z
  7.3        2        1.3
```

4. 1. 1 Prosté aritmetické výrazy

Prostým : av: rozumíme takový výraz, který se skládá ze znaménka a jediného prvku ar. výrazu.

syntax:

<prostý ar. výraz>:=[<zn>]<prvek ar. výrazu>

Příklad:

–SUMA , X(I*2) , –1.25 , INT(B)

Výsledkem prostého : av: je hodnota prvku, který je ve výrazu zapsán, ovlivněná znaménkem.

4. 1. 2 Jednoduchý aritmetický výraz

Jednoduchým : av: rozumíme takový výraz, který se skládá z několika prvků ar. výrazů spojených ar. operátory. Neobsahuje však žádné závorky.

syntax:

<jednoduchý av>:=[<zn>]<prvek av><ar.operátor><prvek av>
[<ar.operátor><prvek av>...]

Příklad:

A+B , A*2–C , –6/ABS(SUMA)+POCET^2 , C*B DIV X(5)

Výsledkem jednoduchého :av: je číselná hodnota, která se vypočte se zřetelem na prioritu operátorů z hodnot prvků ar. výrazů.

Priorita aritmetických operátorů:

1)	Umocňování	| ^ |
2)	Násobení a dělení	| * | / | DIV | MOD |
3)	Sčítání a odčítání	| + | – |

Postup vyhodnocení jednoduchého : av: probíhá ve třech krocích. V prvém kroku se provedou operace umocňování, v druhém kroku operace násobení a dělení a v třetím kroku sčítání a odčítání. Přitom v jednotlivých krocích se postupuje při vyhodnocování výrazu zleva doprava. Postup vyhodnocení jednoduchého : av: je vidět z následujícího příkladu.

Platí: A=1:B=2:C=3:D=4
Výraz: C^2+D*C/3+A-B^C+D DIV C
1. krok: 9+D*C/3+A-8+D DIV C
2. krok: 9+ 4 +A - 8 + 1
3. krok: 9+ 4 +1 - 8 + 1 = 7

Poznámka:

$\frac{D}{B}$*3 se zapíše D/B*3, ale $\frac{D}{B*3}$ se zapíše D/(B*3)

4 . 1. 3 Složený aritmetický výraz

Složeným aritmetickým výrazem rozumíme takový výraz, který se skládá z několika jednoduchých : av: uzavřených v závorkách a spojených ar. operátory. Uzavíráním jednoduchých : av: do závorek lze měnit postup vyhodnocování : av:, který je dán prioritou operátorů. V složeném : av: se provede nejprve vyhodnocení jednoduchých : av: v závorkách (odstranění závorek) a z takto získaných hodnot se provede vyhodnocení celého výrazu podle priority ar. operátorů, které spojují jednoduché : av: uzavřené v závorkách.

syntax:
<složený av>:=[<zn>](<jedn. av>)<ar. operátor>(<jedn. av>)
[<ar. operátor>(<jedn. av>)...]

Příklad:
(A^2-D/B*3)*3-A+(D+INT(C)+X(I)-1.2) DIV (A MOD C)

Poznámka:
Při konstrukci velmi složitých : av: lze jednoduchý výraz v závorkách nahradit dalším složeným výrazem, vzniká tzv. vnořování závorek.

Příklad:
B*(C+D/(B*3))-0.01*(INT(D+5)+2*(X(I+1)-6)-1)

Častou chybou při tvorbě složených : av: je chybné umístění nebo chybný počet závorek. Proto je třeba realizovat tvorbu složených : av: s větším počtem vnoření závorek opatrně. Je vhodnější složitý : av: rozdělit do několika jednodušších výrazů, což nám umožní snadněji ověřit správnost jeho funkce.

Příklad:
V 1 =C+D/(B*3):V2=X(I+1)-6:V3=INT(D+5)+2*V2-1)
V=B*V1-0.01*V3:PRINT V,V1,V2,V3

4. 2 Relační výrazy (: rv:)

Relační výrazy provádějí porovnání hodnot (aritmetických nebo znakových) a na základě použitých relačních operátorů provedou vyhodnocení výrazu. Výsledkem relačního výrazu je logická 0 (FALSE) jestliže relace výrazu neplatí nebo logická 1 (TRUE), když je relace platná. Z hlediska konstrukce lze : rv: rozdělit na prosté, jednoduché a složené rel. výrazy.

syntax:
<relační výraz>:=|<prostý rv>|<jednoduchý rv>|<složený rv>|
<relační operátor>:=|<|>|=|<=|<=|<>|NOT|AND|OR|

K správnému zápisu : rv: je třeba znát základní zásady Booleovské algebry, které jsou uvedeny v následující tabulce:

a	0	1	0	1	význam operace
b	0	0	1	1	
a OR b	0	1	1	1	disjunkce a,b (a nebo b)
a AND B	0	0	0	1	konjukce a, b (a i b)
NOT a	1	0	1	0	negace a

Význam správného pochopení konstrukce :rv: vyplývá z jejich použití v příkazech IF, WHILE, UNTIL.

4. 2. 1 Prostý relační výraz

Je zvláštním druhem rel. výrazu. TB totiž nerozlišuje logickou 0 a 1 od aritmetické 0 a 1. To je odlišnost TB od vyšších programovacích jazyků. To umožňuje vytvořit prostý rel. výraz, který neprovádí žádné porovnání. Pouze provede vyhodnocení : av: a výraz pokládá za pravdivý, jestliže výsledná hodnota : av: je větší nebo rovna 0.5 a za nepravdivý, je-li hodnota menší než 0.5. Změnu výsledku prostého : rv: můžeme změnit operátorem negace.

syntax:
<prostý rel. výraz>:=[NOT]<av>

Příklad:
PRINT NOT A*B
IF A+B-C THEN ...

4. 2. 2 Jednoduchý relační výraz

Jednoduchý : rv: je takový výraz, který se skládá z dvojic : av: nebo : $v:, které jsou vzájemně spojeny rel. operátory, ale neobsahují žádné závorky.

syntax:
<jednoduchý rv>:=|<av>|<$v>|<rel. operátor>|<av>|<rv>|
[<rel. operátor>|<av>|<$v|...]

Příklad:
C*B>X(5) OR T$="ANO" , A OR B AND T$>C$(6)

Vyhodnocení jednoduchého : rv: probíhá ve dvou fázích: – v první fázi se provede vyhodnocení :av: a : $v:.
– v druhé fázi se provádí porovnání výsledných hodnot : av: a : $v: podle priority relačních operátorů.

Priorita relačních operátorů:
1) |<|>|=|<=|>=|<>|
2) |NOT|
3) |AND|
4) |OR|

Druhá fáze vyhodnocení : rv: probíhá tedy ve čtyřech krocích. Přitom v jednotlivých krocích se postupuje zleva doprava. Postup vyhodnocení jednoduchého : rv: je vidět z následujícího příkladu.

Platí: A=1:B=2:C=3:D$="A"
Výraz: A+B>1.5 OR D$="A" AND B^2<C OR NOT A
1. fáze: 3 > 1.5 OR "A"="A" AND 4< 3 OR NOT 1
1. krok: TRUE OR TRUE AND FALSE OR NOT 1 (2. fáze)
2. krok: TRUE OR TRUE AND FALSE OR FALSE
3. krok: TRUE OR FALSE OR FALSE
4. krok: TRUE – výsledek, výraz je pravdivý

4. 2. 3 Složený relační výraz

Složeným : rv: rozumíme takový výraz, který se skládá z několika jednoduchých : rv: uzavřených v závorkách a spojených rel. operátory. Uzavřením jednoduchých : rv: do závorek lze měnit postup vyhodnocení : rv:, který je dán prioritou rel. operátorů.

syntax:
<složený rv>:=(<jedn. rv>)<rel. operátor>(<jedn. rv>)
[<rel. operátor>(<jedn. rv>)...]

V složeném : rv: se nejprve provede vyhodnocení výrazů uvnitř závorek (odstranění závorek) a potom vyhodnocení celého : rv: se provede podle priority rel. operátorů, které spojují závorky.

Příklad:
(A^2>9 OR C=0) AND (T$="A" AND ABS(D+5)<>12)

Tak jako u složených : av: lze i u : rv: provádět vnořování závorek.
Příklad:
(A=0 AND (B>C OR CHR$(Q)=T$))

4. 3 Znakové výrazy (: $v:)

Znakový výraz pracuje se znakovými proměnnými (nebo jejich částmi), znakovými konstantami a znakovými funkcemi. Výsledkem znakového výrazu je znakový řetězec. Všechny znakové výrazy v TB jsou konstruovány jako prosté znakové výrazy.

syntax:
<znakový výraz>:=:|<$p>[(<od>[,<do>])]|<$c>|<$f>|

kde – <od> je: :av:, jehož hodnota udává pozici v řetězci znakové proměnné, od které bude zahájeno vyhodnocování výrazu. Pokud není : od: uvedeno pracuje se od začátku řetězce.

kde – <do> je : av:, jehož hodnota udává poslední pozici v řetězci znakové proměnné, do které bude znak. výraz vyhodnocován. Není-li : do: uvedeno, pracuje se do konce řetězce.

Příklad:
"konstanta", STR$(A), TEXT$(5,9), ZNAK$

Vyhodnocování : $v: je vidět z následujícího příkladu:

```
10  COM PR$(17)
20  PR$="Petr, Ondra, Martin" :  REM $v = konstanta
30  PRINT PR$: REM od, do neni uvedeno, cely retezec
40  PRINT PR$(12) : REM od 12 pozice do konce retezce
50  PRINT PR$(6,10) : REM od 6 do 10 pozice
60  PRINT CHR$(65) : REM $v je znaková funkce
RUN
Petr, Ondra, Martin
Martin
Ondra
A
READY
```

4. 4 Funkce

Funkce jsou nedílnou součástí výrazů v TB. Funkce je speciální program, jehož výsledkem je aritmetická nebo znaková hodnota, která se do výrazu dosadí na místo identifikátoru funkce. To nám dává možnost používat funkce jako argumenty ve výrazech stejným způsobem jako proměnné nebo konstanty. Podle hodnoty, kterou funkce předává do výrazu, rozlišujeme funkce na aritmetické a znakové.

syntax:
<typ funkce>:=|<aritmetická funkce>|<znaková funkce>|
<zápis funkce>:=<id funkce>[(<seznam parametrů>)]
<seznam parametrů>:=<parametr1>[,<parametr2>...]

kde <parametry> jsou :av: nebo : $v: a na základě jejich hodnot se provede výpočet funkce.

Výsledek funkce: – ar. hodnota není-li na konci : id: znak $.
– znaková hodnota, když : id: funkce končí znakem $.

Příklad:
X=A*COS(ALFA):Y$=CHR$(Q):C=VAL(T$)

Podle účelu, k němuž slouží, lze funkce rozdělit do následujících skupin:
A) Adresové funkce
B) Trigonometrické funkce
C) Číselné funkce
D) Speciální funkce
E) Funkce ovládačů her
F) Řetězcové funkce
Poznámka:
Všechny funkce v TB jsou standartní, neboť TB neumožňuje definici vlastních funkcí.

A) Adresové funkce

ADR(<$p>) – výsledkem je adresa počátku : $p: v paměti

DPEEK(<adr>) – výsledkem je číselná hodnota 0 ... 32767, která se vypočítá z obsahu adres : adr: + 256*:adr+1:

FRE(0) – výsledkem je počet volných Bytů v uživatelské části paměti. Udává, kolik místa nám zbývá pro deklaraci proměnných a program.

PEEK(<adr> – výsledkem je číselná hodnota 0...255, uložená v jednom Bytu paměti na adrese : adr:.

USR(<adr>[,p1...p5]) – funkce provede spuštění podprogramu ve strojovém kódu, který je uložen v paměti od adresy : adr:. Současně umožňuje předat podprogramu parametry p1 ... p5.

Příklad:
ADR(T$), DPEEK(88), FRE(0), PEEK(82), USR(1536)

B) Trigonometrické funkce

ANT(<av>) – arcustangens
COS(<av>) – cosinus
SIN(<av>) – sinus

Při výpočtu hodnot trigonometrických funkcí můžeme pomocí předznamenání určit, zda hodnota : av: bude ve stupních (DEG) nebo v radiánech (RAD). Normálně platí předznamenání RAD.

syntax:
<předznamenání trig. funkcí>:=|DEG|RAD|

Příklad:
DEG: PRINT COS(60):RAD: PRINT COS(1)

C) Číselné funkce

ABS(<av>)	– absolutní hodnota : av:
CLOG(<av>)	– dekadický logaritmus : av:
DEC(<$v>)	– provádí převod hexadec. čísla zapsaného v : $v: na decimální číslo (HEX("A1F3"))
EXP(<av>)	– exponenciální funkce (2.718.: av:)
FRAC(<av>)	– desetinná část : av: (FRAC(1.75)=0.75)
INT(<av>)	– výsledkem je nejvyšší celé číslo, které je menší nebo rovno hodnotě : av: (INT(–0.3)=–1)
LOG(<av>)	– přirozený logaritmus : av:
RAND(<av>)	– náhodné celé číslo |0|1| ... |: av: –1|
RND	– náhodné číslo 0 ... 1
SGN(<av>)	– určení znaménka;: av: menší 0 výsledek funkce 0, jinak je výsledek 1.
SQR(<av>)	– druhá odmocnina : av:
TRUNC(<av>)	– celá část čísla hodnoty : av: (TRUNC(–0.3)=0)

D) Speciální funkce

ERL	– udává : čř:, na kterém vznikla chyba
ERR	– udává číslo chyby
INKEY$	– obsahuje znak právě stisklé klávesy. Není-li stlačena žádná klávesa, obsahuje " ".
TIME	– není funkcí, ale speciální proměnnou (tj. můžeme jí přiřazovat hodnotu), která obsahuje časový údaj v padesátinách vteřiny. Pokud jí není přiřazena žádná hodnota, obsahuje čas od zapnutí počítače.
TIME$	– speciální proměnná obsahující čas ve tvaru: hhmmss, kde hh – jsou hodiny, mm – minuty a ss – vteřiny. Umožňuje nám nastavit např. astronomický čas: TIME$="130500"

Příklad:

```
10  ≠A: IF INKEY$=" " THEN GO≠A : REM ceka stisk klavesy
20  GRAPHICS 2: TIME$="000000": REM stopky
30  ≠CAS: POSITION 1,1: PRINT≠6; TIME $: GO≠CAS
```

E) Funkce ovladačů

PADDLE(<av>) – udává polohu ovladače na kanálu : av: 0 ... 3.

PTRIG(<av>) – vyhodnocuje stlačení tlačítka ovladače na kanálu : av: 0 ... 3.

STICK(<av>) – udává polohu joysticku na kanálu : av: |0|1|.
Čísla poloh:

10	14	6
11	15	7
9	13	5

STRIG(<av>) – vyhodnocuje stlačení tlačítka joysticku na kanálu :av: |0|1|. Je-li tlačítko stlačeno, výsledkem funkce je 0, jinak je výsledek 1.

Příklad:

```
10  ≠TEST: PRINT STICK(0), STRIG(0): GO≠TEST
```

Tímto jednoduchým programem můžeme ověřit správnost funkce joysticku. Program ukončíme stlačením BREAK.

F) Řetězcové funkce

ASC(<$v>) – výsledkem je číselná hodnota (v kódu ATASCII) prvního znaku : rv:. PRINT ASC("ABC")

CHR$(<av>) – výsledkem je znak, jehož kód v tabulce znaků odpovídá hodnotě : av:. PRINT CHR$(65)

HEX$(<av>) – převod dekadického čísla do hexadec. řetězce.

INSTR(<$v1>, <$v2>[<od>]) – funkce hledá v řetězci : $v1: stejnou posloupnost znaků, kterou obsahuje řetězec : $v2:. Ar. výraz : od: udává pozici, od které se hledání v řetězci :$v1: začíná. Výsledkem je číslo pozice, kde v řetězci : $v1: začíná posloupnost znaků : $v2:.
př.: PRINT INSTR("Petr,Ondra,Martin","Ondra")

LEN(<$p>] – zjišťuje délku řetězce, který je uložen v : $p:.

STR$(<av>) – převod číselné hodnoty do řetězce číslic.
X=765:A$=STR$(X) – výsledek v A$ je "765"

UINSTR(<$v1>, <$v2>[,<od>]) – provádí stejnou funkci jako INSTR s tím rozdílem, že nerozlišuje malá a velká písmena, normální a inverzní znaky.

VAL(<$v>) – provádí převod řetězce číslic na číselnou hodnotu.
Př.: PRINT VAL("7342")

Poznámka:
Výrazy, které jsou uvedeny jako parametry funkcí, mohou obsahovat další funkce.

Příklad:
A=4: B=6: C= –2: PRINT SQR(A*A+B*B/ABS(4*A*C))

5. JEDNODUCHÉ PŘÍKAZY

Příkazem rozumíme předpis na provedení určité algoritmické funkce. Jestliže je algoritmus posloupností algoritmických akcí, potom je program tvořen posloupností příkazů. Podle funkce, kterou vykonávají, se příkazy dělí do následujících skupin.:

syntax:
<příkaz>:=|<jednoduchý příkaz>|<strukturovaný příkaz>|
|<skokový příkaz>|<příkaz vstupu/výstupu>|
|<příkaz grafiky>|<zvukový příkaz>|
|<systémový příkaz>|

Jednoduchými nazýváme ty příkazy, které v sobě neobsahují jiný příkaz.

syntax:
<jednoduchý příkaz>:=|<přiřazovací příkaz>|<komentář>|
|<příkaz zastavení chodu programu>|
|<adresový příkaz>|

Příklad:
```
10  A=A*B: REM prirazovaci prikaz
20  REM komentar
30  PAUSE 50: REM prikaz zastaveni chodu programu
40  POKE 82,0:REM adresovy prikaz
```

5. 1 Přiřazovací příkaz

Přiřazovací příkaz reprezentuje akci, kterou proměnné přiřazujeme určitou hodnotu na základě vyhodnocení výrazu na pravé straně příkazu. To znamená, že provádíme zápis výsledné hodnoty výrazu do paměti na místo, které je pojmenováno identifikátorem proměnné. Přitom symbol = není chápán matematicky, ale vyjadřuje příkaz přiřaď proměnné na levé straně výslednou hodnotu výrazu na pravé straně. Podle typu výrazu dělíme přiřazovací příkazy na aritmetické, relační a znakové.

syntax:
<přiřazovací příkaz>:=[LET]<proměnná>=<výraz>
<typ přiřazovacího příkazu>:=|<aritmetický přiřaz. příkaz>|
|<relační přiřaz. příkaz>|
|<znakový přiřaz. příkaz>|

Příklad:
A=B*C+A/2 : R = A>B OR T$="A" :Z$="text"

5. 1. 1 Aritmetický přiřazovací příkaz

Aritmetický přiřazovací příkaz je nejčastěji používaným příkazem, kterým aritmetické proměnné nebo prvku indexované proměnné je přiřazena výsledná hodnota aritmetického výrazu.

syntax:
<ar. přiřaz. příkaz>:=[LET]|<ap>|<prvek ip>|=<av>

Příklad:
PI=3.1415: INDEX=A*2: VEKTOR(I)=(SUMA+POCET)*ABS(X)
I=I+1: REM vezmi obsah promenne I, zvets o 1 a prirad I

5. 1. 2 Relační přiřazovací příkaz

Výsledkem relačního výrazu je logická 0 (FALSE) nebo logická 1 (TRUE). Na rozdíl od vyšších programovacích jazyků TB interpretuje výsledek : rv: jako aritmetické hodnoty 0,1. Proto se hodnota : rv: přiřazuje do : ap: nebo prvku : ip:.

syntax:
<relační přiřaz. příkaz>:=[LET]|<ap>|<prvek ip>|=<rv>

Příklad:
R=A>0 AND (T$="A" OR T$="a") OR X(I)*A+B>C

Relační výrazy s výhodou použijeme tam, kde potřebujeme vyhodnotit složitý relační výraz. Tento složitý : rv: rozdělíme na několik jednodušších, jejichž výsledky přiřadíme proměnným.

Příklad:
Výraz: (A>B OR C^2>D) AND (X(1)=0 AND P$="A")
rozdělíme: R1=A>B OR C^2>D : R2=X(1)=0 AND P$="A"
Výsledek: R=R1 AND R2
Kontrolní výpisy: PRINT R1,R2,R

Poznámka:
Používání relačního přiřazovacího příkazu umožňuje do programu zapsat následující matematický nesmysl:

Příklad:

```
10  ≠VSTUP: INPUT "A,B=";A,B
20  A=A=B
30  PRINT A:GO≠VSTUP
```

Příkaz na řádku č. 20 realizuje relační přiřazovací příkaz, kdy do proměnné A je přiřazena hodnota výsledku : A=B. První symbol = vyjadřuje přiřazení a druhý symbol = je relační operátor.

5. 1. 3 Znakový přiřazovací příkaz

Znakovým přiřazovacím příkazem přiřazujeme do znakové proměnné řetězec, který získáme vyhodnocením znakového výrazu. Přitom znakový přiřazovací příkaz umožňuje určit pozici v znakové proměnné, od které bude hodnota : $v: do proměnné uložena.

syntax:
<znakový přiřaz. příkaz>:=[LET]<$p>[(<od>)]=<$v>

kde <od> je : av: jehož hodnota určuje první pozici v : $p:, od které se výsledek : $v: do proměnné začne zapisovat. Není-li : od: uvedeno, bude se výsledek ukládat od začátku, tj. od 1. pozice.

Příklad:

```
10  COM T$(20)
20  T$="Ondra" : REM prirazeni od zacatku
30  T$(6)="....." : REM prirazeni od 6. pozice
40  FOR I=1 TO 5
50     T$(10+I)=T$(6-I,6-I)
60  NEXT I
70  PRINT T$
RUN
Ondra.....ardnO
```

Z příkladu je vidět, jak lze v TB provádět pomocí znakových přiřazovacích příkazů úpravy, spojování a rozdělování řetězců.

5. 2 Komentáře

Komentář je velmi důležitou součástí programů, přestože nemá žádný vliv na jeho chod. Příkazy komentáře jsou nevýkonné příkazy. Komentář slouží programátorovi k popisu funkce programu, vysvětlení některých příkazů, grafickému rozdělení programu na jednotlivé části.

syntax:
<komentář>:=|REM text|--|

Příkaz komentáře REM popisuje pomocí textu, umístěného za klíčovým slovem REM, popis jednotlivých funkcí programu. Příkaz komentáře -- od sebe graficky oddělí jednotlivé části programu. Při výpisu programu příkazem LIST se místo dvou znaků – zobrazí (vytiskne) 30 znaků (pokud není výpis potlačen předznamenáním ✻L–).

Příklad:

```
10  REM textovy komentar
15  MAX=5:STO=100:REM definice konstant
20  --
30  DIM VEKTOR(MAX):REM deklarace promennych
35  COM RADEK$(40),ZNAK$(1)
40  --
50  ≠HLAVNI.PROGRAM
LIST
```

5. 3 Příkazy zastavení chodu programu

Pokud potřebujeme provést zastavení nebo ukončení chodu programu, použijeme příkazy PAUSE, STOP nebo END.

syntax:
<příkaz zastavení chodu programu>:=|PAUSE <av>|STOP|END|

PAUSE příkaz provádí zastavení chodu programu na dobu, která je dána hodnotou : av:. Na jednu vteřinu zastavíme program příkazem PAUSE 50. Tento příkaz je často využíván při tvorbě melodií a zvukových signálů.

STOP příkaz zastaví chod programu, ale všechny soubory ponechává otevřeny a proměnné nemění svoji hodnotu. Příkaz nám dává možnost (např. při ladění programu) vyhodnotit výsledky a provést změny obsahu proměnných. Program pokračuje v chodu po zadání systémového příkazu CONT.

END příkaz ukončuje chod programu a současně provede uzavření všech souborů, ponechává však proměnným jejich hodnotu.

příklad:

```
10  SOUND 1,121,10,14:PAUSE 25
20  CAS=5:PAUSE CAS*10
30  DSOUND:PRINT "stop":STOP
40  PRINT "pokracovani"
50  END
```

5. 4 Adresové příkazy

TB umožňuje programátorovi pomocí adresových příkazů přímo měnit obsah paměti. Nejmenším elementem paměti počítače Atari, se kterým můžeme pracovat, je 1 Byt. Přitom adresa, kterou v adresových příkazech zadáváme, určuje, se kterým Bytem paměti chceme pracovat.

syntax:

<adresový příkaz>:=|POKE<adr>,<av>|DPOKE<adr>,<av>|
|[–]MOVE<adr1>,<adr2>,<počet>|
<Byt>:=<nejmenší adresovatelný element paměti>
<adr>:=<adresa jednoho bytu v paměti>
<počet>:=<počet Bytů paměti>

kde <Byt> je element, který může nabývat hodnot 0...255.
kde <adr> je : av: jehož hodnota určuje místo 1 bytu paměti, se kterým budeme pracovat (0...65535).

POKE příkaz provádí změnu obsahu jednoho Byte paměti na adrese : adr:. Do Bytu provede zápis hodnoty : av:.

DPOKE příkaz provádí přepis dvou Bytů paměti na adrese : adr: a : adr+1:. Do těchto Bytů provádí zápis hodnoty : av: podle vzorce: <av>–256*INT(<av>/256) → <adr>
INT(<av>/256) → <adr+1>

MOVE příkaz provádí přesun bloku Bytů z jednoho místa paměti (: adr1:) na druhé (: adr2:). Velikost bloku je dána hodnotou : počet:. Předznamenání – MOVE určuje, že přesun bloku bude prováděn od konce k začátku.
Pozn.: Příkaz MOVE nahrazuje následující příkazy:
MOVE: FOR I=0 TO N: POKE A1+I,PEEK(A2+I):NEXT I
–MOVE:FORI=NTO0STEP–1:POKEA1+I,PEEK(A2+I):NEXT I
kde: N=:počet:–1, A1=:adr1:, A2=:adr2:

Pro práci s adresovými příkazy je třeba vědět, jak je organizována paměť vašeho počítače. Počítače Atari 800 mají paměť rozdělenou do tří částí:

1) Systémová komunikační oblast obsahující systémové adresy, vyrovnávací paměti a volnou paměť (256 Bytů) pro podprogramy ve strojovém kódu.

2) Uživatelská oblast, kam se ukládá program, proměnné a video RAM.

3) Systémová programová oblast, kde je uložen TB a obslužné podprogramy.

Adresové příkazy jsou příkazy nejnižší úrovně, nejsou systémem kontrolovány a tedy jejich nesprávné použití může vést k havárii systému. Proto je vhodné používat adresové příkazy pouze tam, kde nelze jinými způsoby docílit požadovaný efekt. Adresové příkazy se nejčastěji používají ke změnám systémových adres a tím k modifikaci chodu systému.

Tato problematika je podrobně rozpracována např. v příručce P. Dočekala Adresy počítače Atari (I, II).

Příklad:

```
POKE 82,0:REM posun zacatku radky od sloupce 0
TABELATOR=201:POKE TABELATOR,POCET.SLOUPCU
PRINT FRE(0):DPOKE 14,DPEEK(14)-1000:PRINT FRE(0)
MOVE 128*256,224*256,1024:REM presun sady znaku
-MOVE ODKUD,KAM,POCET
```

5. 4. 1 Důležité adresy systému

A) Rozsahy paměti:
- 14,15 – konec uživatelské oblasti paměti
- 88,89 – začátek video RAM

B) Obrazový editor:
- 82 – levý okraj textu 0 ... 39
- 83 – pravý okraj textu 1 ... 39
- 84 – č. řádku, kde je kurzor
- 85,86 – č. sloupce, kde je kurzor
- 201 – počet sloupců tabelátoru 3 ... 39
- 752 – volba viditelnosti kurzoru
  - 0 – viditelný, 1 – neviditelný
- 756 – volba sady znakového souboru
  - 224 – normální, 204 – mezinár. abeceda
  - 226 – malá písmena a graf. znaky

C) Vyrovnávací paměti:
- 1021 ... 1151 – magnetofonu
- 960 ... 999 – tiskárny

D) Klávesnice:
- 702 – volba režimu:
  - 0 – malá písmena
  - 64 – velká písmena
  - 128 – grafické znaky
- 755 – volba zobrazení : 0 – inverzní, 2 – normální

E) Tlačítka: 732 – tlačítko HELP: 0 – nestlačeno,
17 – stlačeno
53279 – stlačení tlačítka :
0 – OPTION + SELECT + START
1 – OPTION + SELECT
2 – OPTION + START
3 – OPTION
4 – SELECT + START
5 – SELECT
6 – START
7 – žádné nestlačeno

6 STRUKTUROVANÉ PŘÍKAZY

Jestliže chceme vytvořit rozsáhlejší program, který bude správně fungovat, musíme se držet určitých pravidel. Tato pravidla byla definována prof. Dijstrou a Wirthem, kteří se zabývali problematikou dokazování správnosti funkce programů. Došli k závěru, že nelze dokázat funkční správnost obecného sekvenčního programu. Důkaz je však možný, pokud je program vytvořen množinou programových struktur. Na základě tohoto závěru byl vytvořen programový jazyk Pascal, který používá pět základních typů programových struktur. Tyto struktury realizuje příkazy BEGIN/END, IF, CASE, WHILE, REPEAT/UNTIL. Tyto příkazy byly v podstatě převzaty i do TB, který nám tak umožňuje tvořit programové struktury pomocí příkazů IF/ENDIF, ON/EXEC, WHILE/WEND, REPEAT/UNTIL, PROC/ENDPROC.

Program tedy lze rozdělit na posloupnost programových struktur, přičemž platí základní pravidlo, že každá programová struktura má pouze jeden vstup a jeden výstup! Pokud správně tvoříme program podle zásad strukturovaného programování, potom se v programu nevyskytují skokové příkazy, sekvence provádění příkazů má logickou posloupnost.

Strukturované příkazy v TB jsou takové příkazy, které jsou vytvořeny z posloupnosti několika příkazů vytvářející jeden logický celek a je při jejich realizaci změněno sekvenční, tj. postupné provádění příkazů zapsaných v prog. řádcích. Podle funkce se dělí strukturované příkazy na podmíněné příkazy a příkazy cyklu.

syntax:
<strukturovaný příkaz>:=|<podmíněný přík.>|<příkaz cyklu>|
|<procedura>|

6. 1 Složený příkaz

Složený příkaz je takový příkaz, kterým z posloupnosti příkazů vytváříme jednu syntaktickou jednotku. V jazyku Pascal je složený příkaz uzavřen klíčovými slovy BEGIN/END. V TB budeme pokládat za složený příkaz takovou posloupnost příkazů, která má jeden vstup, jeden výstup a je zapsána na jednom nebo více programových řádcích.

syntax:
<složený příkaz>:=<čř> <příkaz>[:<příkaz>...]
...
<čř> <příkaz>[:<příkaz>...]

Diagram 1: Složený příkaz

Vstup

< příkaz >

...

< příkaz >

Výstup

Příklad:

```
 50  ≠VSTUP:INPUT "a,b";A,B
 60  C=A*B:POCET=POCET+1
 70  FOR I=1 TO POCET
 80     SUMA=SUMA+X(I)
 90  NEXT I
100  ≠VYSTUP
```

Poznámka:
Z příkladu je vidět, že součástí složeného příkazu může být i strukturovaný příkaz, pokud je zachována základní podmínka: jeden vstup a jeden výstup!

6. 2 Podmíněné příkazy

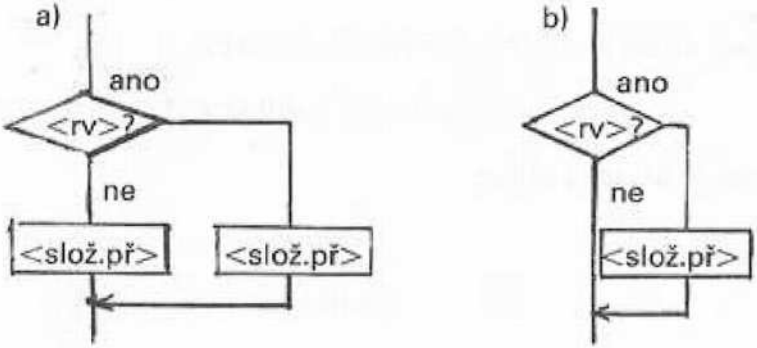
Podmíněné příkazy umožňují podmínit realizaci některých příkazů nebo provést volbu jedné z několika alternativ na základě výsledku relačního výrazu.

syntax:
<podmíněný příkaz>:=|<příkaz IF>|<příkaz ON>|

6. 2. 1 Příkaz IF

Příkaz IF slouží k rozvětvení programu do dvou větví na základě výsledku : rv:. Pokud je výsledek : rv: platný, provádí se příkazy v jedné větvi příkazu IF, je-li výsledek neplatný, provádí se příkazy v druhé větvi.

Diagram 2: Příkaz IF



K zápisu příkazu IF máme v TB k dispozici dvě varianty. První varianta je stejná jako v Atari Basicu (jednořádkový příkaz IF), druhá varianta tvoří strukturu složenou z několika programových řádků.

syntax:

```
<příkaz IF>:=<čř> IF <rv>
                <složený příkaz 1>
        <čř> ELSE
                <složený příkaz 2>
        <čř> ENDIF
:=<čř> IF <rv> THEN <příkaz>[:<příkaz>...]
```

Pokud je podmínka : rv: splněna provede se <složený př.1> (nebo příkazy za klíčovým slovem THEN). Není-li podmínka : rv: splněna provede se <složený př.2> pokud je uvedeno klíčové slovo ELSE. Jestliže není podmínka : rv: splněna a není uvedeno klíčové slovo ELSE, neprovede se v rámci příkazu IF žádný příkaz (diagram 2.b).

Příklad:

a) úplný příkaz IF

```
50 INPUT A,B
60 IF A=0 OR B=0
70    PRINT "vyraz plati"
75    SUMA=0
80 ELSE
85    PRINT "neplati"
90    X(A)=A/B:PRINT X(A)
95 ENDIF
```

b) neúplný příkaz IF

```
50 INPUT A,B
60 IF A*B::0
70    PRINT "plati"
80    SUMA=OSUMA+A/B
90 ENDIF
   ...
```

c) jednořádkový příkaz IF

```
60 IF A+B=0 THEN PRINT "plati":SUMA=SUMA+A+B
   ...
```

Z příkladů je vidět, že jednořádkový zápis příkazu IF použijeme tam, kde při splnění podmínky : rv: provádíme malý počet příkazů, které můžeme zapsat do řádku. V ostatních případech použijeme strukturovaného zápisu příkazu IF.

Poznámka:
V rámci strukturovaného příkazu můžeme jako součást složeného příkazu použít další strukturovaný příkaz. Je to podobné, jako když u výrazů používáme závorky pro vnořování. Závorky u strukturovaných příkazů jsou nahrazeny klíčovými slovy.

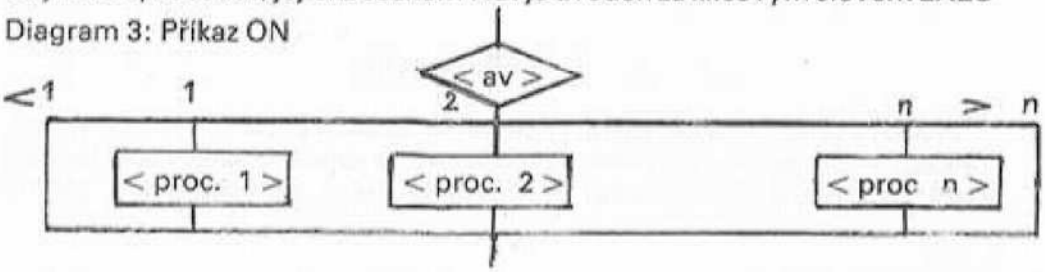
Příklad:

```
100 REPEAT
110    INPUT A,B
120    IF A MOD B=0
130       FOR I=1 TO MAX
140          X(I)=A
150          SUMA=SUMA+A
160       NEXT I
170       PRINT A,SUMA
180    ELSE
190       PRINT B,A MOD B,X(B)
200       SUMA=0:X(B)=0
210    ENDIF
220    C=C+A*B
230 UNTIL C>1000
   ...
```

6. 2. 2 Příkaz ON

Příkaz ON slouží k zápisu vícenásobného větvení programu. Umožňuje provést výběr jedné z více alternativ, která bude vybrána na základě výsledku ar. výrazu v příkaze ON. Na základě výsledné hodnoty : av: se provede jedna z procedur, jejichž seznam : id: je uveden za klíčovým slovem EXEC

Diagram 3: Příkaz ON



syntax:
<příkaz ON> := ON <av> EXEC <seznam id. procedur>
:= ON <av> GOSUB <čř>,<čř>...
<seznam id. procedur> := <id1>,<id2>[,<id3>...]

Příkaz ON/GOSUB zabezpečuje, aby programy zapsané v Atari Basicu, bylo možno spouštět v TB. Pro tvorbu programu v TB nemá význam.

Příkaz ON/EXEC provede vyhodnocení hodnoty : av:. Tato hodnota (zaokrouhlená na celé číslo) určuje pořadové číslo : id: procedury v seznamu za klíčovým slovem EXEC, která bude vykonána.

Příklad:

```
10 ≠VSTUP:INPUT "x,y=";X,Y
20 ON X+Y EXEC PRVNI,SUMA,NULA,SUMA
30 GO≠VSTUP
40 PROC PRVNI
42   PRINT "prvni"
45 ENDPROC
50 PROC SUMA
52   S=S+A+B: PRINT "suma";S
55 ENDPROC
60  PROC NULA
62   PRINT "nula":S=0
65 ENDPROC
```

V případě, že bude:
X+Y=1 provede se procedura PRVNI;
X+Y=2 nebo 4 provede se procedura SUMA;
X+Y=3 provede se procedura NULA.
X+Y:1 nebo X+Y:4 neprovede se nic.

Poznámka:
Jestliže výsledná hodnota : av: není celé číslo, provede se pro vytvoření pořadového čísla zaokrouhlení. Např.:
X+Y=0.49 – vyhodnotí jako 0, neprovede se nic
X+Y=0.50 – vyhodnotí jako 1, provede proc. PRVNI
X+Y=1.49 – vyhodnotí jako 1, provede proc. PRVNI
X+Y=1.50 – vyhodnotí jako 2, provede proc. SUMA

6. 3 Příkazy cyklu

Příkazy cyklu (repetiční příkazy) umožňují opakování určité posloupnosti příkazů. Počet opakování může být definován explicitně, tzn. je dán hodnotami parametrů cyklu (příkaz FOR/NEXT) nebo implicitně, kdy cyklus

probíhá tak dlouho, dokud není splněna podmínka jeho ukončení (příkazy WHILE/WEND, REPEAT/UNTIL, DO/LOOP).

syntax:
<příkaz cyklu>:=|<příkaz WHILE>|<příkaz REPEAT>|
|<příkaz DO>|<příkaz FOR>|

6. 3. 1 Příkaz WHILE

Příkaz WHILE je příkazem cyklu, kdy je prováděno opakování posloupnosti příkazů (složený příkaz) ohraničených klíčovými slovy WHILE a WEND tak dlouho, pokud platí podmínka : rv:.

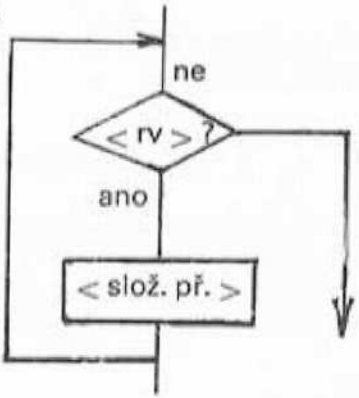
syntax:
<příkaz WHILE>:=<čř>WHILE<rv>
<složený příkaz>
<čř> WEND

Příklad:

```
10 X=0
20 WHILE X>=0
30    INPUT "x=";X
40    PRINT X,X*X
50 WEND
```

Příkazy 20 až 50 se budou provádět tak dlouho, dokud bude platit podmínka X>=0. Cyklus se ukončí po zadání X<0.

Diagram 4: Příkaz WHILE



Poznámka:
Pokud při vstupu do cyklu podmínka : rv: neplatí, potom se cyklus vůbec neprovede. V příkladě by k této situaci došlo při změně řádku 10 X= −1

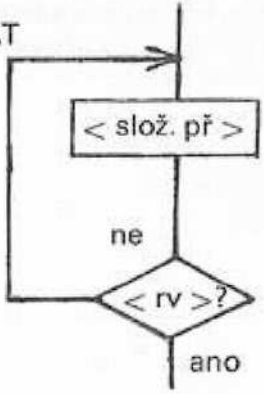
6. 3. 2 Příkaz REPEAT

Příkaz REPEAT je příkazem cyklu, kdy je prováděno opakování posloupnosti příkazů ohraničených klíčovými slovy REPEAT a UNTIL, dokud nedojde k splnění podmínky : rv: v klíčovém slově UNTIL.

syntax:

<příkaz REPEAT>:=<čř> REPEAT
<složený příkaz>
<čř> UNTIL <rv>

Diagram 5: Příkaz REPEAT



Z diagramu je vidět, že složený příkaz uvedený za klíčovým slovem REPEAT, bude proveden nejméně jednou. Opakování cyklu, na rozdíl od příkazu WHILE, probíhá tak dlouho, dokud nedojde k splnění podmínky : rv:.

Příklad:

```
10 REPEAT
20    INPUT A:SUMA=SUMA+A:PRINT SUMA,A
30 UNTIL A=0 OR SUMA>100
```

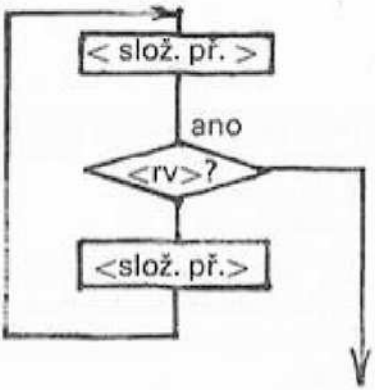
6. 3. 3 Příkaz DO

Příkaz DO je příkazem nekonečného cyklu, kdy je neustále prováděno opakování posloupnosti příkazů ohraničených klíčovými slovy DO a LOOP. Příkaz DO nemá tedy žádnou podmínku k svému ukončení. TB umožňuje ukončení každého příkazu cyklu pomocí příkazu EXIT.

syntax:

<příkaz DO>:=<čř> DO
<složený příkaz>
<čř> LOOP
< příkaz ukončení cyklu>:= EXIT

Diagram 6: Příkaz DO (s příkazem EXIT)



Příklad:

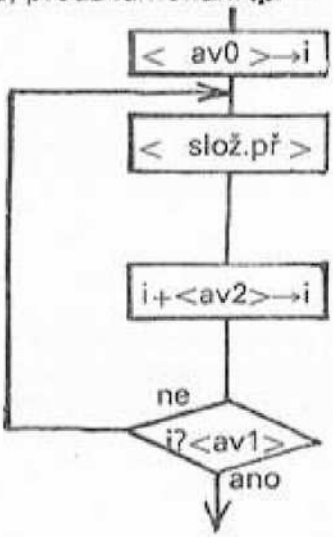
```
10 S=0:Q=0
20 DO
30    INPUT "a=";A
40    IF A=0 THEN EXIT
50    S=S+A:Q=Q+1
60 LOOP
70 PRINT S,Q
```

6. 3. 4 Příkaz FOR

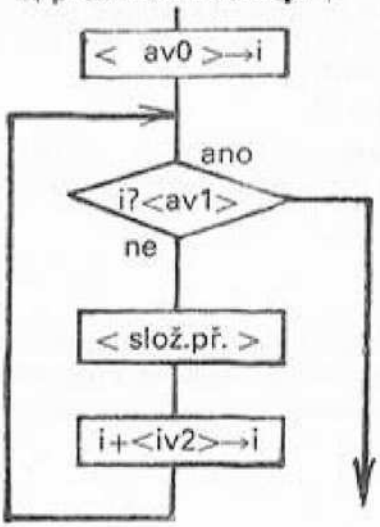
Příkaz cyklu FOR slouží k provedení předepsaného počtu opakování posloupnosti příkazů ohraničených klíčovými slovy FOR a NEXT. Počet opakování je dán hodnotami ar. výrazů uvedených v příkazu FOR. TB umožňuje používat dva druhy příkazu FOR a volba je realizovaná pomocí předznamenání ⁎F.

Diagram 7: Příkaz FOR

a) předznamenání ⁎F–



b) předznamenání ⁎F+



syntax:

<příkaz FOR>:=<čř> FOR <ap>=<av0>TO<av1>[STEP<av2>]
<složený příkaz>
<čř> NEXT <ap>

<předznamenání cyklu>:=|*F–|*F+|

kde <ap> – aritmetická proměnná, která je řídící proměnnou cyklu. V průběhu cyklu automaticky mění svůj obsah. Jestliže její hodnota dosáhne hodnoty : av1: tj. hodnoty konce cyklu, je cyklus ukončen.

<av0> –je ar. výraz, jehož výsledná hodnota určuje počáteční hodnotu řídící proměnné cyklu.

<av1> –je ar. výraz, jehož hodnota určuje poslední hodnotu, kterou může řídící proměnná cyklu dosáhnout.

<av2> – je ar. výraz, o jehož hodnotu příkaz konce cyklu NEXT zvětší hodnotu řídící proměnné cyklu. Pokud není tento : av2: uveden, zvětší se řídící proměnná cyklu o 1.
Hodnota : av2: určuje, zda je cyklus vzestupný (tj. hodnota : av2:>0) nebo sestupný (tj. : av2:<0).
–Podmínka konce vzestupného cyklu: : ap:<>:av1:
–Podmínka konce sestupného cyklu : :ap:::av1:

Předznamenání *F– (normální) určuje, že test konce cyklu se provede na jeho konci, tzn. příkazy uvnitř FOR/NEXT se provedou nejméně jednou. Předznamenání *F+ určuje, že test konce cyklu se provede na jeho začátku.

Příklad:

```
10 *F–
20 INPUT "pocatek, konec, krok ="; POCATEK,KONEC,KROK
30 FOR I=POCATEK TO KONEC STEP KROK
40    PRINT I
50 NEXT I
```

Změníme-li: 10 *F+, zadáme vstupní údaje např.: 10,5,1 potom se příkaz cyklu neprovede (podmínka konce splněna).

6. 4 Podprogramy

Jedním z klíčových problémů tvorby programů je jejich složitost a s ní spojená zvládnutelnost. Programovací jazyky umožňují pojmenovat určitou posloupnost příkazů a potom toto jméno používat v příkazech volání pod-

programů, kterými se aktivuje provedení pojmenované posloupnosti příkazů na různých místech programu. Takto pojmenovanou posloupnost příkazů nazýváme podprogramem (procedurou).

6. 4. 1 Procedury

Atari Basic neumožňuje tvorbu pojmenovaných podprogramů. (Podprogram začíná :čř: a končí RETURN). TB používá k deklaraci procedury příkazy PROC a ENDPROC. Realizace procedury v programu (volání podprogramu) je prováděno příkazem EXEC.

syntax:

```
<podprogram>:=<čř> PROC <id procedury>
                        <složený příkaz>
                    <čř> ENDPROC
                <=<čř><složený příkaz>
                    <čř> RETURN
<příkaz volání podprogramu>=|EXEC<id procedury>|
                            |GOSUB<čř>|
```

Příkazy GOSUB a RETURN v TB tvoří pozůstatek Atari Basicu, abychom v TB mohli spouštět programy vytvořené v Atari Basicu.

Příklad:

```
10  INPUT POLOMER
20  EXEC PLOCHA
30  PRINT SUMA,VYSLEDEK
    ...
99  END
100 PROC PLOCHA
110     VYSLEDEK=3.14*POLOMER^2
120     SUMA=SUMA+VYSLEDEK
130 ENDPROC
```

Používání procedur zvyšuje srozumitelnost programů a lépe se programy ověřují. Proto je vhodné procedury používat v co nejširším měřítku.

Poznámka:
Procedury v TB jsou oproti procedurám ve vyšších progr. jazycích omezeny v tom, že není možné proceduře předávat parametry a všechny proměnné v TB jsou globální, tj. proměnné deklarované v proceduře jsou přístupné v celém programu. Proto při používání procedur v TB je třeba věnovat větší pozornost proměnným, které se v procedurách používají.

6. 4. 2 Podprogramy ve strojovém kódu

Tvorba podprogramů ve strojovém kódu vyžaduje značné znalosti počítače a jeho programování. Program se skládá z množiny strojových instrukcí zapsaných ve strojovém kódu, která je uložena od konkrétní adresy. Nejčastěji se podprogram do paměti zapisuje pomocí příkazu POKE z vnitřního souboru programu DATA. Volání podprogramu ve strojovém kódu realizuje funkce USR.

Příklad:

```
10 REM podprogram pro presun stranek pameti
20 RESTORE≠PODPROGRAM
30 FOR I=0 TO 23: READ A: POKE 1536+I,A: NEXT I
   ...
80 POKE 1536+5, ORIGINAL: POKE 1536+8,KOPIE
90 X=USR(1536): REM spusteni podprogramu
100 ≠PODPROGRAM
110 DATA 104,160,255,173,0,0,141,0,0,136,192,0
120 DATA 240,9,238,3,6,238,6,6,76,2,6,64
```

7 SKOKOVÉ PŘÍKAZY

Příkazy programu v TB se provádějí v pořadí, jak jsou zapsány, pokud není direktivně dáno jiné pořadí. Ke změně pořadí se v TB používají skokové příkazy, které umožňují přímo odevzdat řízení výpočtu programu z jednoho místa na druhé. Z logického hlediska se dělí skokové příkazy na podmíněné a nepodmíněné.

syntax:
<příkaz skoku>:=|<nepodmíněný skok>|<podmíněný skok>|

Při tvorbě programu si musíme uvědomit, že skokové příkazy porušují přirozenou posloupnost provádění příkazů, ztěžují ověřování, snižují čitelnost programu. Z těchto důvodů se časté používání skokových příkazů nedoporučuje. Pokud program správně logicky sestavíme do programových struktur, potom není třeba skokové příkazy používat.

Poznámka:
Příkazem skoku nelze předat řízení programu na příkaz, který leží uvnitř strukturovaného příkazu.

7. 1 Nepodmíněné skoky

Nepodmíněný příkaz skoku provádí předání řízení programu z jednoho místa na druhé.

syntax:
<nepodmíněný skok>:=|GO≠<návěští>|GOTO<čř>|

Příkazem GO≠<návěští> se provede skok v programu z jednoho místa (na němž se nachází skokový příkaz) na jiné místo, tj. na programový řádek, který je označen návěštím, uvedeným v příloze GO. Program bude pokračovat prvním příkazem, který je uveden za návěštím.

Příkaz GOTO : čř: je opět pozůstatek Atari Basic a provádí stejnou funkci jako příkaz GO, tj. předá řízení na první příkaz, který je zapsán na prog. řádku : čř:.

Příklad:

```
10  ≠ZACATEK
20  INPUT " cislo ";CISLO
    ...
90  PRINT CISLO: GO≠ZACATEK
```

7. 2 Podmíněné skoky

Podmíněné příkazy skoku jsou takové příkazy, ve kterých je předání řízení z jednoho místa programu na druhé ovlivněno výsledkem rel. výrazu. V TB můžeme používat tři druhů podmíněných příkazů skoku (ON, IF, TRAP):

syntax:
<podmíněný skok>:=|ON<av>GO≠<návěští1>[,<návěští2>...]|
|ON<av>GOTO<čř>[,<čř>...]|
|IF<rv>THEN<čř>|
|TRAP≠<návěští>|TRAP<čř>|

a) Příkaz ON provádí obdobnou funkci, jako byla popsána v kapitole 6.2.2. Zde však seznam neobsahuje : id: procedur, ale seznam je tvořen návěštími, na které na základě výsledku : av: bude proveden skok. Příkaz skoku ON tvoří tzv. programový přepínač.

Příklad:

```
10  ≠VSTUP
20  INPUT "cislo ";CISLO
30  ON CISLO GO≠JEDNA,DVA,TRI
40  PRINT CISLO: GO≠VSTUP
100 ≠JEDNA:PRINT "jedna":GO≠VSTUP
110 ≠DVA:PRINT "dva":GO≠VSTUP
120 ≠TRI:PRINT "tri":GO≠VSTUP
```

Příkaz ON<av>GOTO provádí stejnou funkci s tím rozdílem, že řízení předává na řádky, jejichž : čř: tvoří seznam.

b) Příkaz IF<rv>THEN<čř>je pozůstatek z Atari Basicu a je možno jej nahradit jednořádkovým příkazem IF ve tvaru:
IF <rv> THEN GO≠<návěští>.

c) Příkaz TRAP≠<návěští> je zvláštním případem podmíněného skoku. Provádí nastavení návěští, na které bude proveden skok při vzniku chyby v některém z příkazů následujících za příkazem TRAP.

Příklad:

```
10  TRAP≠CHYBA
20  ≠VSTUP: INPUT "cislo ";CISLO
30  PRINT CISLO: GO≠VSTUP
80  ≠CHYBA: PRINT "chybny vstup !"
85  PRINT "radek:";ERL,"chyba:  ";ERR
90  GO≠VSTUP
```

Jestliže místo čísla vložíme na vstupu znakový řetězec, dojde k chybě a program bude pokračovat příkazy, které následují za návěštím uvedeným v příkaze TRAP.

Poznámka:
Zrušení nastavení návěští, kam se předá řízení při vzniku chyby, je realizováno příkazem TRAP 40000. Jestliže v některém z příkazů následujícím za tímto příkazem dojde k chybě, program se zastaví a systém vypíše zprávu ERROR.

7. 3 Skoky z podprogramu

TB umožňuje provést příkaz skoku z podprogramu (procedury) zpět do hlavního programu na určené návěští. To znamená, že podprogram

nebude ukončen příkazy ENDPROC nebo RETURN. Pokud chceme takový skok provést, musí příkazu GO předcházet příkaz POP, který ruší návratovou adresu z podprogramu.

syntax:
<rušení návratové adresy podprogramu>:=POP

Z logického hlediska je používání příkazu POP nesprávné a odporuje zásadám tvorby programů. Ve většině prog. jazycích je takovýto postup zakázán. Lze jej použít pouze při ošetření chybových stavů nebo při nutnosti zrušení normálního konce příkazů cyklu.

8 PŘÍKAZY VSTUPU / VÝSTUPU

Řešit problém na počítači znamená realizovat převod vstupních informací na výstupní podle algoritmu, který je převeden do zápisu programu. K realizaci této úlohy je nezbytná komunikace mezi počítačem a okolím prostřednictvím vstupně/výstupních zařízení. Hlavním úkolem programu je získávat (číst) informace ze vstupních zařízení, tyto informace zpracovat a výsledky podávat (zapisovat) na výstupní zařízení.

Počítače Atari 800/130 mají přehledně a srozumitelně definován systém označení jednotlivých V/V zařízení, které lze k nim připojit a jejich prostřednictvím realizovat komunikaci s okolím.

syntax:
<V/V zařízení>:=|C|D|D2|E|K|P|R|S|T|

kde C –magnetofon (dataset) 600Bd – vstup i výstup
D –disková jednotka (číslo 1) – vstup i výstup
D2 –disková jednotka (číslo 2) – vstup i výstup
E –obrazový editor (klávesnice + obrazovka) –vstup/výstup
K –klávesnice – vstup
P –tiskárna – výstup
R –manipulační program RS232
S –TV obrazovka (monitor) – výstup
T –magnetofon 2000Bd – vstup/výstup (TBASIC"T")

8. 1 Soubory

Informace na jednotlivých V/V zařízeních jsou z logického hlediska organizovány do souborů. Soubor je tvořen množinou informací, která bude ze zařízení čtena nebo kam bude zapisována.

Program tedy neprovádí nic jiného, než transformaci informací ze vstupních souborů do výstupních. Práce se soubory je základní úlohou programování!

Z hlediska organizace informací v souboru rozdělujeme soubory v TB do dvou základních typů:

– sekvenční: jsou to soubory, kde čteme/zapisujeme informaci jednu po druhé. Tzn., chceme-li např. číst informaci, která je v pořadí 15., musíme přečíst i 14 předcházejících. Soubory tohoto typu jsou na magnetofonu, tiskárně.

– soubory s přímým přístupem: můžeme přímo číst/zapisovat informace z kteréhokoliv místa souboru. Např. soubory na disku, obrazovce.

Práce se soubory v TB je realizována posloupností příkazů OPEN, příkazy V/V provádějící čtení/zápis informací a příkazů CLOSE.

8. 1. 1 Příkaz OPEN

V programech většinou pracujeme s několika soubory. Proto musíme v příkazech V/V jednoznačně určit, se kterým souborem mají pracovat. Soubor tedy identifikujeme v rámci programu. Tuto identifikaci souboru provádí příkaz OPEN, který zpřístupňuje informace na souboru pro V/V příkazy (otevírá soubor).

syntax:
<příkaz OPEN>:=OPEN ≠<ids>,<kód>,<char>"<vvz>:[<jmeno>]"
<ids>:=|1|2|...|7|
<kód>:=|4|6|8|9|12|...

kde<ids> – identifikátor souboru je dán hodnotou : av:, která určuje identifikační číslo souboru.

<kód> – je : av: jehož hodnota specifikuje způsob práce se souborem: 4=čtení
8 = zápis
12 = čtení i zápis
9 = zápis na konec souboru
6 = práce s diskovým adresářem

<char> – je : av: jehož hodnota tvoří dodatečnou charakteristiku souboru, která specifikuje způsob organizace souboru (normálně = 0).

<jmeno> – určuje, který z množiny souborů na disku nebo magnetofonu má být otevřen.

Příklady:
OPEN≠5,8,0,"S:"
MGF=1:OUT=8:OPEN≠MGF,OUT,0,"T:POKUS"

Příkaz OPEN provádí zápis základních informací o souboru (typ zařízení, velikost bufferu atd.) do tabulek IOCB (Input Output Control Block). Atari 800/130 má k dispozici 8 tabulek IOCB (č.:0...7).IOCB 0 je určeno pro rezidentní soubor.

8. 1. 2 Rezidentní soubor

K tomu, abychom mohli s počítačem po jeho zapnutí zahájit dialog, slouží rezidentní soubor. Tento soubor je automaticky otevírán systémem počítače. Na konci nahrávání TB do počítače je provedeno otevření rezidentního souboru příkazem OPEN = 0,12,0,"E:". To nám umožňuje po skončení nahrávání TB zahájit práci. Rezidentní soubor tedy zabezpečuje základní komunikaci s počítačem. Je to soubor typu textový editor, který provádí vstup informací z klávesnice a současně je zobrazuje na obrazovku.

Pokud v programu nebo v příkazovém řádku zadáváme V/V příkazy pracující s tímto souborem, nemusíme uvádět : ids:.

Příklad:
PRINT≠0;"text na TV" je stejné PRINT "text na TV"
INPUT≠0;CISLO je stejné INPUT CISLO

Jestliže provedeme příkaz CLOSE≠0, uzavře se rezidentní soubor, nemáme možnost komunikovat s počítačem a musíme stlačit tlačítko RESET, které obnovuje systém a otevře opět rezidentní soubor.

8. 1. 3 Příkaz CLOSE

Příkaz CLOSE provádí opačnou operaci než příkaz OPEN, tj. provádí uzavírání souboru a výmaz informací z IOCB : ids:. Po provedení příkazu CLOSE nejsou informace na souboru přístupné pro příkazy V/V.

syntax:

<příkaz CLOSE>:=CLOSE [≠<ids>]

kde <ids> je ar. výraz, jehož hodnota určuje, který soubor má být uzavřen (tj. udává číslo tabulky IOCB, která je příkazem vymazána).

Pokud není :ids: uveden, příkaz CLOSE provede uzavření všech souborů s výjimkou rezidentního souboru. Provádí výmaz tabulek IOCB č.: 1...7.

Příklad:

```
10  OPEN≠1,8,0,"S:"
    ...
50  PRINT≠1;"vypis do souboru"
60  PAUSE 50
    ...
90  CLOSE≠1
```

8. 1. 4 Příkazy STATUS a XIO

Příkaz STATUS slouží k zjištění stavu souboru, který se nachází na vnějším zařízení. Testuje, zda je zařízení připraveno k práci. Příkaz STATUS zapisuje do : ap: číslo chyby zařízení : ids: (není-li chyba : ap: = 0.)

syntax:
<příkaz STATUS>:=STATUS≠<ids>,<ap>

Příklad:

```
10  TRAP≠CHYBA:OPEN≠1,8,0,"D:POKUS"
    ...
90  ≠CHYBA:STATUS≠1,C.CHYBY:PRINT C.CHYBY
```

Příkaz XIO je univerzálním příkazem V/V operace. Jeho univerzálnost je na úkor srozumitelnosti. Proto je vhodnější používat ostatní příkazy V/V a příkaz XIO použít pouze tam, kde není jiná možnost.

syntax:
<příkaz XIO>:=XIOxx.≠<ids>,<kód>,<char>,"<vvz>:[<jmeno>]"

kde xx je číslo kódu operace, která bude příkazem XIO vykonána se souborem : ids:.

Příklad:
XIO03,≠1,8,0,"C:" je stejné OPEN≠1,8,0,"C:"
XIO254,≠1,0,0,"D:" provádí formátování disku
...

8. 2 Příkazy V/V řízené seznamem

Jestliže příkaz OPEN provedl otevření souboru, můžeme pomocí příkazů V/V ze souboru informace číst nebo do něj informace zapisovat. TB umožňuje dva základní způsoby přenosu informací. Prvním způsobem je přenos informací, který je řízen (definován) seznamem uvedeným v příkazu V/V.

syntax:
< příkaz V/V řízený seznamem> :=|<příkaz PRINT>|
|<příkaz INPUT>|

8. 2. 1 Příkaz PRINT

Příkaz PRINT realizuje výstup dat do souboru na výstupním zařízení. Provádí zápis informací, které jsou dány hodnotami výrazů v seznamu do výstupního souboru : ids:.

syntax:
<příkaz PRINT>:=|PRINT|?| [≠<ids>|,|;|] [<seznam výrazů>]
<seznam výrazů>:=<výraz>[|,|;|<výraz>...]
<oddělovač>:=| , | ; |

Ze zápisu syntaxe je vidět, že místo klíčového slova PRINT je možno použít symbol ?, který plní stejnou funkci.

Příklad:
PRINT "text" je stejné jako ? "text"
PRINT≠6;"vypis" je stejné jako ? ≠6;"vypis"

Oddělovače | , | ; | použité v seznamu výrazů určují, kam bude hodnota výrazu ve výstupním souboru umístěna.

– středník určuje, že hodnota následujícího výrazu bude umístěna ve výstupním souboru bezprostředně k předchozí (přilepena).
Příklad:
PRINT "text";"TEXT"
I=3: PRINT "X(";I;")=";123.45

– čárka určuje, že před výpisem hodnoty výrazu do výstupního souboru se vypíše určitý počet mezer, který je dán obsahem systémové adresy (: adr: = 201) tabelátoru. To znamená, že hodnoty výrazů jsou ve výstupním souboru zarovnány do sloupců, přičemž šířka sloupců je dána hodnotou tabelátoru (: adr: 201). Změna šířky sloupců tabelátoru (normálně 10) se provádí příkazem: POKE 201,|3|4|...
Příklad:
PRINT "text","TEXT"
POKE 201,5:PRINT "text","TEXT"
A=2: C=3: D=4: PRINT A,B,C,A*B*C, "text"

Nejčastěji používáme příkazu PRINT při výpisu informací na obrazovku. Soubor otevřený na zařízení "S:" je soubor s přímým přístupem, tj. pomocí příkazu POSITION můžeme určit souřadnice na obrazovce, od kterých bude příkaz PRINT informace do souboru (na obrazovku) zapisovat.

Příklad:

```
10  COM RETEZ$(20)
20  RETEZ$="AbcdeFghijKlmnoPqrst"
50  OPEN ≠ 3,8,0,"S:"
60  POSITION 2,5:PRINT ≠3;RETEZ$(6,10)
70  A=2: B=3: C=4: POSITION 2,10:?≠3; A*B,A;B;C
80  PRINT ≠3; RETEZ$,A+B+C;"km"
90  GET Q
99  CLOSE=3
```

8. 2. 2 Příkaz INPUT

Příkaz INPUT realizuje vstup informací ze souboru na vstupním zařízení do proměnných (aritmetických nebo znakových), které jsou uvedeny v seznamu proměnných. Typ vstupních informací, které vstupují ze souboru, musí být stejný jako typ proměnné, do které je informace ukládána.

Soubor informací na vstupním zařízení je tvořen posloupností aritmetických a znakových konstant, které jsou v souboru odděleny čárkami.

syntax:

<příkaz INPUT>:=INPUT [≠<ids>|,|;|] [<$c>|,|;|]<seznam>
<seznam>:=:|<ap>|<$p>|[,|<ap>|<$p>|...]

Příkaz INPUT v TB umožňuje při vstupu informací ze souboru typu "E:" (textový editor) vypsat řetězcovou konstantu (komentář), kde můžeme uvést, jaké informace program očekává. Za komentářem vypíše symbol ?, čím dává najevo, že očekává vstup informací z klávesnice.

Příklad:

```
INPUT "vloz prumer:";PRUMER
INPUT "vloz A,B,C:";A,B,C
```

Upozornění:
V seznamu proměnných nemůže být prvek indexované proměnné. Nelze ani číst informace do řetězcové proměnné od určité pozice. Tyto operace je třeba realizovat prostřednictvím pomocných proměnných a přiřazovacích příkazů:

Příklad:

```
10 DIM POLE(5)
20 COM RETEZ$(20):POM$(5)
30 INPUT "c.prvku,hodnota:";I,P:POLE(I)=P
40 INPUT "pozice,text:";P,POM$
50 RETEZ$(p)=POM$
60 PRINT POLE(I), RETEZ$(P)
```

V případě, že ukládáme příkazem PRINT informace do souboru, který budeme později číst příkazem INPUT, musíme si uvědomit, že do souboru je nutno zapsat mezi jednotlivé informace i čárky. Jinak dojde při jeho čtení k chybě.

Příklad:

```
10  REM zapis
20  COM RADEK$(40)
30  OPEN≠1,8,0,"C:"
40  FOR I=1 TO 4
50     INPUT "vloz:";RADEK$
60     PRINT≠1,I;",";RADEK$
70  NEXT I
80  CLOSE≠1
90  END
```

```
10  REM cteni
20  COM RADEK$(40)
30  OPEN≠1,4,0,"C:"
40  FOR I=1 TO 4
50     INPUT≠1,A,RADEK$
60     PRINT A, RADEK$
70  NEXT I
80  CLOSE≠1
90  END
```

V příkaze 60 PRINT=1,I;",";RADEK$ je do výstupního soubou zapsána znaková konstanta ",", čím je dosaženo oddělení informací pro příkaz INPUT.

8. 3 Bytově orientované V/V příkazy

Druhou skupinu V/V příkazů tvoří příkazy, které přenášejí jeden nebo skupinu Bytů mezi souborem a pamětí počítače. Jejich výhodou oproti příkazům V/V řízených seznamem je větší rychlost přenosu informací.

syntax:

```
<Bytové V/V příkazy>:=|<příkaz GET>|<příkaz PUT>|
                      |<příkaz BGET>|<příkaz BPUT>|
                      |<příkaz %GET>|<příkaz %PUT>|
```

8. 3. 1 Příkaz GET

Příkaz GET přečte ze vstupního souboru jeden Byt a uloží jej do aritmetické proměnné.

syntax:

```
<příkaz GET>:=GET [≠<ids>,]<ap>
```

Po provedení příkazu je v : ap: zapsán kód ATASCII znaku, který byl ze vstupního souboru přečten.

Příklad:

```
10 REPEAT
20    PRINT "stlac klavesu!": GET KEY
30    PRINT "znak: ";CHR$(KEY);" ma kod:";KEY
40 UNTIL CHR$(KEY)="N"
```

Poznámka:
Často potřebujeme chod programu zastavit a po stlačení klávesy pokračovat v chodu. K tomu lze využít příkazu GET.

Příklad:

```
10 PRINT "key:";:GET Q
20 IF CHR$(Q)="E" THEN ? "konec!":END
   ...
```

8. 3. 2 Příkaz PUT

Příkaz PUT zapisuje do výstupního souboru jeden Byt, jehož hodnota je dána výsledkem ar. výrazu.

syntax:
<příkaz PUT>:=PUT [≠<ids>,]<av>

Příklad:

```
10  INPUT "kod:";KOD
20  IF KOD: 256 AND KOD >=0 THEN PUT KOD
```

Poznámka:
Příkazy PUT, které pracují se souborem typu "E:" (textový editor), umožňují provádět i řízení editace pomocí kódů editačních kláves (INS,CLEAR, TAB ...).

Příklad:

```
10  CLS
20  FOR I=1 TO 20
30     POSITION 2,I:PRINT I, "————————"
40  NEXT I
50  GET Q
60  POSITION 0,10:PUT 156: PUT 157:REM vymaz 10. radku
70  POSITION 0,20
```

8. 3. 3 Příkaz BGET

Jestliže chceme pracovat se soubory většího rozsahu, potom k realizaci V/V operací jsou nejvhodnější bytové příkazy V/V, které provádějí přenos skupiny Bytů. Příkaz BGET provádí přenos skupiny Bytů ze vstupního souboru a ukládá je do paměti od stanovené adresy.

syntax:
<příkaz BGET<:=BGET [≠<ids>,]<adr>,<počet Bytů>

kde <počet Bytů> je ar. výraz, jehož hodnota určuje počet Bytů, které

budou přečteny ze vstupního souboru a zapsány do paměti na adresy:
<adr>...<adr+počet Bytů–1>

Příklad:

```
 5  POCET=100
10  COM POLE$(POCET)
20  PRVA=ADR(POLE$)
30  OPEN≠1,4,0,"C:"
40  BGET≠1,PRVA,POCET
```

Poznámka:
Při používání příkazu BGET musíme dávat pozor na to, abychom správně zadali jeho parametry. Při nesprávném zadání adresy a počtu můžeme provést přepsání systému a dojde k jeho havárii.

8. 3. 4 Příkaz BPUT

Příkaz BPUT povádí výpis obsahu skupiny Bytů paměti od určené adresy do výstupního souboru.

syntax:
<příkaz BPUT>:=BPUT [≠<ids>,]<adr>,<počet Bytů>

Následující příklad provede výpis obsahu video RAM (tj. informací zobrazených na obrazovce) do výstupního souboru a zpátky.

Příklad:

```
10  COM RADEK$(30)
20  GRAPHICS 0: POSITION 10,10: PRINT "text"
55  ...
60  ? "zapis na mgf"
65  OPEN≠1,8,0,"T:VIDEORAM"
70  BPUT≠1,DPEEK(741),992
80  CLOSE≠1
90  ...
100 CLS:? "cteni z mgf"
110 OPEN≠1,4,0,"T:VIDEORAM"
120 BGET≠1,DPEEK(741),992
130 CLOSE≠1
140 END
```

8. 3. 5 Příkaz %GET

Příkazy %GET a %PUT provádějí přenos skupiny 6 Bytů, ve kterých je obsažena aritmetická hodnota. Umožňují tedy provádět rychlý přenos číselných hodnot mezi V/V soubory a aritmetickými proměnnými.

Příkaz %GET provede přečtení skupiny 6 Bytů ze vstupního souboru a uloží jejich číselnou hodnotu do aritmetické proměnné.

syntax:
<příkaz %GET>:=%GET [≠<ids>,]<ap>

Příklad:

```
10  OPEN≠1,4,0,"D:CISLA"
30  %GET≠1,POM
40  PRINT POM
60  CLOSE≠1
```

8. 3. 6 Příkaz %PUT

Příkaz %PUT provádí zápis skupiny 6 Bytů, která obsahuje výslednou hodnotu aritmetického výrazu.

syntax:
<příkaz %PUT>:=%PUT [≠<ids>,]<av>

Příklad:

```
10  OPEN≠1,8,0,"D: CISLA"
20  INPUT "a,b,c,d,e:";A,B,C,D,E
30  %PUT≠1,A*B+C: %PUT≠1,D
50  CLOSE≠1
```

8. 4 Vnitřní soubor

TB umožňuje pracovat se souborem informací, které jsou součástí programu a vytvářejí vnitřní soubor dat programu. Nejčastěji jsou součástí vnitřního souboru počáteční hodnoty a kódy strojových instrukcí podprogramů.

8. 4. 1 Příkaz DATA

Vnitřní soubor je tvořen množinou konstant, které jsou zapsány v příkazech DATA. Příkaz DATA může být umístěn kdekoliv v programu (nejčastěji na jeho konci). Při chodu programu je příkaz DATA ignorován, je to nevýkonný příkaz sloužící k uložení informací vnitřního souboru do programu.

syntax:
<příkaz DATA>:=<čř> DATA |<ac>|<$c>|[,|<ac>|<$c>|...]

Příklad:

```
130  DATA 3.1415,text,1.85E7,ahoj
140  DATA 0,2,1.25,-17.6
```

Poznámka:
Znakové konstanty, které jsou umístěny v příkaze DATA, se nedávají do uvozovek. Totéž platí i pro vstup znakových konstant v příkaze INPUT.

8. 4. 2 Příkaz READ

Příkaz READ provádí čtení informací z vnitřního souboru a zapisuje je do aritmetických nebo znakových proměnných, které jsou uvedeny v seznamu (je řízen seznamem). Příkaz READ provádí postupně (sekvenčně) čtení informací z vnitřního souboru od začátku nebo od místa, na které ukazuje příkaz RESTORE.

syntax:
<příkaz READ>:=READ|<ap>|<$p>|[,]<ap>|<$p>|...]

Příklad:

```
10  COM NAZEV$(5),TEXT$(20)
20  READ NAZEV$,PI,TEXT$
    ...
100 DATA Kruh,3.1415,Doplnkovy text
```

S příkazem READ pracujeme podle obdobných zásad, jako s příkazem INPUT.

8. 4. 3 Příkaz RESTORE

Příkaz RESTORE provádí určení místa ve vnitřním souboru, od kterého následující příkazy READ začnou číst informace.

syntax:
<příkaz RESTORE>:=|RESTORE ≠<návěští>|RESTORE<av>|

kde <návěští> určuje, za kterým návěštím se nalézá první příkaz DATA, od kterého následující příkaz READ začne číst informace z vnitřního souboru.

<av>je aritmetický výraz, jehož hodnota určuje číslo řádku, za kterým se nalézá první příkaz DATA, od kterého příkaz READ začne číst informace z vnitřního souboru.

Není-li příkaz RESTORE uveden, čte se vnitřní soubor od začátku.

Příklad:

```
10  COM JMENO$(10)
20  RESTORE≠POC.PODMINKY
30  REPEAT
40     READ JMENO$,VYSKA,VAHA
50     PRINT JMENO$,VYSKA,VAHA
60  UNTIL JMENO$(1,2)="/E"
99  ≠POC.PODMINKY
100 ...
101 DATA Pavel,1.70,68
102 DATA Petr,1.75,70
103 DATA Ondra,1.60,55
    ...
140 DATA /E,0,0
```

```
10  COM JMENO$(10)
20  INPUT "cislo:";C
30  RESTORE 100+C
40  READ JMENO$,VYSKA
50  READ VAHA
60  ?JMENO$,VYSKA,VAHA
100 ...
101 DATA Pavel, 170,68
102 DATA Petr, 175, 70
103 DATA Ondra,160,55
    ...
```

8. 5 Speciální V/V příkazy

Speciální příkazy jsou určeny pro práci se soubory na jediném typu zařízení. Výhodou jejich použití je zvětšení přehlednosti programu a vedou k zjednodušení programu. Díky své specializaci dokáží realizovat širší spektrum funkcí. U některých z nich není třeba, aby předcházel příkaz OPEN. Tento příkaz se provede automaticky při použití příkazu V/V.

syntax:

```
<speciální V/V příkazy<:=|<příkaz LPRINT>|<příkaz LOCATE>|
                         |<příkaz CLS>|<příkaz NOTE>|
                         |<příkaz POINT>|
```

8. 5. 1 Příkaz LPRINT

Příkaz LPRINT provádí výstup informací na tiskárnu. Je to příkaz výstupu informací řízený seznamem. Pracuje obdobným způsobem jako příkaz PRINT.

syntax:

```
<příkaz LPRINT>:=LPRINT<seznam výrazů>
```

Příklad:

```
LPRINT A$, B*C, TEXT$(4,10)
```

8. 5. 2 Příkaz LOCATE

Příkaz LOCATE provádí vstup informací z obrazovky do : ap:, tj. provede přečtení jednoho pixelu z místa obrazovky, které je určeno souřadnicemi : sx: a : sy:. Obrazovka slouží v příkazu LOCATE jako vstupní zařízení.

syntax:
<příkaz LOCATE>:=LOCATE<sx>,<sy>,<ap>

Kde <sx> a <sy> jsou aritmetické výrazy, jejichž hodnoty určují souřadnice na obrazovce, odkud bude informace přečtena.

Jestliže pracuje obrazovka v textovém módu, je do : ap: zapsán kód přečteného znaku (textového pixelu). Pracuje-li obrazovka v grafickém módu, je do : ap: zapsáno číslo barvy barvového pixelu.

Příklad:

```
10  GRAPHICS 0
20  POSITION 5,5:PRINT "ABCDE"
30  LOCATE 5,5,A
40  POSITION 10,10: PRINT CHR$(A)
```

8. 5. 3 Příkaz CLS

Příkaz CLS provádí výmaz obrazovky, tj. provádí výmaz souboru na obrazovce (nuluje videoRAM). Po jeho provedení se celá obrazovka vybarví barvou podkladu.

syntax:
<příkaz CLS>:=CLS

Příklad:

```
PRINT "za 2 sec vymaz": PAUSE 100: CLS
```

8. 5. 4 Příkaz NOTE

Příkaz NOTE pracuje s diskovými soubory a do aritmetických proměnných ukládá informace o tom, na kterém sektoru a Bytu je prováděna V/V operace. Ukazuje, ve které části souboru se právě nacházíme.

syntax:
<příkaz NOTE>:=NOTE≠<ids>,<ap.sektor>,<ap.byt>

kde <ap. sektor> a <ap. byt> jsou : ap:, které po provedení příkazu NOTE budou obsahovat číslo sektoru a číslo Bytu, kde se provádí V/V operace.

Příklad:

```
20  OPEN≠1,4,0,"D: POKUS"
    ...
80  NOTE≠1,SEC,BYT:PRINT SEC,BYT
```

8. 5. 5 Příkaz POINT

Příkaz POINT umožňuje realizovat přímý přístup k informacím uloženým v diskových souborech. Určuje konkrétné místo v souboru, odkud bude informace čtena nebo kam bude zapsána.

syntax:
<příkaz POINT>:=POINT≠<ids>,<sektor>,<byt>

kde <sektor> a <byt> jsou aritmetické výrazy, jejichž hodnota určuje číslo sektoru a číslo bytu na disku, kde bude provedena následující V/V operace.

Příklad:

```
10  COM BUFFER$(100)
20  OPEN≠1,8,0,"D:POKUS"
    ...
30  SEC=2 : BYT=8 : POINT≠1, SEC, BYT
90  PRINT≠1,BUFFER$
```

Poznámka:
Příkaz POINT plní obdobnou funkci jako příkaz RESTORE u vnitřního souboru a příkaz POSITION u souborů na obrazovce (typu "S:" nebo "E:").

9 PŘÍKAZY GRAFIKY

Význam grafické úpray výstupních informací zpracovaných na počítači v současné době prudce roste. Hlavní efekt, který počítačová grafika přináší, spočívá v tom, že uživatel bez potřeby úzké specializace může v grafické formě obsáhnout a v logických souvislostech pochopit mnohem větší množství informací, než by to bylo možno na daném prostoru zaznamenat textem nebo souborem číselných hodnot.

Počítače ATARI 800/130 jsou vybaveny speciálními obvody, které umožňují použít širokou škálu grafického zobrazení výstupních informací na obrazovce, tj. výstupu informací do souborů na zařízení typu S a E.

Příkazy grafiky tvoří zvláštní skupinu V/V příkazů, které realizují grafický výstup informací.

9. 1 Základní pojmy

<TV bod>
Počítač ATARI 800/130 standartně rozděluje obrazovku na 192 řádků a na každém řádku pracuje s 320 body. TV bod je tedy nejmenší technicky

zobrazitelný element na obrazovce. TV body vytváří na obrazovce matici:

```
0,0          →          319,0

↓            <pixel>
y

0,191                   319,191
```

<pixel>
Pixel je nejmenší element, který je možno příkazy grafiky zapsat do výstupního souboru. Pixel je tvořen jedním nebo maticí TV bodů, jejichž počet uspořádání a charakter je závislý od zvoleného módu.

<charakter pixelu>:=|<textový pixel>|<barvový pixel>|

<textový pixel> obsahuje jeden znak z množiny znaků daného módu (textového módu). Po přenosu textového pixelu do výstupního souboru se na obrazovce objeví znak. Tyto pixely jsou do výstupního souboru zapisovány příkazem PRINT.

<barvový pixel> obsahuje informaci o barvě zobrazení daného prostoru, který pixel na obrazovce zabírá. Jeden barvový pixel je do výst. souboru zapsán příkazem PLOT.

Příklad:
GRAPHICS 2: POSITION 2,2: PRINT#6,"T":PAUSE 100
GRAPHICS 3: COLOR 1: PLOT 2,2

<velikost pixelu> je dána počtem x*y TV bodů, které pixel na obrazovce zabírá. Přitom x – počet bodů na řádku (sloupců) a y – počet řádek.

<souřadnice x,y>
Příkazy grafiky pracují se souborem s přímým přístupem, tzn. můžeme informace zapisovat do konkrétního místa souboru (na určené místo obrazovky). Soubor je tvořen maticí pixelů a příkazem GRAPHICS určujeme jejich velikost, tj. maximální počet sloupců a řádků matice pixelů.

syntax:
<souřadnice x>:=|0|1|...|<sx max>|
<souřadnice y>:=|0|1|...|<sy max>|

Příklad:

```
      0    1    2    3    4    ...
  0
  1              *                    souřadnice pixelu <sx>=2
  2                                                     <sy>=1
  ...
```

<textové okénko>
Textové okénko je soubor typu E, který je otevřen pro dolních 320 x 32 TV bodů obrazovky. Textové okénko umožňuje pokračovat uživateli v komunikaci s počítačem na dolních čtyřech řádcích rezidentního souboru.

Příklad:
GRAPHICS 7 : COLOR 1: PLOT 10, 10: DRAWTO 30,30

V horní části obrazovky je otevřen grafický soubor, do kterého byla příkazy PLOT a DRAWTO zapsána množina pixelů tvořící čáru. Dolní část obrazovky tvoří textové okénko, kde můžeme pokračovat v zadávání dalších příkazů nebo vstupních údajů.

Poznámka:
Pokud výstupní soubor zabere celou obrazovku (není vytvořeno textové okénko) potom po ukončení programu dojde k vymazání tohoto souboru. Systém provede automatické otevření rezidentního, aby mohl komunikovat s uživatelem.

Příklad:
GRAPHICS 23: COLOR 1: PLOT 10,10: DRAWTO 30,30: PAUSE 200

<videoRAM>
VideoRAM je oblast paměti tvořící vyrovnávací paměť (buffer) pro speciální obvody (ANTIC a GTIA), které 50 x za sekundu přenášejí její obsah do souboru na obrazovce. Velikost videoRAM a její umístění závisí od zvoleného módu.

9. 2. Příkaz GRAPHICS

Příkaz GRAPHICS určuje jaká bude organizace souboru na obrazovce. Provádí jeho otevření (IOCB č. 6, tj. : ids: = 6), vymazání videoRAM (obrazovka se zbarví do barvy podkladu) a nastaví standartní barvy do barvových registrů.

syntax:
<příkaz GRAPHICS>:=GRAPHICS <mod>
<mod>:=|0|1|...|31|

kde **<mod>** je aritmetický výraz, jehož hodnota určuje způsob organizace výstupního souboru na obrazovce. Tj. určuje číslo dodatečné charakteristiky příkazu OPEN, který je součástí provedení příkazu GRAPHICS.

Příklad:
GRAPHICS 18: PRINT≠6;"TEXT":GET Q: END je stejné jako:
OPEN≠6,8,2,"S:":PRINT≠6;"TEXT":GET Q

Mód tedy určuje velikost, charakter a rozmístění (matici) pixelů ve výst. souboru. Podle charakteru pixelů, které budeme do souboru zapisovat, se módy dělí do dvou základních typů:

syntax:
<typ módu>:=|<textový mód>|<grafický mód>|

9. 2. 1 Textové módy

Textové módy realizují výstup textových informací (tj. zápis textových pixelů) na obrazovku. Vlastní zápis do výst. souboru je realizován příkazem PRINT, přičemž umístění textového pixelu v souboru (na obrazovce) je určeno příkazem POSITION.

Tabulka textových módů

číslo módu	velikost pixelu	rozdělení : sx:	obrazovky : sy:	počet barev	velikost videoRAM	poznámka
0	8x8	0...39	0...23	1/2	992	
1	16x8	0...19	0...19	5	674	+text. okno
2	16x16	0...19	0....9	5	424	+text. okno
12	8x8	0...39	0...19	5	1154	+text. okno
13	8x16	0...39	0....9	5	664	+text. okno
17	16x8	0...19	0...23	5	672	
18	16x16	0...19	0...11	5	420	
28	8x8	0...39	0...23	5	1152	
29	8x16	0...39	0...11	5	660	

Poznámka:
1/2 je 1 barva ve dvou stupních jasu.

Podle způsobu práce lze textové módy rozdělit do tří skupin:

a) mód 0 – je základním módem počítače, je bezprostředně spojen s rezidentním souborem (tvoří jeho zobrazovací část na obrazovce). Umožňuje zobrazení všech znaků z abecedy počítače a to okamžitě po stlačení klávesy nebo příkazem PRINT. Příkaz GRAPHICS 0 provádí otevření rezidentního souboru.

b) módy 2, 3, 18, 19 – umožňují provádět výstup zvětšených textových pixelů, tzn. vypisovat na obrazovce text většími písmeny. Příkaz GRAPHICS : mod: otevírá výstupní soubor : ids:=6. Proto zápis informací do souboru je realizován příkazem PRINT≠6;: seznam:. Volba barev v těchto módech není prováděna příkazem COLOR, ale způsobem zápisu znaků do seznamu v příkaze PRINT. Množina zobrazitelných znaků je u těchto módů omezena.
Barva 1: normální velká písmena
Barva 2: normální malá písmena.
Barva 3: inverzní velká písmena.
Barva 4: inverzní malá písmena.
Barva 5: pozadí (podklad).

Příklad:

```
10  GRAPHICS 2
20  PRINT≠6;"VELKA NORMALNI"
30  PRINT≠6,"mala normalni"
40  PRINT≠6;"VELKA INVERZNI"
50  PRINT≠6;"mala inverzni"
```

c) módy 12, 13, 28, 29 – tvoří skupinu pseudotextových módů. Výstup textových pixelů je realizován výkonnými příkazy grafiky. Přitom znak, který bude v pixelu obsažen, je dán kódem znaku uvedeným v příkaze COLOR.

Příklad:

```
10  GRAPHICS 12
20  COLOR ASC("A"):PLOT 5
30  COLOR ASC("Q"):PLOT 7,5
40  COLOR ASC("*"):TEXT 0,0,"Xaver"
```

Poznámka:
Podobnou volbu zobrazení pro příkaz TEXT můžeme realizovat i v ostatních textových módech.

Příklad:

```
10  GRAPHICS 1
20  COLOR ASC("*"):TEXT 0,0 TIME$
```

9. 2. 2 Grafické módy

V grafických módech je realizován zápis barvových pixelů do výstupního souboru na obrazovce. Zápis je prováděn výkonnými příkazy grafiky PLOT, DRAWTO, PAINT, CIRCLE, TEXT, FILLTO, přičemž barva a umístění pixelů je dána příkazy COLOR, SETCOLOR a POSITION.

Tabulka grafických módů.

číslo módu	velikost pixelu	rozdělení : sx:	obrazovky : sy:	počet barev	velikost videoRAM	poznámka
3	8x8	0...39	0...19	4	424	+text. okno
4	4x4	0...79	0...39	2	694	+text. okno
5	4x4	0...79	0...39	4	1174	+text. okno
6	2x2	0...159	0...79	2	2174	+text. okno
7	2x2	0...159	0...79	4	4190	+text. okno
8	1x1	0...319	0...159	1/2	8112	+text. okno
9	4x1	0...79	0...191	1/16	8138	
10	4x1	0...79	0...191	9	8138	
11	4x1	0...79	0...191	16	8138	
14	2x1	0...159	0...159	2	4270	+text. okno

15	2x1	0...159	0...159	4	8112	+text. okno
19	8x8	0...39	0...23	4	432	
20	4x4	0...79	0...47	2	696	
21	4x4	0...79	0...47	4	1176	
22	2x2	0...159	0...95	2	2184	
23	2x2	0...159	0...95	4	4200	
24	2x1	0...159	0...191	2	4296	
24	1x1	0...319	0...191	1/2	8138	
31	2x1	0...159	0...191	4	8138	

Poznámka:
1/n je jedna barva v n–stupních jasu.
a) módy 8, 24 jsou základními grafickými módy, kde pixel je tvořen jediným TV bodem. Barvový pixel v těchto módech je tvořen jedinou barvou ve dvou stupních jasu.

Příklad:
```
10  GRAPHICS 8
20  COLOR 1: CIRCLE 160,80,70: PAINT 160,80
30  COLOR 2: PLOT 0,0: DRAWTO 319,159
40  COLOR 1: PLOT 319,0: DRAWTO 0,159
```

b) módy 4,6,14,20,22,30 tvoří skupinu dveubarevných grafických módů. Při zápisu barvového pixelu můžeme příkazem COLOR volit jednu ze dvou barev. Musíme si uvědomit, že jedna z těchto dvou barev tvoří podklad obrazu, tj. po provedení příkazu GRAPHICS zabírá celou obrazovku.

Příklad:
```
10  GRAPHICS 4
20  COLOR 1: CIRCLE 40,20,30,15: PAINT 40,20
30  COLOR 2: TEXT 36,16,"Text"
```

c) módy 3,5,7,15,19,21,23,31 jsou skupina standartních čtyřbarevných módů. Při zápisu barvového pixelu můžeme vybírat příkazem COLOR jednu ze čtyř barev. Jedna z těchto čtyř barev je barvou podkladu. Tyto grafické módy nám poskytují širokou škálu možností grafického barevného výstupu informací na obrazovku.

Příklad:
```
10  GRAPHICS 7
20  FOR BARVA=1 TO 3
30     COLOR BARVA
40     CIRCLE BARVA*40,40,30: PAINT BARVA*40,40
50     FOR TX=1 TO 4
60        COLOR TX
70        TEXT BARVA*40,TX*10+15,"aBc"
80     NEXT TX
90  NEXT BARVA
```

d) módy 9, 10, 11 – tvoří skupinu speciálních grafických módů, které poskytují velkou nabídku barev pro zápis barvového pixelu do výstupního souboru. Tvar pixelu 4x1 je předurčuje zejména k tvorbě sloupcových diagramů.

Příklad:

```
10  GRAPHICS 10
20  FOR I=1 TO 8
30     READ BARVA,JAS: POKE 704+I, BARVA*16+JAS
40     COLOR I
50     FOR X=I*8 TO I*8+5
60        PLOT X,0: DRAWTO X,180
70     NEXT X
80  NEXT I
90  GET Q:END
100 DATA 0,4,15,2,3,2,4,2,7,2,8,14,13,10,12,2
```

9. 2. 3 Uchování obsahu videoRAM

Příkaz GRAPHICS provádí výmaz videoRAM, nastavuje obraz do barvy podkladu. Pokud potřebujeme obsah videoRAM uchovat, zvětšíme hodnotu : av:, kterým určujeme : mod: o 32. Potom obsah videoRAM po provedení příkazu GRAPHICS : mod: +32 zůstane zachován.

Příklad:

```
10  GRAPHICS 7
20  COLOR 1: PLOT 5,5:DRAWTO 50,50
30  PAUSE 50
40  GRAPHICS 7
50  COLOR 2: PLOT 50,5: DRAWTO 5,50
60  END
```

Po skončení tohoto programu zůstane na obrazovce jedna čára. Chceme-li zachovat obě vykreslené čáry, změníme řádek 40 GRAPHICS 7+32.

9. 3 Přípravné příkazy

Přípravné příkazy grafiky neprovádějí zápis informací do výstupního souboru, tj. nemění obsah obrazovky.

syntax:

<přípravný příkaz>:|<příkaz POSITION>|<příkaz COLOR>|
|<příkaz SETCOLOR>|

9. 3. 1 Příkaz POSITION

Příkaz POSITION slouží k určení místa ve výstupním souboru (místa na obrazovce). To znamená, že příkazem POSITION určujeme souřadnice x (sloupce) a y (řádku), kam bude pixel následujícím výkonným příkazem zapsán.

syntax:
<příkaz POSITION>:=POSITION <sx>,<sy>

Příklad:

```
10  CLS
20  POSITION 5,8: INPUT "vloz A=";A
30  POSITION 25,2: PRINT "A=";A
40  POSITION 2,22:? "stlac klavesu";:GET Q
50  GRAPHICS 7: COLOR 1
60  POSITION 5,5: DRAWTO 50,50
```

9. 3. 2 Příkaz COLOR

Příkaz COLOR určuje v grafických módech, který barvový registr bude použit pro zápis informací, které budou následujícími výkonnými příkazy odeslány do výst. souboru.

syntax:
<příkaz COLOR>:=COLOR :av:

kde hodnota : av: určuje, z kterého barvového registru budou vybrány údaje o barvě a jasu zobrazení pixelu pro následující výkonné příkazy.

Příklad:

```
10  GRAPHICS 5
20  FOR BARVA A=1 TO 3
30     COLOR BARVA
40     TEXT 5, BARVA *10, BARVA
50     TEXT 20, BARVA *10, ".barva"
60  NEXT BARVA
```

9. 3. 3 Příkaz SETCOLOR

Pro zobrazení informací na obrazovce můžeme vybírat z množiny 128 barev. Množina barev je dána 16 základními barvami a každá barva může být zobrazena v 8 stupních jasu. Výběr barev z této množiny, které můžeme v daném módu používat, provádíme příkazem SETCOLOR.

Příkaz SETCOLOR provádí zápis čísla barvy a stupně jasu do barvového registru. Obvod GTIA potom na základě obsahu těchto barvových registrů provádí "zabarvení" informací při přenosu z videoRAM na obrazovku.

Tabulka barev:

0	šedá	8	modrá základní
1	zlatá	9	světlemodrá
2	oranžová	10	tyrkysová
3	červenooranžová	11	modrozelená
4	růžová	12	zelená
5	purpurová	13	žlutozelená
6	červenooranžová	14	oranžovozelená
7	tmavěmodrá	15	světleoranžová

syntax:
<příkaz SETCOLOR>:=SETCOLOR<barvový reg.>,<barva>,<jas>

kde <barvový reg.> je : av:, jehož hodnota určuje, do kterého registru budou zapsány údaje o barvě a jasu.

kde <barva> je : av:, jehož hodnota udává číslo barvy, která bude zapsána do barvového reg. <barva>:=|0|1|...|15|

kde <jas> je : av:, jehož hodnota udává stupeň jasu, kterým bude zvolená barva zobrazena. <jas>:=|0|2|...|14|
Čím vyšší je hodnota : jas:, tím světlejší bude vykreslení barvy na obrazovce.

Zobrazení barev na barevném TV přijímači závisí na jeho správném nastavení. Proto je třeba při volbě barev zohlednit nastavení TV přijímače.

Musíme si uvědomit, že hodnota : av: v příkaze COLOR není totožná s číslem barvového registru v příkaze SETCOLOR. Proto je třeba se řídit podle následující tabulky.

čísla módů	počet barev	číslo SETCOLOR	číslo COLOR	poznámka
0,8,24	1/2	1	1	jas písma, kresby
		2	2	barva a jas podkladu
		4	–	barva okraje obrazovky
1,2,17,18	5	0	–	VELKA PISMENA NORMALNI
		1	–	mala pismena normalni
		2	–	VELKA PISMENA INVERZNI
		3	–	mala pismena inverzni
		4	–	barva podkladu

4,6,14	2	0	1	1. barva, kresba
20,22,30		1	–	jas písma v okně
		2	–	podklad okna
		4	2	2. barva, podklad
3,5,7,15	4	0	1	1. barva
19,21,23		1	2	2. barva, písmo v okně
31		2	3	3. barva, podklad okna
		4	4	4. barva, podklad
9	1/16	4(barva)	0...15	číslo COLOR = jas
11	16	4 (jas)	0...15	číslo COLOR = číslo barvy

U módu 10 je mechanizmus nastavování barev jiný. Mód 10 pracuje nejen s 5 základními barvovými registry, ale i s registry PMG (pohybové grafiky.) Tyto registry se nastavují příkazy POKE : adr:, : barva: *16+: jas: (viz př. v kap. 9.2.2 d).

9. 4 Výkonné příkazy grafiky

Výkonné příkazy grafiky provádějí přenos jednoho nebo množiny barvových pixelů do výst. souboru na obrazovce.

syntax:
<výkonný př. grafiky>:=|<příkaz CIRCLE>|<příkaz DRAWTO>|
|<příkaz FILLTO>|<příkaz PAINT>|
|<příkaz PLOT>|<příkaz TEXT>|

9. 4. 1 Příkaz PLOT

Příkaz PLOT provádí přenos jednoho barvového pixelu do výstupního souboru. Místo umístění pixelu ve výst. souboru na obrazovce je určeno souřadnicemi : sx: a : sy:. Barva pixelu je určena nejbližším předcházejícím příkazem COLOR.

syntax:
<příkaz PLOT>:=PLOT<sx>,<sy>

kde <sx> a <sy> jsou : av:, jejichž hodnoty udávají umístění pixelu ve výst. souboru. Maximální dovolené hodnoty jsou dány rozsahem : sy: a : sx: pro daný mód.

Příklad:
GRAPHICS 3: COLOR 1: PLOT 5,5: PLOT 10,6

9. 4. 2 Příkaz DRAWTO

Příkaz DRAWTO provádí vykreslení čáry z jednoho místa (které je dáno posledním zápisem do souboru) na druhé, jehož souřadnice jsou určeny v tomto příkaze. Provádí tedy spojení dvou míst na obrazovce čarou, která se skládá z množiny barvových pixelů, jejichž barva je určena předcházejícím příkazem COLOR.

syntax:
<příkaz DRAWTO>:=DRAWTO <sx>,<sy>

kde <sx>, <sy> jsou : av:, jejichž hodnoty udávají souřadnice místa, kam bude čára vedena, tj. souřadnice posledního pixelu čáry.

Příklad:
GRAPHICS 7:COLOR 1: PLOT 5,5: DRAWTO 70,70: DRAWTO 70,5

Poznámka:
Nejčastější chybou při použití příkazu DRAWTO je nesprávné nastavení výchozího místa čáry. Tato souřadnice je určena posledním výkonným příkazem grafiky. Proto je vhodné vždy před příkazy DRAWTO zařadit příkaz POSITION, kterým nastavíme výchozí bod.

Příklad:
GRAPHICS 7: COLOR 1: POSITION 10,10: DRAWTO 10,40

9. 4. 3 Příkaz CIRCLE

Příkaz CIRCLE provádí vykreslení elipsy nebo kružnice, jejichž střed je dán : sx: a : sy:. Provádí tedy přenos množiny pixelů do výst. souboru na obrazovce, které jsou organizovány do elipsy.

syntax:
<příkaz CIRCLE>:=CIRCLE<sx>,<sy>,<poloměr X>[,<poloměr Y>]

kde <sx>, <sy> jsou :av:, jejichž hodnoty udávají souřadnice středu elipsy nebo kružnice.

<poloměr X>, <poloměr Y> udávají velikosti poloos elipsy ve směru x a y. Pokud není hodnota <poloměr Y> uvedena, příkaz CIRCLE vykreslí kružnici.

Příklad:
GRAPHICS 7: COLOR 1: CIRCLE 40,40,30,10: REM elipsa
GRAPHICS 7: COLOR 1: CIRCLE 40,40,30: REM kružnice

9. 4. 4 Příkaz PAINT

Příkaz PAINT provádí vyplnění ohraničené plochy barvou, která je dána předcházejícím příkazem COLOR.

syntax:
<příkaz PAINT>:=PAINT<sx>,<sy>

kde <sx>, <sy> jsou : av: udávající souřadnice libovolného místa ležícího uvnitř ohraničené plochy, která má být vyplněna.

Příklad:

```
10  GRAPHICS 7: COLOR 1
20  PLOT 10,10: DRAWTO 50,10: DRAWTO 50,50
30  DRAWTO 10,50: DRAWTO 10,10
40  COLOR 2: PAINT 11,11
50  COLOR 3: CIRCLE 30,30,10
60  COLOR 4: PAINT 30,30
```

9. 4. 5 Příkaz TEXT

Příkazem TEXT je prováděn zápis znakových řetězců nebo číselných hodnot do výstupních souborů v grafických módech. To znamená, že odstraňuje jeden z nedostatků Atari Basicu, který neumožňoval normálním způsobem provést popis obrazů. Příkaz TEXT provádí přenos matic (8x8 pixelů) obsahující znaky.

syntax:
<příkaz TEXT>:=TEXT<sx>,<sy>,|<av>|<$v>|

kde <sx>, <sy> udávají souřadnice pixelu, od kterého bude zobrazena hodnota : av: nebo : $v:.

Příklad:

```
10  REM hodiny
20  COM CAS$(6)
30  INPUT "zadej cas(hhmmss):";CAS$
40  TIME$=CAS$:GRAPHICS 4: COLOR 1
50  DO: TEXT 1,1, TIME$:LOOP
```

9. 4. 6 Příkaz FILLTO

Příkaz FILLTO nahrazuje POSITION x,y : XIO 18,≠6,0,0,"S: ".
FCOLOR n provádí volbu barvy pro FILLTO. Odpovídá příkazu POKE 765,n.

syntax:
<příkaz FILLTO>:=FILLTO <sx>,<sy>

10 ZVUKOVÉ PŘÍKAZY

Počítače Atari 800/130 umožňují vytvářet současně čtyři zvukové signály. K tomu slouží čtyři tónové generátory, které modulují signál vysílaný počítačem do TV přijímače. Zesílení a reprodukci tohoto signálu realizují obvody TV přijímače. To znamená, že hlasitost a zabarvení tónu můžeme ovlivňovat jak příkazy TB, tak regulátory zvuku na přijímači.

syntax:
<zvukové příkazy>:=|<příkaz DSOUND>|<příkaz SOUND>|

10. 1 Příkaz DSOUND

Příkaz DSOUND provádí vypnutí zvuku, tzn. provádí vynulování hlasitosti všech čtyř tónových generátorů.

syntax:
<příkaz DSOUND>:=DSOUND

10. 2 Příkaz SOUND

Příkaz SOUND provádí nastavení parametrů výstupního signálu z tónového generátoru.

syntax:
<příkaz SOUND>:=SOUND<generátor>,<tón>,<zkreslení>,<hlas>
<generátor>:=|0|1|2|3|
<tón>:=|0|1|...|255|
<zkreslení>:=|0|2|...|14|
<hlas>:=|0|1|...|15|

kde <generátor> je : av:, jehož hodnota udává, kterému tónovému generátoru příkaz SOUND určuje parametry.

<tón> je : av:, jehož hodnota udává frekvenci (výšku) tónu, který bude tónovým generátorem vyslán do TV přijímače.

<zkreslení> je : av:, jehož hodnota udává tvar výst. signálu. Čisté signály jsou 10 a 14, ostatní obsahují šumy.

<hlas> je : av:, jehož hodnota udává intenzitu signálu (hlasitost), který bude vyslán do přijímače.

Příklad:

```
10  SOUND 0,121,10,10: PAUSE 100
20  SOUND 1,108,10,10: PAUSE 100
30  SOUND 2,96,10,10: PAUSE 100
40  SOUND 3,91,10,10: PAUSE 100
50  DSOUND
60  FOR TON=255 TO 0 STEP -1
70    SOUND 0,TON,14,6:PAUSE 5
80  NEXT TON
90  DSOUND
```

K tvorbě melodií potřebujeme znát, jaké hodnoty : tón: odpovídají notám

Tabulka not:

nota	C	Cis	D	Dis	E	F	Fis	G	Gis	A	Ais	H
tón (CO)	243	230	217	204	193	182	173	162	153	144	136	128
tón (C1)	121	114	108	102	96	91	85	81	76	72	68	64
tón (C2)	60	57	53	50	47	45	42	40	37	35	33	31
tón (C3)	29											

11 SYSTÉMOVÉ PŘÍKAZY

Systémové příkazy slouží k určení základních funkcí činnosti počítače. Zásadním způsobem mění práci počítače, se soubory (tedy i s programem) pracují jako s celky.

Zabezpečují vstup a výstup programu, jeho tvorbu, spuštění, ukončení atd. TB je oproti Atari Basicu významně rozšířen o příkazy systému pracující se soubory na disku. Tím prakticky odpadá časté používání příkazu DOS a zpětné natahování Basicu. Speciální skupina systémových příkazů umožňuje efektivně ověřovat správnost funkce programu.

syntax:

```
<systémové příkazy>:=|<příkazy V/V programů>|
                     |<příkazy tvorby programu>|
                     |<výkonné příkazy systému>|
                     |<speciální příkazy systému>|
                     |<diskové příkazy systému>|
                     |<systémové klávesy>|
```

Poznámka:
Některé z příkazů systému můžeme zařadit i do programu, využít jejich možností ke změně funkcí počítače.

11. 1 Příkazy vstupu a výstupu programu

Všechny informace, se kterými počítač pracuje, jsou uspořádány do souborů. Tedy i program, který je tvořen množinou speciálně určených a uspořádaných informací, je souborem. Příkazy V/V programu realizují zápis tohoto souboru z paměti počítače na vnější zařízení a naopak, jeho čtení z vnějšího zařízení a uložení do uživatelské oblasti paměti počítače.

syntax:
<příkaz V/V dprogramu>:=|<příkaz CLOAD>|<příkaz CSAVE>|
|<příkaz ENTER>|<příkaz LIST>|
|<příkaz LOAD>|<příkaz SAVE>|
|<příkaz BLOAD>|

S vyjímkou příkazu LIST tyto příkazy pracují se zařízením C, T (dataset) a D (disk). Vytvářejí dvojice, kde jeden příkaz provádí zápis programu na výstupní zařízení a druhý jeho čtení do paměti.

11. 1. 1 Příkazy CLOAD a CSAVE

Příkazy jsou určeny pro nahrátí programu z datasetu do paměti počítače (CLOAD) a k uložení programu na magnetofonovou pásku v datasetu (CSAVE). Příkaz CSAVE provádí výpis paměti od první adresy uživatelské oblasti do poslední adresy obsahující program. Příkaz CLOAD ukládá čtený program od první adresy uživatelské oblasti, původní obsah přepisuje.

syntax:
<příkaz CLOAD>:=CLOAD
<příkaz CSAVE>:=CSAVE

11. 1. 2 Příkazy ENTER a LIST

Příkazy ENTER a LIST umožňují vytvářet programy z několika dílčích programů, ty odladit samostatně a na závěr je spojit do jednoho celku. Pracují s jednotlivými programovými řádky, provádí výpis nebo nahrání programu v jeho „zdrojové" formě.

syntax:
<příkaz ENTER>:=ENTER "<vvz>:[<jmeno>]"
<příkaz LIST>:=LIST ["<vvz>:[<jmeno>]"] [<od>[,<do>]]
<předznamenání příkazu LIST>:=*L|+|-|

kde <od>, <do> udávají rozsah : čř: se kterým bude příkaz LIST pracovat.

a) příkaz ENTER

Příkaz ENTER čte program z vnějšího zařízení, který byl na ně zapsán příkazem LIST. Čtení provádí po programových řádcích a na základě : čř: je zařazuje do souboru (programu) v paměti počítače. Pokud se v paměti počítače již programový řádek stejného : čř: nachází, potom provede jeho přepsání. Neprovádí tedy výmaz uživatelské oblasti paměti, kde je uložen program, ale čtené programové řádky podle : čř: do něj zařazuje.

Příklad:
ENTER "D: POKUS"
ENTER "C:"

b) příkaz LIST

Příkaz LIST nám nabízí širokou škálu možností výpisu programu na vnější zařízení. Předznamenání ✻L+ udává, že program bude vypisován v normální formě, která automaticky graficky rozlišuje programové struktury. Předznamenání ✻L– toto rozlišení potlačí (výpis stejný jako v Atari Basicu).

Možnosti:
LIST – výpis celého programu do rezidentního souboru na obrazovku.
LIST 20 – výpis řádku 20 na obrazovku
LIST 20,40 – výpis řádku 20 až 40
LIST "C" – výpis celého programu na dataset 600Bd
LIST "P:" – výpis celého programu na tiskárnu
LIST "T: POKUS" – výpis programu na dataset 2000Bd
LIST "D: POKUS",20,40 – výpis řádku 20 až 40 do souboru POKUS na disku.

Příklad:
30 REM 2. cast programu
40 PRINT "text 2"
60 PRINT "text 4"
LIST "C:" – zápis 1. části programu na dataset
NEW
10 REM 1. cast programu
20 PRINT "text 1"
30 REM bude prepsan
50 PRINT "text 3"
ENTER "C:" – ctení 2. casti programu z datasetu
LIST – vypis celeho programu

11. 1. 3 Příkazy LOAD a SAVE

Zatímco příkazy CLOAD a CSAVE pracují pouze s datasetem (600Bd), příkazy LOAD a SAVE umožňují pracovat se všemi záznamovými zařízeními, přičemž základním je disk.

syntax:
<příkaz LOAD>:=LOAD "<vvz>:[<jmeno>]"
<příkaz SAVE>:= SAVE "<vvz>:[<jméno>]"

Příkazy provádějí stejnou funkci jako příkazy CLOAD/CSAVE. Jsou určeny pro práci s diskovými soubory. Při zápisu na dataset je příkaz SAVE pomalejší než CSAVE, protože přestávky mezi datovými bloky jsou delší.

Příklad:
SAVE "D: POKUS"
LOAD "D: POKUS"
SAVE "C:"

11. 1. 4 Příkaz BLOAD

Příkaz BLOAD provede přečtení programu ve strojovém kódu z disku do paměti počítače. Po jeho provedení dochází k přepisu paměti počítače.

syntax:
<příkaz BLOAD>:=BLOAD "D[2]:<jméno>"

Příklad:
BLOAD "D: PROG1"

11. 2 Příkazy tvorby programu

Příkazy tvorby programu ovlivňují celý program, jeho část nebo mění jeho organizaci.

syntax:
<příkaz tvorby programu>:=|<příkaz NEW>|<příkaz DEL>|
|<příkaz RENUM>|

11. 2. 1 Příkaz NEW

Příkaz NEW provádí výmaz uživatelské oblasti paměti počítače. To znamená, že vymaže i program, který v ní byl uložen.

syntax:
<příkaz NEW>:=NEW

Příklad:

```
10  PRINT "příkaz programu"
NEW
LIST
```

11. 2. 2 Příkaz DEL

Příkaz DEL provádí výmaz určené množiny programových řádků z programu uloženého v paměti počítače.

syntax:
<příkaz DEL>:=DEL <od>, <do>

kde <od>, <do> udává : čř: prvního a posledního řádku, který bude příkazem DEL vymazán z programu.

Příklad:

```
10  PRINT "prvy"
20  A=20: PRINT A,A+RAND(A)
30  REM vymazana cast – zacatek
40  PRINT "vymaz"
50  PRINT "posledni vymazany radek"
60  PRINT "zaver"
DEL 30,50          – vymaz radku 30 az 50
LIST
NEW                – vymaz celeho programu
LIST
```

11. 2. 3 Příkaz RENUM

Příkaz RENUM provádí přečíslování programových řádků. Pokud se v programu vyskytují příkazy GOTO : čř:, provede přečíslování i v těchto příkazech. Přečíslování se provede od prvního určeného řádku do konce programu.

syntax:
<příkaz RENUM>:=RENUM<staré čř>, <nové čř>, <krok čř>

kde <staré čř> udává : čř: v programu, od kterého začíná přečíslování. <nové čř> po provedení příkazu RENUM nahradí v programu : staré čř: <krok čř> udává, o kolik bude zvětšeno následující : čř: oproti : čř: předcházejícího programového řádku.

Příklad:

```
10  REM zacatek
20  A=20
25  B=30
31  PRINT A,B
35  END
RENUM 25,30,10
LIST
```

Poznámka:
Při přečíslování směrem dopředu, tj. když : nové čř: je menší než : staré čř:, může dojít ke změně posloupnosti programových řádků a tedy k chybnému sestavení programu.

11. 3 Výkonné příkazy systému

Výkonnými příkazy systému předáváme řízení chodu počítače jinému systému (např. DOS) nebo programu.

syntax:
<výkonný příkaz systému>:=|<příkaz BRUN>|<příkaz BYE>|
|<příkaz CONT>|<příkaz DOS>|
|<příkaz RUN>|

11. 3. 1 Příkaz BRUN

Příkaz BRUN provede přečtení programu ve strojovém kódu z disku a provede jeho spuštění.

syntax:
<příkaz BRUN>:=BRUN "D[2]:<jmeno>"

11. 3. 2 Příkaz BYE

Příkazem BYE ukončujeme práci TB a počítač přejde do systému SELF TEST, kde můžeme otestovat jeho základní části. Provádí tedy výmaz Turbo Basicu.

syntax:
<příkaz BYE>:=BYE

11. 3. 3 Příkaz CONT

Jestliže došlo k přerušení činnosti programu (po stlačení klávesy BREAK nebo příkazy STOP, END), potom po zadání příkazu CONT bude program pokračovat v další činnosti.

syntax:
<příkaz CONT>:=CONT

Poznámka:
Chceme-li pokračovat po zastavení chodu programu z jiného místa, můžeme to provést příkazy GO≠<návěští> nebo GOTO<čř>.

11. 3. 4 Příkaz DOS

Provedení příkazu DOS se u jednotlivých verzí TB liší. U standartní verze dojde po zadání příkazu DOS k ukončení činnosti TB a do paměti počítače se z disku zavede diskový operační systém. Další práce v TB je možná po opětovném nahrání TB do paměti.

U verzí TBasicDOS a TBasic"T" dojde po zadání příkazu DOS k přepnutí do pseudosystému OS, který nám umožňuje provádět základní funkce diskového OS v Turbo 2000. TB není z paměti počítače vymazán a po stlačení tlačítka RESET můžeme pokračovat v práci s TB.

syntax:
<příkaz DOS>:=DOS

11. 3. 5 Příkaz RUN

Příkazem RUN provádíme spuštění programu. Příkaz RUN provede výmaz uživatelské oblasti počítače za programem, tj. zruší deklarace všech proměnných.

syntax:
<příkaz RUN>:=RUN

11. 4 Speciální příkazy systému

Speciální příkazy systému umožňují efektivní ladění programů. Mohou být zadány jak v systému TB, tak jako součást programu. Po ověření programu se z něj vyřadí.

syntax:
<speciální přík. systému>:=|<příkaz DUMP>|<příkaz TRACE>|

11. 4. 1 Příkaz DUMP

Příkaz DUMP provádí výpis hodnot všech proměnných, které byly v programu deklarovány a výpis všech použitých : id:, tj. návěští a jmen procedur. Umožňuje získat přehled o chodu programu.

syntax:
<příkaz DUMP>:=DUMP "<vvz:>[<jmeno>]"

Příklad:

```
10  TRAP≠CHYBA
20  INPUT A
    ...
90  ≠CHYBA: DUMP "S:":GET Q
```

11. 4. 2 Příkaz TRACE

Příkazem TRACE zadáme systému TB, že při chodu programu vypisuje v hranatých závorkách čísla prog. řádků, které právě vykonává. To nám při ladění programu umožní ověřit si správnou posloupnost provádění příkazů.

syntax:
<příkaz TRACE>:=TRACE|–|+|

kde příkaz **TRACE+** zapíná výpis : čř: při chodu programu a příkaz **TRACE–** výpis : čř: ruší.

Příklad:

```
10  TRACE+
20  FOR I=1 TO 3
30    PRINT I
40  NEXT I
50  TRACE –
    ...
```

11. 5 Diskové příkazy systému

Diskové příkazy systému umožňují provádět základní operace se soubory na disku. Nahrazují tedy příkazy DOSu.

syntax:
<diskové přík. systému>:=|<příkaz DIR>|<příkaz DELETE>|
|<příkaz LOCK>|<příkaz UNLOCK>|
|<příkaz RENAME>|

11. 5. 1 Příkaz DIR

Příkaz DIR provádí výpis obsahu diskového adresáře. Dává tedy uživateli přehled o souborech, které se na disku nacházejí.

syntax:
<příkaz DIR>:=DIR "D[2]:[volba]*.*"

Kde <volba> udává počáteční posloupnost znaků z jmen souborů, které budou do výpisu zařazeny.

Příklad:
DIR "D:*.*" – výpis celého adresáře disku
DIR "D:T*.*" – výpis jmen souborů začínající T

11. 5. 2 Příkaz DELETE

Příkaz DELETE provádí výmaz souboru z disku.

syntax:
<příkaz DELETE>:=DELETE "D[2]:<jméno>"

Příklad:
DELETE "D: POKUS.BAS"

11. 5. 3 Příkaz RENAME

Příkaz RENAME provádí přejmenování souboru na disku. Tj. provádí výměnu starého jména v adresáři za nové.

syntax:
<příkaz RENAME>:=RENAME "D[2]:<staré jméno>,<nové jméno>"

Příklad:
RENAME "D: POKUS.BAS,TEST.BAS"

11. 5. 4 Příkaz LOCK

Příkaz LOCK přidá do adresáře disku ke jménu souboru dodatečnou informaci, která chrání soubor před přepsáním nebo výmazem.

syntax:
<příkaz LOCK>:LOCK "D[2]:<jmeno>"

Příklad:
LOCK "D: TEST.BAS"

11. 5. 5 Příkaz UNLOCK

Příkaz UNLOCK ruší ochranu souboru na disku před přepsáním nebo výmazem.

syntax:
<příkaz UNLOCK>:=UNLOCK "D[2]:<jméno>"

Příklad:
UNLOCK "D: TEST.BAS"

11. 6 Systémové klávesy

Klávesa BREAK a tlačítko RESET mají speciální funkci a ovlivňují chod systému.
<systémové klávesy>:=|<klávesa BREAK>|<tlačítko RESET>|

11. 6. 1 Klávesa BREAK

Stlačením klávesy BREAK provádíme zastavení chodu programu, pokud není předznamenáním ✻B+ tato funkce potlačena. Stlačení klávesy BREAK má obdobnou funkci jako příkaz STOP. Pokračování chodu programu můžeme zadat systémovým příkazem CONT.

syntax:
<předznamenání klávesy BREAK>:=✻B|–|+|

Po zadání předznamenání ✻B+ je stisk klávesy považován za chybu (stav ERROR) a provede se příkaz TRAP≠<návěští>.

11. 6. 2 Tlačítko RESET

Stlačením tlačítka RESET se přeruší všechny prováděné funkce a počítač se nastaví do výchozího stavu TB. Pogram i hodnoty proměnných zůstanou zachovány.

12 CHYBÁŘ

kód	Vysvětlení
2	Malá paměť. V uživatelské oblasti paměti není místo pro další příkazy nebo proměnné.
3	Hodnota : av: je : 0, zatímco je očekáváno celé kladné číslo.
4	Byl překročen max. povolený počet : id: proměnných, které byly v programu deklarovány. (256)
5	Délka : $v: je větší než délka proměnné, do níž je hodnota přiřazována.
6	Po dosažení konce vnitřního souboru (pole DATA) se v programu vyskytl příkaz READ.
7	: čř: je větší než 32767
8	Do číselné proměnné je vkládána nečíselná hodnota.
9	Rozsah dimenzí : ip: je větší než 5460 nebo délka :$p: je větší než 32767.
10	Přetečení zásobníku, velký počet vnoření procedur.
11	Hodnota : av: je mimo číselný rozsah (dělení 0).
12	Příkazy GOTO, GOSUB THEN : čř: provádějí skok na neexistující programový řádek.
13	Nesprávné použití příkazů FOR/NEXT. Program narazil na příkaz NEXT, k němuž nenašel příkaz FOR.
14	Příliš dlouhý řádek (počet znaků na řádku: 120).
15	Vymazán řádek obsahující příkaz GOSUB nebo FOR.
16	Příkazu RETURN nepředcházel příkaz GOSUB.
17	Syntaktická chyba, příkazy nejsou zapsány povoleným způsobem.
18	Parametr :$v: ve funkci VAL obsahuje nečíselný znak.
19	Paměť nestačí k zavedení celého programu.
20	Hodnota : ids: je větší než 7 nebo je menší než 1.
21	Zaváděný program není vytvořen v TB.
23	Příkazu WEND nepředchází odpovídající WHILE.
24	Příkazu UNTIL nepředchází odpovídající REPEAT.
25	Příkazu LOOP nepředchází odpovídající DO.
26	Příkaz EXIT se nenachází uvnitř struktury.
27	Interpretor dospěl k příkazu PROC. Procedura není volána příkazem EXEC.
28	Příkazu ENDPROC nepředchází příkaz PROC nebo EXEC.
29	V příkazu EXEC byla volána neznámá procedura.
30	Návěští uvedené ve skokovém příkazu se v programu nenachází.
128	Nesprávné přerušení V/V operace (klávesa BREAK).

129 IOCB : ids: je již použito (otevřeno) pro jiný soubor. Dva soubory nemohou mít stejné : ids:.
130 Nedefinované V/V zařízení, neexistující : typ V/V:.
131 Příkaz vstupu použit pro soubor, který je otevřen pouze pro výstup informací.
132 Neplatný příkaz pro daný typ V/V zařízení.
133 Soubor : ids: nebyl otevřen (chybí OPEN: ids:..).
134 Neplatné číslo : ids:.
135 Příkaz výstupu byl použit pro soubor, který je otevřen pouze pro vstup informací.
136 Při V/V příkazech byl dosažen konec souboru (EOF).
137 Nedovolená délka záznamu, chybné čtení souboru.
138 V/V zařízení není zapnuto, neodpovídá.
139 Chybný přenos informací z V/V zařízení.
140 Ztráta informace při přenosu z V/V zařízení do počítače.
141 Zadané : sx:, : sy: jsou mimo povolený rozsah.
142 Chybný přenos dat na seriové sběrnici.
143 Chybný kontrolní součet při přenosu dat.
144 Disketa chráněna proti zápisu.
145 Neplatné (neexistující) číslo grafického módu.
146 Funkce nebyla vykonána.
147 Do uživatelské oblasti paměti se již nevejde VideoRAM zvoleného grafického módu.
160 Chybné číslo : ids: V/V zařízení.
161 Je otevřeno víc než 8 souborů.
162 Disketa je zaplněna, nelze zapsat další informace.
163 Chybná činnost V/V systému DOS.
164 Nesouhlasí číslo souboru na disketě.
165 Chybné jméno souboru.
166 Hodnota parametru : byt: v příkazu POINT je velká.
167 Soubor je chráněn proti přepisu.
168 Neplatný příkaz.
169 Adresář diskety je plný.
170 Soubor se na disketě nenachází.
171 Příkazu POINT nepředcházel příkaz OPEN pro aktualizaci souboru.
172 Soubor nebyl vytvořen v daném systému DOS.
173 Vadné sektory na disketě.

Přehled použité literatury

P. Nowák, Uživatelský popis programu Turbo Basic (dle HAPPY COMPUTER)
O. Burger a kol., Basic počítačů Atari 600XL/800XL
P. Dočękal, Adresy počítačů Atari 600XL/800XL
L. Molnár, Programovanie v jazyku PASCAL
J. Vogel, Programovanie v jazyku FORTRAN
Atari BASIC, uživatelská příručka
Atari Turbo Basic XL, příručka

Název : TURBO BASIC
Zpracoval : Ing. Miloslav Vaněk
Vydavatel : ZO Výpočetní technika
Svazarm Hodonín

Odpovědný redaktor : Ing. Josef Januška
Počet stran : 86
Tisk : Tiskárna Sigma Hodonín
Náklad : 1500 výtisků
Vydání :1., Hodonín 1989
JKPOV : 735 342 411
o 370042889

Neprošlo jazykovou úpravou

Igi/2024