





ZPRAVODAJ

ATARI  
klub Praha

příloha XIII

Ing. Vlastimil Bílek

TURBO BASIC

XL 1.5

uživatelská příručka

Vydává 487. ZO Svazarmu -  
ATARI KLUB v Praze 4.  
Šéfredaktor a vedoucí redakční rady  
JUDr. Jan Hlaváček.  
Zástupce šéfredaktora  
ing. Stanislav Borský.  
Technická redakce O. Strnadová.  
Jazyková úprava S. Bürgerová.  
Adresa redakce:

487. ZO Svazarmu - Atari klub Praha  
REDAKCE

poštovní přihrádka 51  
100 00 P r a h a 10

Redakční rada: M. Bayer, ing. J. Biskup,  
RNDr. J. Bok, CSc., ing. S. Borský,  
ing. V. Friedrich, ing. O. Hanuš, RNDr.  
L. Hejna, CSc., ing. J. Chábera, ing. P.  
Jandík, Z. Lazar, prom. fyz., CSc., F.  
Tvrdek, P. Vacek.

Otisk povolen se souhlasem redakce  
při zachování autorských práv a s  
uvedením pramene. Rukopisy nevyžá-  
dané redakcí se nevracejí. Za pů-  
vodnost a věcnou správnost ručí  
autor.

Vychází šestkrát ročně. Neprodejné.  
Členům klubu distribuováno zdarma.  
Nepravidelné přílohy na objednávku  
jsou kompenzovány zvláštním klubo-  
vým příspěvkem.

Rozsah přílohy 93 stran.

Do tisku předáno v VIII/1989.

Vytiskla tiskárna USI Praha.

Vydávání schváleno OV Svazarmu Praha  
4, OSK ONV Praha 4.

Evidenční číslo UVTEI 86 042.

© ATARI KLUB Praha, 1989.

Autor programu  
Frank Ostrowski

Vydal Happy Computer, 1985

*TURBO BASIC XL 1.5*

© Ing. Vlastimil Bílek, 1989

Recenze RNDr. Jiří Bok, CSc.,

Zdeněk Lazar, prom.fyz., CSc.,

Ing. Jiří Chábera,

Michal Mareš

© Publication ATARI KLUB Praha, 1989

## 1. Ú V O D

Počítače ATARI řady XL a XE jsou standardně vybaveny interpretem programovacího jazyka Atari BASIC (dále jen AB). Tento jazyk byl určen pro počítače řady TM(400/800). Nové řady počítačů XL a XE s sebou přinesly i nové možnosti, které AB nevyužívá. Kromě toho ve srovnání se současnými mikropočítači a jejich programovými prostředky má AB značně omezené možnosti a je i pomalý. Proto bylo vytvořeno několik verzí Basicu "šitých na míru" malým počítačům ATARI (BASIC X+, XL a XE). Jedním z nich je i Turbo-BASIC XL 1.5 (autor F. Ostrowski). Úpis interpretu i překladače byl uveřejněn v časopise Happy Computer "Sonderheft" 1985.

Programovací jazyk Turbo-BASIC (dále jen TB) se z hlediska uživatele jeví jako rozšíření stávajícího jazyka AB. Prakticky všechny programy napsané v AB jsou použitelné i pro interpret TB. Újmkou tvoří programy využívající uživatelsky definovaný DL, PMG, strojové rutiny, u kterých je třeba provést přesun v paměti nebo přeadresaci, apod. Programy napsané v AB a spuštěné v TB jsou v průměru 2-5x rychlejší, než v AB, programy napsané s použitím instrukcí TB jsou ještě asi 2-4x rychlejší. Svou rychlostí se může TB srovnávat se standardním interpretem Basicu pro některé 16-bitové počítače.

Důsledným používáním nových příkazů a možností TB je možno zpracování programu ještě více urychlit, zkrátit a zpřehlednit.

Interpret TB existuje ve verzi pro disketovou jednotku i kazetový magnetofon, obě verze jsou v podstatě shodné. Existuje již také kazetová verze TB v systému TURBO 2000 a v SUPER TURBO systému, včetně překladače.

Následující výklad předpokládá dobrou znalost AB, není proto vhodný pro začínající programátory. Výklad neobsahuje ty příkazy, které přebírá AB z TB zcela beze změn.

Pokud nebude uvedeno jinak, předpokládá se spolupráce s disketovou jednotkou ATARI 1050 pod DOSem 2.5.

## 2. Z A V E D E N Í D O P O Č Í T A Č E

Načítání TB z diskety se provádí z DOSu příkazem L (BINARY LOAD). Je také možno uložit TB na disketu s názvem "AUTORUN.SYS".

V tom případě je TB spouštěn automaticky z DOSu po zapnutí počítače, případně po studeném startu.

Z kazety se TB zavede do počítače zapnutím počítače při současném stisknutí kláves START a OPTION (ozve se zvukový signál). Potom je nutno nastavit kazetu na začátek programu, na magnetofonu stisknout klávesu PLAY a na klávesnici počítače libovolnou funkční klávesu (mimo BREAK). Dále probíhá zavádění do počítače stejně jako u disketové verze, tj. objeví se hlavička TB a po načtení programu nápis READY.

V systému TURBO 2000 použijeme TURBO loaderu nebo operačního systému TOS XL 2.4 (TOS 4.1).

### 3. V Ý K L A D T B

#### 3.1 Zkratky, značky a symboly použité v textu

v	= aritmetický nebo relační výraz (např. A, A+B*C, A=B, A>B, A+PEEK(B*C), ...)
čř	= číslo řádku
n	= jméno proměnné
jméno	= jméno návěští nebo procedury
a, b, c, d, p, ...	= konstanty, argumenty, parametry, ...
....	= posloupnost příkazů nebo řádků, které nejsou pro daný výklad podstatné
ř	= řetězec
<=>	= odpovídá v AB
<==>	= je ekvivalentní, totožné
DL	= display list
PMG	= Player - Missile - graphics (hráč - střela - grafika)
TB	= Turbo-BASIC
AB	= Atari-BASIC
CTB	= překladač TB
OS	= operační systém
DOS	= diskový operační systém

#### 3.2 Editace programu

Při editaci programu platí stejné zásady, jako v AB. TB však oproti AB dovoluje po přerušení programu (ať už chybou nebo programově (STOP, END) nebo klávesou /BREAK/) a následně editaci pokračovat v jeho vykonávání příkazem

CONT, a to i pokud byl program přerušen v těle cyklu, procedury, uprostřed řádku apod. Přitom se hodnoty proměnných ani zásobníků nesmažou. Nesmíme ovšem editovat (měnit) řádky, na nichž je definován začátek nebo konec cyklu, jenž v okamžiku přerušeni právě probíhal. V takovém případě může být další vykonávání programu chybné.

Dalším přínosem TB je optické rozlišení struktur. TB jejich těla odsazuje o dvě mezery doprava. Úpis programu je tak přehlednější a často pomůže odhalit chybějící příkazy cyklu apod. Pokud je v programu chyba nebo bylo použito více příkazů NEXT, ENDPROC, ENDIF apod. na jednom řádku nebo nebyly použity na začátku řádku (jako 1. příkaz na řádku), případně není program napsán strukturovaně, jeví se úpis jako neuspořádaný.

TB automaticky převádí malá a inverzní písmena na velká, rozepisuje zkratky, doplňuje nepovinné mezery a uvozovky na konci řádku, kromě řetězců následujících za příkazy REM a DATA a řetězcových proměnných.

Popis příkazů pro editaci programu je v odstavci 3.4.7.

### 3.3 Jména proměnných, návěští a procedur v TB

Proměnné v TB slouží k uchovávání číselných hodnot a znakových řetězců (jako v AB).

Číselné proměnné mohou být jednorozměrné (skaláry), jednorozměrné nebo dvojrozměrné (vektory) a znakové řetězce jsou jednorozměrné (vektory). K označení skalárů a vektorů slouží jména proměnných, vektory je nutné deklarovat příkazy DIM nebo COM. Až potud je TB stejný jako AB.

V TB lze v programu definovat tzv. jména návěští, která slouží ke skoku na definované místo v programu nebo k nastavení ukazovátka dat.

Jmen proměnných + návěští + procedur může být v TB 256 (na rozdíl od AB, kde je tabulka jmen omezena na 128. Jména zaznamenaná v tabulce od 129. místa výše však spotřebují při definování 2 byty (místo 1 bytu). Při každém volání zabírá jméno v programu 1 byte.

Délka jména může být 119 znaků, interpret rozlišuje všech 119 znaků. Délka jména je tedy omezena pouze délkou řádku v Basicu. Jméno proměnné MAXIMUM je v AB odlišováno od jména proměnné MATICE na rozdíl například od Basicu IQ-151 a Basicu pro ZX Spectrum, které rozlišují pouze první dva znaky, a pro něž je proto jméno MAXIMUM totožné se jménem MATICE.

### 3.4 Příkazy TB

#### 3.4.1 Strukturované programování - úvod

TB se svojí stavbou přibližuje vyšším programovacím jazykům. Umožňuje například používání strukturovaných příkazů známých z jazyka Pascal.

Struktury jsou konstrukce, které jsou tvořeny předpisy pro postupné, podmíněné, či opakované provádění dílčích příkazů. Struktury můžeme vytvářet do několika úrovní, tzn. že dílčími příkazy strukturovaných příkazů mohou být opět struktury. Dodržovat zásady strukturovaného programování se vyplatí především tehdy, chcete-li svůj program po odladění přeložit pomocí TB překladače.

Mezi strukturované příkazy patří v TB příkazy skoků, příkazy cyklů a procedur.

#### 3.4.2 Příkazy větvení (podmíněného skoku)

##### IF/ELSE/ENDIF

IF je příkaz větvení programu na základě splnění určité podmínky. Varianty tohoto příkazu v TB jsou:

```
IF v THEN ČR (<=>AB)
IF v THEN ... (<=>AB)
IF v : ... :ELSE : ... :ENDIF
IF v : ... :ENDIF
```

Při zpracování příkazu se vypočte hodnota výrazu v. Pokud nabývá nenulové hodnoty ("pravda - true"), provede se posloupnost příkazů uvedená za THEN, resp. mezi IF v : a :ELSE, není-li použit ELSE, pak mezi IF v : a :ENDIF.

Je-li hodnota výrazu v nulová ("nepravda - false"), provede se posloupnost příkazů mezi ELSE : a :ENDIF. Nebylo-li použito ELSE, podmínka se přeskočí a pokračuje se na dalším řádku (v případě IF v THEN ...), nebo následujícím příkazem za ENDIF.

Mezi příkazy IF/ELSE/ENDIF může být libovolné množství příkazů případně řádků.

Vše lépe uvidíte na následujícím příkladě. Posloupnost příkazů, která se provede v případě splnění podmínky, označíme "P", "N" pak v případě nesplnění. Příkaz, který následuje po podmínce označíme jako "D".

Př. 1:

- a) Č# IF v THEN P  
    Č# N       (N je zde totožné s D)
- b) Č# IF v :P:ELSE :N:ENDIF :D
- c) Č# IF v :P:ENDIF :N  
    (N je zde totožné s D)

Příkaz podmínky ve tvaru bez THEN musí být ukončen příkazem ENDIF. Jednotlivá klíčová slova podmínky musí být od ostatních příkazů oddělena dvojtečkou nebo napsána na dalším řádku. Příkaz IF může obsahovat další příkaz IF. Z těla podmínky (příkazy mezi IF v : a :ENDIF) lze libovolně vyskočit.

Poznámka autora: Zcela výjimečně může dojít k chybě č.22 NEST tam, kde je struktura IF/ELSE/ENDIF, ale strukturalizace programu je v pořádku. Pravděpodobně jde o chybu v TB, kterou jsem v interpretu doposud nenašel, a kterou lze odstranit použitím dvou příkazů ENDIF následujících za sebou.

Př. 2:

```
10 INPUT "Zadej A";A
20 IF A THEN ? "v=pravda":GOTO 40
30 ? "v=nepravda"
40 ? "A=";A
100 GOTO 10
```

Př. 3:

```
10 INPUT "Zadej A";A
20 IF A>=A^2-A: ? "v=pravda":ELSE :
   ? "v=nepravda":ENDIF : ? "A=";A:
   ? "A^2-A=";A^2-A: ? "v=";A>=A^2-A
100 GOTO 10
```

Př. 4:

Totéž jako Př.3, jen přehledněji zapsáno (výhodné zejména při ladění složitějších programů):

```
10 INPUT "Zadej A";A
20 IF A>=A^2-A
30 ? "v=pravda"
40 ELSE
50 ? "v=nepravda"
60 ENDIF
70 ? "A=";A: ? "A^2-A=";A^2-A: ? "v=";
   A>=A^2-A
100 GOTO 10
```

Př. 5:

```
10 INPUT "Zadej A";A
20 IF A<>2
30 ? "A=2"
40 ENDIF
50 GOTO 10
```

Př. 6:

Rádky 20, 30 a 40 z př. 5 lze opět zapsat dohromady jako:  
20 IF A<>2: ? "A=2":ENDIF

Př. 7:

Následující program je určen pro výpočet kořenů kvadratické rovnice a je hezkou ukázkou použití podmíněných skoků:

```
5 REM Program pro vyocet korenu
   kvadraticke rovnice  $A*X^2+B*X+C=0$ 
10 INPUT "Zadej parametry rovnice A,B,C"
   ;A,B,C
20 IF A=0 AND B=0
30   ? "Degenerovana rovnice"
40 ELSE
50   IF A=0
60     ? "Jednoduchy koren";-C/B
70   ELSE
80     IF C=0
90       ? "Koreny jsou: ";-B/A;" ";0
100      ELSE
110        DISCR=B^2-4*A*C
120        IF DISCR>=0
130          DISCR=SQR(DISCR)
140          IF B>=0
150            X1=(-B-DISCR)/2/A
160          ELSE
170            X1=(-B+DISCR)/2/A
180          ENDIF
190          ? "Koreny jsou: ";X1;" ";X2
200        ELSE
210          X1=-B/2/A
220          X2=SQR(-DISCR)/2/A
230          ? "Koreny jsou komplexni:"
240          ? X1;" +i*";" ";X2
250          ? X1;" -i*";" ";X2
260        ENDIF
270      ENDIF
280    ENDIF
290  ENDIF
300 ENDIF
```

### 3.4.3 Procedury

Proceduru v TB je třeba chápat jinak, než v Pascalu. V TB hodnoty proměnných použitých uvnitř procedur jsou obecně platné pro celý program a naopak. Všechny proměnné mají na rozdíl od Pascalu a jiných obdobných programovacích jazyků globální charakter, proto je hodnota libovolné konkrétní proměnné v jednom podprogramu stejná, jako v kterémkoli jiném místě programu.

## PROC/ENDPROC

Příkazy definice procedury. Procedura začíná příkazem PROC jméno a končí příkazem ENDPROC, který je obdobou příkazu RETURN, tj. program pokračuje příkazem následujícím za příslušným příkazem EXEC.

U těla procedury je možno volat jiné procedury, a to i sebe samu (rekurze). Počet volání při rekurzi je omezen aktuální kapacitou volné paměti RAM.

Proceduru lze opustit použitím příkazu EXIT (viz dále).

## EXEC

Syntax příkazu:  
EXEC jméno

Tento příkaz volá proceduru daného jména. Je obdobou příkazu GOSUB čí, vykonávání procedury EXEC/PROC/ENDPROC je však rychlejší a obvykle vyžaduje i méně paměti (viz dále).

Proceduru lze volat pouze příkazem EXEC jméno, nikoliv tedy např. GOSUB čí či jinak.

Jméno procedury je zapsáno v tabulce proměnných (viz např. DUMP) a potřebuje 8 bytů + pro každý znak jména 1 byte. Každé volání procedury příkazem EXEC jméno zabírá 4 byty, na rozdíl od GOSUB čí, který spotřebuje pokaždé 10 bytů (GOSUB A zabírá také 4 byty).

Z toho tedy vyplývá, že paměťově se použití EXEC/PROC/ENDPROC místo GOSUB čí/RETURN při vícenásobném použití vyplatí, o časových úsporách nemluvě!

Je-li jméno procedury zapsáno v tabulce proměnných jako 129, a více, spotřebuje jméno procedury 2 byty a každé volání procedury o 1 byte více (viz odst. 3.3).

Př. 8:

```
10 REM Příklad nastavení grafického modu
20 EXEC G0:LIST:PAUSE 300:EXEC G1:
   ? #6;"GRAFIKA 1":PAUSE 300:EXEC G2:
   ? #6;"GRAFIKA 2":PAUSE 300
30 EXEC G17: ? #6;"GRAFIKA 17":PAUSE 300:
   EXEC G18: ? #6;"GRAFIKA 18":PAUSE 300
90 END
100 PROC G0:GRAPHICS 0:DPOKE 709,8:
   ENDPROC
110 PROC G1:GRAPHICS 1:EXEC C1:ENDPROC
120 PROC G2:GRAPHICS 2:EXEC C2:ENDPROC
130 PROC G17:GRAPHICS 17:EXEC C1:ENDPROC
```

```

140 PROC G18: GRAPHICS 18: EXEC C2: ENDPROC
150 PROC C1: MOVE ADR("><848"), 708, 5:
    ENDPROC
160 PROC C2: MOVE ADR("86420"), 708, 5:
    ENDPROC

```

### 3.4.4 Příkazy cyklu

Tyto příkazy předepisují opakované provádění zadaných příkazů (posloupností příkazů). Příkazy cyklu jsou:

```

REPEAT/UNTIL
WHILE/WEND
DO/LOOP
FOR/NEXT

```

#### REPEAT/UNTIL

Syntax tohoto příkazu je:  
REPEAT : .... :UNTIL v

Příkaz se provádí tak, že se provede daná posloupnost příkazů, a potom se vyhodnotí podmínka opakování (výraz v za UNTIL). Jde o smyčku s podmínkou na konci. Pokud je hodnota v "nepravda", celý cyklus mezi REPEAT : a :UNTIL se opakuje. Pokud je hodnota výrazu v "pravda", pokračuje program vykonáváním příkazu následujícího za UNTIL v (překlad REPEAT - UNTIL je opakuj, dokud není hodnota výrazu v "pravda"). Celý cyklus se tedy provede vždy alespoň 1x.

Př. 9:

```

Výpočet částečného součtu řady  $X^n/n!$ 
(n=1 až M), kde M je nejmenší n, pro
které platí  $ABS(X^n/n!) < EPS$ 
10 INPUT "X, EPS"; X, EPS
20 I=0: CLEN=1: SOUCET=1
30 REPEAT
40   I=I+1: CLEN=CLEN*X/I
50   SOUCET=SOUCET+CLEN
60 UNTIL ABS(CLEN) < EPS
70 ? "SOUCET="; SOUCET

```

Pokud chcete ušetřit paměť, můžete totéž zapsat do jednoho řádku.

Př. 10:

```

Řádky 30-60 z Př. 9 lze zapsat jako:
30 REPEAT : I=I+1: CLEN=CLEN*X/I:
SOUCET=SOUCET+CLEN: UNTIL ABS(CLEN) < EPS

```

Zkrátíte tím program (v tomto případě o 9 bytů, tj. na 1 ušetřený řádek 3 byty), ovšem na úkor přehlednosti. Zrychlení programu není příliš zřetelné (v tomto případě asi o 1.5%).

### WHILE/WEND

Syntax tohoto příkazu je:  
WHILE v: .... :WEND

U tohoto příkazu se nejprve testuje hodnota podmínky (smyčka s podmínkou na začátku). Je-li hodnota výrazu v "pravda", provede se posloupnost příkazů mezi WHILE v: a :WEND. Není-li podmínka splněna, tělo cyklu se přeskočí a pokračuje se vykonáváním dalšího příkazu za WEND. Může tedy nastat případ, kdy tělo smyčky nebude provedeno ani jednou (je-li hned na počátku hodnota výrazu v "nepravda").

Př. 11:

Výpočet největšího společného dělitele (nsd) dvou čísel:

Platí:

$x > y \dots \text{nsd}(x, y) = \text{nsd}(x - y, y)$

$x < y \dots \text{nsd}(x, y) = \text{nsd}(x, y - x)$

$x = y \dots \text{nsd}(x, y) = x = y$

5 REM Výpočet nsd kladných čísel A a B

10 INPUT "Zadej A, B"; A, B

20 X=A:Y=B

30 WHILE X<>Y

40 IF X>Y

50 X=X-Y

60 ELSE

70 Y=Y-X

80 ENDIF

90 WEND

100 ? "Nejv. s. d. čísel ";A;" a ";B;"  
je ";X

Pozn. recenzenta: Pro praktické účely je výhodnější použít tzv. Euklidův algoritmus (viz např. některé středoškolské učebnice matematiky).

### DO/LOOP

Syntax tohoto příkazu je:  
DO : .... :LOOP

DO/LOOP je smyčka s podmínkou umístitelnou kdekoli v těle cyklu, popřípadě bez podmínky,

čímž vzniká nekonečná smyčka. Cyklus DO/LOOP je možno resulerně opustit jen příkazem EXIT. Existuje ještě druhá možnost opuštění této smyčky pomocí příkazů POP :GOTO , což však neodpovídá zásadám strukturovaného programování.

Př. 12:

```
10 DO
20   A=A+1
30   IF A=100 THEN EXIT
40 LOOP
50 ? A
```

je ekvivalentní

```
10 FOR A=0 TO 99
20 NEXT A
30 ? A
```

Př. 13:

```
10 POKE 764,255
20 DO
30   IF PEEK(764)=33 THEN EXIT
40 LOOP
50 ? "Stisknul jsi /SPACE BAR/!"
```

Pozn.: Na adrese 764 je kód poslední stisknuté klávesy. 255 znamená, že nebyla stisknuta žádná klávesa, 33 je kód SPACE (mezera).

### FOR/NEXT/STEP

Syntax tohoto příkazu je:

```
FOR n=v0 TO v1 STEP v2:....:NEXT n
FOR n=v0 TO v1:....:NEXT n (Je-li krok
v2=1)
```

Tento příkaz přebírá TB z AB v podstatě beze změny. Jediným rozdílem je způsob vykonávání smyčky po příkazu \*F, podrobnosti jsou uvedeny u příkazů \*F+, \*F-.

### \*F (\*F+)

Po vykonání tohoto příkazu je u všech smyček FOR/NEXT nejprve testován parametr smyčky a porovná se s její konečnou hodnotou. Je-li parametr smyčky menší nebo roven konečné hodnotě, smyčka se provede, je-li větší, celý cyklus FOR/NEXT se přeskočí.

Př. 14:

```
10 FOR I=2 TO 1: ? I: NEXT I: REM Smyčka
   se provede
20 ? "%F+"
30 *F
40 FOR I=2 TO 1: ? I: NEXT I: REM Smyčka
   se neprovede
```

Příkaz \*F+ často nahradí příkazy s IF na přeskočení smyčky FOR/NEXT apod.

\*F-

Tento příkaz ruší příkaz \*F+ (\*F), tj. smyčky FOR/NEXT proběhnou nejméně 1x. Příkaz RUN neruší příkaz \*F+.

Srovnáme-li příkazy cyklů v TB, zjistíme zhruba toto:

Příkaz cyklu	test podmínky	typ podmínky	nejmenší počet průchodů
=====	=====	=====	=====
FOR/NEXT	konec cyklu	parametr cyklu	1
*F+/FOR/NEXT	začátek cyklu	parametr cyklu	0
REPEAT/UNTIL	konec cyklu	v	1
WHILE/WEND	začátek cyklu	v	0
DO/LOOP	kdekoliv uvnitř cyklu	v pomocí IF	0

Z této tabulky také snáze usoudíme, který typ cyklu při daných podmínkách lépe vyhovuje.

### 3.4.5. Příkazy opuštění struktur

EXIT

Tento příkaz je jediným příkazem přípustným ve strukturovaném programování, kterým je možno předčasně ukončit všechny uvedené typy

struktur. Příkaz EXIT opustí strukturu skokem na příkaz následující za jejím koncem.

Po EXIT může následovat výraz v, který na vykonání příkazu nemá vliv (tento fakt lze využít podobně jako REM). Viz DO/LOOP (př.12 a 13).

Strukturu lze obecně opustit také příkazem GOTO. V tomto případě však TB nezapomene návratovou adresu. Jak jsem již uvedl na jiném místě této publikace, je využití příkazu GOTO pro tyto účely velmi ošidné a neodpovídá zásadám čistého strukturovaného programování.

### 3.4.6 Symbolická návěští

Definice symbolických návěští umožňují skok na definované místo programu nebo nastavení ukazovátka dat.

`# jméno`

Syntax příkazu:  
čř # jméno

Návěští # jméno umožňuje skok na definované místo v programu. Definice jména návěští odpovídá definici jména procedury, tj. vyžaduje 8 + 1 byte na každý znak jména. Např. na jméno definované příkazem # NAHORU spotřebuje TB 8+6=14 bytů. Každé volání návěští, ať už příkazem GOTO, RESTORE#, či TRAP# vyžaduje 1 byte. Příkazy s "# jméno" jsou kromě toho rychlejší než s "čř" a při vícenásobném použití šetří paměť.

`GOTO# jméno`  
`TRAP# jméno`

`GOTO# jméno (TRAP# jméno)` odpovídá GOTO čř (TRAP čř), jen místo na čř se provede skok na místo, kde je definováno návěští příkazem # jméno.

`RESTORE# jméno`

Nastavuje ukazovátka dat pro příkaz READ na řádek označený návěštím # jméno. Odpovídá příkazu RESTORE čř.

Př. 15:

```
10 # ABC
20 A=A+1
30 IF A<1000 THEN GO# ABC
```

nebo totéž zkráceně:

```
10 # ABC:A=A+1:IF A<1000 THEN GO# ABC
```

Př. 16:

```
10 RESTORE #DATA2
1000 # DATA1
1010 DATA 1,2,3,4,5
1020 # DATA2
1030 DATA 6,7,8,9,0
1040 READ A,B,C,D,E: ? A,B,C,D,E
```

Př. 17:

```
10 # ZNOUU
20 TRAP #CHYBA
30 INPUT A,B
40 ? A/B
50 GO# ZNOUU
100 # CHYBA
110 ? "Neumim delit nulou!":GO# ZNOUU
```

### 3.4.7 Příkazy násobného větvení

ON/GO# ON/EXEC
-------------------

Syntax příkazů je:

```
ON v GO# jméno1, jméno2, jméno3, ...
ON v EXEC jméno1, jméno2, jméno3, ...
```

Definuje skok na návěští nebo volání procedury v závislosti na hodnotě výrazu v. Je obdobou příkazu ON v GOTO čí1, čí2, čí3, ... a ON v GOSUB čí1, čí2, čí3, .... Vykonávání je však rychlejší.

Pozn.: Stejně jako u ON/GOTO a u ON/GOSUB pokud je hodnota výrazu v menší než 1 nebo větší než počet následujících jmen, pokračuje program na dalším řádku a skok vynechá. Hodnota výrazu v je zaokrouhlována Turbo-Basicem podle vztahu  $v = \text{INT}(v + 0.5)$ . Pokud je hodnota výrazu v rovna 1, provádí se skok na řádek (volání procedury) uvedený jako první, při hodnotě výrazu v=2 na druhý, atd.

### 3.4.8 Příkazy pro editaci programu

\*L-

Tento příkaz vypíná tabelátor, tj. mezery, kterými jsou odsazeny struktury, jsou vynechávány, značka "---", použitá jako REM, se netiskne jako 15x "---", ale jen 1x. Příkaz se používá při editaci dlouhých řádků, k úspoře místa na disketách apod. Tento příkaz nelze zrušit ani příkazem RUN, ani použitím RESET. Pracuje se všemi zařízeními, která umožňují využívat příkazu LIST.

\*L (\*L+)

Zapíná zpět tabelátor, tj. ruší příkaz \*L-. Účinky tohoto příkazu neruší ani příkaz RUN.

--

Syntax příkazu:

čís --

Např. řádek

99 --

se při výpisu vytiskne jako

99 -----

tj. 15x "---". Při běhu programu se chová jako REM. Slouží k optickému dělení programu (oddělování bloků, procedur ap.).

Pozn.: Po příkazu \*L- se tiskne při výpisu programu znak "---" jen 1x.

LIST

Příkaz LIST je oproti AB, kde lze použít varianty

LIST

LIST čís

LIST čís1, čís2

rozšířen o variantu

LIST čís,

Tento příkaz provede výpis programu od udaného čísla do konce programu. Výpis lze samozřejmě zastavit klávesou /BREAK/ (nebo dočasně /CONTROL-1/).

**DEL**

Syntax příkazu:

DEL čí1, čí2 (od, do)

Příkaz k vymazání skupiny po sobě následujících řádků. Příkaz DEL 100,200 vymaže řádky č.100 (včetně) až 200 (včetně).

**RENUM**

Syntax příkazu:

RENUM A,B,C

A=od kterého čísla, B=nové čísla, C=krok

Přečíslování všech řádků programu počínaje řádkem A až do konce programu. Nová čísla řádků pak začínají číslem B a jdou za sebou s krokem C. Příkaz také mění čísla řádků po instrukcích GOTO, GOSUB, TRAP, RESTORE, LIST, DEL, ON/GOTO, ON/GOSUB. Pokud narazil TB při přečíslování na příkaz skoku na neznámý řádek, označí výraz za tímto příkazem znaménkem "-" (mínus). Při běhu programu se toto volání řádku ohlásí losicky jako chyba č. 3 (ERROR- 3 VALUE). Příkaz nedokáže přečíslovat čísla u příkazů, kde se vyskytují proměnné:

GOTO A

TRAP BRK

RESTORE A\*10+10000

apod.

Následuje-li těsně za příkazem nějaká číselná konstanta, je brána jako číslo, je s ní provedeno přečíslování, zbytek výrazu v za příkazem zůstane nezměněn.

Př. 10:

RENUM 0,1,1 →

10 GOTO 100

20 GOTO A

30 GOTO A+10

40 GOTO 10+A

50 GOTO 100+10\*A

60 GOTO 300

100 REM

1 GOTO 7

2 GOTO A

3 GOTO A+10

4 GOTO 1+A

5 GOTO 7+10\*A

6 GOTO -300

7 REM

Pokud zadáte RENUM tak, že dojde k očíslování některých řádků programu stejným číslem, program obvykle "chodí", ale při jeho editaci mohou nastat chyby. V takovém případě zadejte RENUM 0,0,1, čísla řádků se obvykle srovnají. Pokud by měl při přečíslování vzniknout řádek s číslem vyšším než 32767 (nejvyšší možné číslo řádku), ohlásí TB chybu č. 3 - VALUE.

## DUMP

Syntax příkazu je:

DUMP

DUMP "X:NAME.EXT" , kde X je libovolné zařízení

Tento příkaz slouží k výpisu tabulky jmen a tabulky hodnot proměnných na specifikované zařízení. Jména jsou vypisována v pořadí, v jakém jsou v tabulce jmen zaznamenána, tj. v jakém pořadí byla ukládána do paměti při psaní programu. V tabulce zůstávají zapsána i jména, která byla později z programu vymazána apod. Jak se jich lze zbavit je popsáno v kapitole Rady a nápady.

Výpis jmen a hodnot proměnných lze provést například na následující zařízení:

B:, C:, P:, E:, S:, při použití Turbo operačního systému i zařízení T: . Pokud není zařízení specifikováno, provede se výpis na obrazovku.

Jména a hodnoty proměnných se při výpisu ukládají jako řetězce, které lze zavést do paměti počítače pomocí příkazu INPUT.

Ve výpisu lze rozlišit tyto typy jmen:

A =100 proměnná A, jejíž hodnota je 100  
 B( 10,1 deklarované pole B(9) nebo B(9,0)  
 C( 21,11 deklarované pole C(20,10)  
 D( 0,0  
 L\$ 0,0 nedeklarované pole (ř) (objeví se,

např. je-li v programu příkaz ?D(3,5) (L\$(2)) a nebyla dosud programově provedena deklarace pole D (ř L\$), resp. po vymazání pole (ř) z programu)

E\$ 10,20 ř E\$ deklarovaný příkazem DIM E\$(20) , délka ř je 10 znaků (LEN(E\$)=10)

G\$ 0,10 ř G\$ deklarovaný na 10 bytů, prázdný (LEN(G\$)=0, tj. G\$="")

H PROC 100 procedura H zapsaná od řádku 100 (100 PROC H)

I # 120 značka (návěští, label) # I na řádku 120

J ? nedefinovaná procedura nebo návěští

Používá se nejčastěji při ladění programu, poskytuje cenné informace o deklaraci, délce, jménech a hodnotách proměnných.

#### TRACE (TRACE+)

Zapíná trasování (stopování), tj. na obrazovce se vypisují v hranatých závorkách čísla právě prováděného řádku. V grafickém módu s textovým okénkem se výpis provádí v textovém okénku, v grafice 0 na celou obrazovku, u grafických režimů bez textového okénka dojde k jejich zrušení (obdobně jako při použití příkazu PRINT v grafickém režimu 24). Tisk čísel řádků s PROC a #, vyvolaných příkazy EXEC a GO# a řádků, na kterých začínají cykly (při opakování cyklu), je vynecháván.

#### TRACE-

Příkaz TRACE- vypíná režim trasování. Mód TRACE se ruší také v případě hlášení chyby nebo po stisknutí klávesy /BREAK/. TRACE+ se neruší příkazem RUN.

### 3.4.9 Příkazy operující s pamětí

#### DPOKE

Syntax příkazu je:  
DPOKE adr,slovo

Zapíše ( ukládá ) na "adr" a "adr+1" 16-bitové "slovo", tj.:

```
POKE adr,slovo-INT(slovo/256)*256:POKE adr+1,  
INT(slovo/256)
```

čili

```
POKE adr,LO:POKE adr+1,HI  
<=> DPOKE adr,LO+HI*256  
LO = nižší byte, HI = vyšší byte 16-bitového  
čísla
```

```
0 <= adr < 65535.5
```

```
0 <= slovo < 65535.5
```

adr i slovo mohou být samozřejmě výrazy.

Tento příkaz je rychlejší, jednodušší a mnohdy názornější, než 2x POKE, šetří také paměť.

**MOVE**  
**-MOVE**

Syntax příkazů je:  
MOVE odkud,kam,kolik  
-MOVE odkud,kam,kolik

Skvělý příkaz TB, sloužící k přesunu bloků paměti. Odpovídá posloupnosti příkazů:  
FOR I=0 TO kolik-1:POKE kam+I,PEEK(odkud+I):  
NEXT I,

Je však daleko rychlejší. Přesun bloku se provádí od bytu:

MOVE → "odkud" po byte "odkud+kolik-1"  
-MOVE → "odkud+kolik-1" po byte "odkud",  
tj. přesouvá se odzadu ( ukládá také ).  
Používá se, pokud je "odkud+kolik" > "kam",  
tj. došlo-li by k přepsání dosud nepřesunuté části bloku.

Pomocí těchto příkazů lze rychle zaplnit velké bloky paměti stejnou hodnotou.

Př. 19:

```
10 POKE DPEEK(88),10:MOVE DPEEK(88),  
DPEEK(88)+1,959
```

Hodnota 959 proto, jelikož obrazová paměť v grafice 0 zabírá 960 bytů, 0.-tý byte jsme zaplnili prvním příkazem (POKE). Na adrese dané hodnotou DPEEK(88) je začátek obrazové paměti (video RAM).

Př. 20:

```
10 DIM A$(960),B$(960):A$=CHR$(0):  
A$(960)=A$:A$(2)=A$:B$=A$  
20 GRAPHICS 0:LIST  
30 MOVE DPEEK(88),ADR(A$),960  
40 CLS:?"B$":MOVE DPEEK(88),ADR(B$),  
960  
50 DO  
60 MOVE ADR(A$),DPEEK(88),960:  
PAUSE 50  
70 MOVE ADR(B$),DPEEK(88),960:  
PAUSE 50  
80 LOOP
```

Př. 21:

```
10 Q=PEEK(106):Q=Q-4  
20 POKE 106,Q-1:R=Q*256:GRAPHICS 0  
30 MOVE $E000,R,1024  
40 POKE 756,Q  
50 ? R  
60 PAUSE 200:POKE R,1  
70 PAUSE 200:POKE R+7,255  
80 PAUSE 200:POKE R,0:POKE R+7,0
```

Tento program přesune znakovou sadu z ROM do RAM (do ochráněné paměti nad RAMTOP), kde ji je možno libovolně měnit. Znakovou sadu přesouvá řádek č. 30. Na adrese 756 je horní byte ukazatele začátku dat znakové sady. Znaková sada musí začínat na adrese, která je násobkem celého kilobytu.

Ochranu paměti proti možnosti přepsání Turbo-BASICem zajišťují řádky č. 10 a 20. Na adrese 106 je horní byte ukazovátka RAMTOPu, paměť nad touto adresou nemůže být přepsána řádky programu.

Proč je ochráněno 5 stránek paměti (5\*256 bytů) a ne jen 1024 bytů je vysvětleno u příkazu CLS.

### 3.4.10 Příkazy pro grafiku

#### CIRCLE

Syntax příkazu:  
CIRCLE x,y,r  
CIRCLE x,y,rx,ry

Tyto příkazy kreslí kružnici (elipsu) se středem v bodě x,y a poloměrem r (poloosou rx ve vodorovném směru a ry ve svislém směru). Poloměr (poloosy) r (rx, ry) musí být z intervalu (0;255.5), x a y z intervalu (0;65535.5). Střed kružnice (elipsy) nemusí ležet v rozsahu dané grafické obrazovky (na rozdíl od příkazu PLOT). Nevadí, pokud je r natolik velké, že by kružnice "lezla z obrazovky". V tomto případě se přečnivající část kružnice (elipsy) prostě nenakreslí. Při velkých hodnotách dochází k deformacím obrazců.

Př. 22:

```
10 GRAPHICS 24:COLOR 1:  
   CIRCLE 150,100,50:CIRCLE 290,90,250  
20 PAUSE 300  
30 GRAPHICS 24:COLOR 1  
40 FOR X=0 TO 65535 STEP 25  
50 CIRCLE X,65535-X,150  
60 NEXT X  
70 SOUND 0,20,10,15:PAUSE 50:SOUND  
80 GOTO 80
```

#### CLS CLS #6

CLS je příkaz k vymazání obrazovky, v AB MU odpovídá příkaz ?CHR\$(125) případně ? "¶".

CLS maže textové okénko (tj. v srafice 0 celý display). CLS #6 maže obrazovku v grafických a textových režimech (mimo textové okénko).

Příkazy CLS, ? "K" a GRAPHICS nulují paměť od adresy, na kterou ukazuje ukazovátka operačního systému SAUMSC (DPEEK(88)) až po RAMTOP (PEEK(186)\*256), avšak s tím, že paměť maže po celých stránkách (256 bytů). Začíná-li tedy obrazová paměť např. v srafice 0 na celé stránce, maže se 960 bytů (tolik má obrazová paměť v této srafice) + 64 bytů nad RAMTOPem (960+64=1024=4\*256). Proto může dojít ke smazání dat uložených v paměti ochráněné nad RAMTOPem (viz MOVE).

## PAINT

Syntax příkazu:

PAINT x,y

Vybarví uzavřený (ohraničený) obrazec, který má barvu stejnou, jaká je v bodě x,y, barvou předem definovanou příkazem COLOR. Obrazcem může být vnitřek kruhu, různé víceúhelníky, čáry, kružnice nebo obrazce vzniklé jejich překrýváním. Podstatné je, aby nebyla porušena kontinuita (spojitost) barvy v obrazci. Na rozhraní dvou barev se vybarvování zastaví.

Př. 23:

```
10 GRAPHICS 23:COLOR 1
20 CIRCLE 50,40,35:
   CIRCLE 100,50,40,25:
   CIRCLE 100,50,25,40
30 COLOR 2
40 PLOT 2,2:DRAWTO 100,2:
   DRAWTO 100,10:DRAWTO 2,10:
   DRAWTO 2,2
50 CIRCLE 20,10,50:DRAWTO 0,80
60 TEXT 80,85,"PAINT"
70 PAUSE 250
80 COLOR 3
90 PAINT 50,40
100 COLOR 2:PAUSE 50
110 PAINT 3,3:PAINT 100,50
120 COLOR 1:PAUSE 50
130 PAINT 81,86:PAINT 2,2
140 SOUND 0,10,10,15:PAUSE 20:SOUND
150 DO :LOOP
```

Vybarvování probíhá od udaných souřadnic x,y (počítají se v pixelech = obrazových bodech, tak jako u PLOT, TEXT, ...) nejprve směrem dolů a pak nahoru. Pozici kurzoru příkaz PAINT neovlivňuje.

Př. 24:

```
10 GRAPHICS 23
20 COLOR 1:CIRCLE 100,50,40
30 PLOT 0,0:DRAWTO 50,10
40 COLOR 2:PAINT 100,50
50 DRAWTO 20,70
60 DO :LOOP
```

Obrazec jednou již vybarvený nelze přebarvit jinou barvou než barvou číslo 0 (pozadí). Pak teprve lze zadat barvu jinou (příkazem COLOR) a obrazec přebarvit.

Př. 25:

```
10 GRAPHICS 7:COLOR 3:CIRCLE 100,50,20
20 COLOR 1:EXEC PAI
30 COLOR 2:EXEC PAI
40 COLOR 0:EXEC PAI
50 COLOR 3:CIRCLE 100,50,20
60 COLOR 2:EXEC PAI
90 DO :LOOP
100 PROC PAI
110 PAINT 100,50:PUT 253:PAUSE 40
120 ENDPROC
```

Ušimněte si, že PAINT po COLOR 0 přebarví celý obrazec bez ohledu na to, že je různobarvný. Zastaví se pouze na rozhraní s barvou 0 (pozadí).

Vybarvení určité plochy příkazem PAINT je náročné na paměť počítače, velikost a složitost plochy vybarvovaného obrazce je tudíž omezena velikostí volné paměti. Pokud dojde k přeplnění paměti, ohlásí TB chybu č. 2 (ERROR -2 MEM) (nedostatek paměti).

### FCOLOR

Syntax příkazu je stejný jako u příkazu COLOR:

```
FCOLOR v
(<=>) POKE 765,v
kde 0<=v<255.5
```

Používá se k volbě barvy (barvového registru) pro příkaz FILLTO, obdobně jako COLOR pro příkaz DRAWTO. Příklad viz FILLTO.

### FILLTO

Syntax příkazu:

```
FILLTO x,y
<=> POSITION x,y:XIO 10,#6,0,0,"S:"
```

Tento příkaz vybarvuje část obrazovky od úsečky dané příkazy PLOT x1,y1:DRAWTO x2,y2 napravo dokud nenarazí na čáru jiné barvy.

Př. 26:

```
10 GRAPHICS 31
20 FCOLOR 2
30 COLOR 3
40 PLOT 10,10:DRAWTO 150,150
50 PLOT 10,10:FILLTO 150,150
90 DO :LOOP
```

Př. 27:

```
Nyní změňte řádek 50 na:
50 PLOT 10,10:FILLTO 10,150
```

Grafika č. 31 (15) je režim dosažitelný u ATARI XL,XE, na rozdíl od typů 400/800, přímo z BASICu příkazem GRAPHICS 31 (GRAPHICS 15). Je to čtyřbarevná grafika s rozměrem obrazového elementu 2x1 obrazový bod v grafice 8 (tj. 2x širší a na výšku stejný, jako rozměr bodu v grafice 8 (24)).

### 3.4.11 Příkazy zvuku

DSOUND SOUND
-----------------

Oba příkazy vypínají všechny čtyři zvukové kanály, jak ukazuje následující příklad. Pracují naprosto stejně.

```
SOUND <=> FOR I=0 TO 3: SOUND I,0,0,0:NEXT I
DSOUND <=> FOR I=0 TO 3: SOUND I,0,0,0:NEXT I
```

Př. 28:

```
10 FOR I=0 TO 3: SOUND I,I*70,10,10:
   PAUSE 100:NEXT I: SOUND
20 FOR I=0 TO 3: SOUND I,I*50,10,10:
   PAUSE 100:NEXT I: DSOUND
```

DSOUND A,B,C,D
----------------

A = číslo kanálu, B = kmitočet,  
C = zkreslení, D = hlasitost

Pracuje podobně jako SOUND, ale s kanály 1 a 2, případně 3 a 4, spojenými v jeden šestnáctibitový zvukový registr, což umožňuje vytvářet zvuky ve velmi jemném odstupňování kmitočtu. Ten lze pro normu PAL vypočítat ze vztahu:

$$f=1789790/(2*(B+7)).$$

Pro normální SOUND platí:

$$f=63921/(2*(B+1)).$$

Parametry příkazu DSOUND mohou být:

A = 0 nebo 1

B, C, D = z intervalu <0;65535.5)

A, B, C, D mohou být výrazy, jako argument bere TB hodnotu INT(v+0.5).

Pokud použijete 1. a 2. zvukový kanál, můžete použít ještě příkazy SOUND 2, B, C, D a SOUND 3, B, C, D (SOUND 0 a SOUND 1 už ne). Obdobně při použití DSOUND 1, B, C, D lze ještě používat SOUND 0 a SOUND 1, ale SOUND 2 a SOUND 3 ne. Použijete-li DSOUND 0 i DSOUND 1, jsou použity všechny 4 kanály a příkaz SOUND již nelze používat, aniž by nepůsobil na již pracující zvukové kanály! Vzájemné ovlivňování příkazů DSOUND a SOUND je značně komplikované, objasnění tohoto problému přesahuje rámec této publikace.

#### 3.4.12 Příkazy I/O operací

**PUT #k**

Syntax příkazu:

PUT #k, n1, n2, ...

Odlišnosti ve vykonávání tohoto příkazu v TB oproti AB jsou ukázány níže u příkazu GET #k.

**GET #k**

Syntax příkazu:

GET #k, n1, n2, ...

k = číslo kanálu

ni = proměnná, jejíž hodnota je v mezích  
0 <= ni < 255.5

Po příkazech PUT # a GET # může v TB následovat seznam proměnných (v AB smí být pouze jedna). Po GET, stejně jako v AB, nemůže následovat indexovaná proměnná).

**PUT n**

Syntax příkazu:

PUT n1, n2, ...

PUT n (<=>) ? CHR\$(n) nebo ? #0; CHR\$(n)  
i v grafických režimech.

Př. 29:

PUT 125 (je stejné jako CL5)

**GET n**

Syntax příkazu:

GET n1, n2, ...

n1 zde nesmí být indexovaná proměnná

GET n(<=>) OPEN #7, 4, 0, "K:": GET #7, n: CLOSE #7

Příkaz GET n čeká na stisknutí klávesy a přiřadí proměnné n hodnotu ATASCII stisknuté klávesy. Na klávesu /BREAK/ a na klávesy na konzole mimo /HELP/ příkaz nereaguje.

**%PUT**

Syntax příkazu:

%PUT #k;n

Příkaz %PUT je vysvětlen u příkazu %GET (viz níže).

**%GET**

Syntax příkazu:

%GET #k;n

n zde nesmí být indexovaná proměnná

Příkazy %PUT a %GET slouží k ukládání a čtení dat, obdobně jako příkazy ? (PRINT) a INPUT, PUT a GET, BPUT a BGET.

Příkazy %PUT a %GET ukládají (čtou) čísla ve formě 6-bytových konstant. V této formě jsou v TB uloženy všechny hodnoty číselných proměnných mimo konstant %0, %1, %2, %3. %PUT a %GET fungují podobně jako ? (PRINT) a INPUT, ty však ukládají (čtou) čísla ve stejné formě, v jaké se píše při tisku na obrazovku. Číslo 567.8901234 je při PRINT, jak na obrazovku, tak na disketu či kazetu, ukládáno jako znaky 567.8901234R (R=RETURN), tj. zabírá 12 znaků = 12 bytů. Čísla záporná,

Případně čísla v exponenciálním tvaru aEb (-0.123456789 E-12) mohou zabírat ještě více paměti (16 bytů), ale také nemusí (-1 zabírá 2 byty). Čísla zapsaná příkazem %PUT však zaujímají vždy 6 bytů, neboť se zapisují ve formě, v jaké jsou uložena v paměti.

Proto při zápisu racionálních čísel bývá výhodnější používat příkazu %PUT. Data je pak samozřejmě nutno číst ekvivalentním příkazem, tj. %GET ! Tento způsob ušetří v průměru 1/2 paměti a času oproti příkazům ? (PRINT) a INPUT.

Ukládání a čtení dat se provádí předem otevřeným kanálem k.

Následující příklad porovnává oba zmíněné způsoby zápisu a čtení čísel:

Př. 30:

```
10 DIM A(1000)
20 FOR I=1 TO 1000:A(I)=RND:NEXT I
30 ? "PRINT POLE A(1000)"
40 T=TIME
50 OPEN #1,8,0,"D:DEMO"
60 FOR I=1 TO 1000: ? #1;A(I):NEXT I
70 CLOSE :T=TIME-T
80 ? " PRINT: ";T/50;" sec."
90 T=TIME
100 OPEN #1,4,0,"D:DEMO"
110 FOR I=1 TO 1000:INPUT #1;A:A(I)=A:
    NEXT I
120 CLOSE :T=TIME-T
130 ? " INPUT: ";T/50;" sec."
140 DIR "D:DEMO"
150 DELETE "D:DEMO"
160 T=TIME
170 OPEN #1,8,0,"D:DEMO"
180 FOR I=1 TO 1000:%PUT #1;A(I):NEXT I
190 CLOSE :T=TIME-T
200 ? "%PUT: ";T/50;" sec."
210 T=TIME
220 OPEN #1,4,0,"D:DEMO"
230 FOR I=1 TO 1000:%GET #1;A:A(I)=A:
    NEXT I
240 CLOSE :T=TIME-T
250 ? "%GET: ";T/50;" sec."
260 DIR "D:DEMO"
270 DELETE "D:DEMO"
```

Tabulka získaných hodnot:

t = čas s vypnutou verifikací (v sek.)  
t(ver.) = čas se zapnutou verifikací (v sek.)  
paměť = počet sektorů, které zaujímá datový soubor

Verifikace (kontrola zápisu) se zapíná příkazem POKE 1913,67, vypíná POKE 1913,80.

	t	t(ver.)	Paměť
PRINT	37.18	60.92	104
INPUT	34.18	34.14	
%PUT	20.86	32.10	48
%GET	16.30	16.32	

Pozn.: %PUT n1,n2,... provede výstup obdobně jako PUT n kanálem č. 6, tj. obvykle na obrazovku, a to ve formě BCD konstant.

### BPUT

Syntax příkazu:  
BPUT #k,adr,len

Příkaz zápisu bloku dat z paměti kanálem číslo k.

BPUT #k,adr,len (<=>) FOR I=0 TO len-1:PUT #k, PEEK(adr+I):NEXT I.

Popis příkazu u BGET.

### BGET

Syntax příkazu:  
BGET #k,adr,len

Příkaz čtení bloku obdobný BPUT.

BGET #k,adr,len (<=>) FOR I=0 TO len-1:GET #k, A:POKE adr+I,A:NEXT I

k = číslo kanálu

adr = adresa, od které začíná zápis (čtení) bloku

len = délka bloku v bytech

Před zadáním těchto příkazů musí být otevřen příslušný kanál (pro zápis (8) u BPUT, pro čtení (4) u BGET, případně pro zápis i čtení (12) pro oba příkazy).

Tyto příkazy lze s výhodou použít například pro uchovávání obsahu obrazové paměti, ať už na disketu nebo na kazetu.

Př. 31:

```
10 OPEN #1, 8, 0, "D: GRAFIKA0. PIC":
   BPUT #1, DPEEK(88), 960: CLOSE #1
20 CLS
30 OPEN #1, 4, 0, "D: GRAFIKA0. PIC":
   BGET #1, DPEEK(88), 960: CLOSE #1
```

Pro uložení na kazetu musíte nahradit jméno souboru "D:GRAFKA0.PIC" specifikací "C:" a v parametrech příkazu OPEN nahradit nulu číslem 128 (souvisí s délkou mezer mezi jednotlivými záznamy datového souboru!) Také chcete-li uchovávat obrázky v jiné grafice, je třeba místo 960 zapsat příslušný počet bytů dané obrazovky.

Parametry k, adr a len mohou být výrazy, adr a len mohou být z intervalu (0;65535.5).

## CLOSE

Zavírá kanály 1-7.

```
CLOSE <=> FOR I=1 TO 7:CLOSE #I:NEXT I
```

Lze použít např. tehdy, startujeme-li program příkazem GOTO (příkaz RUN sám uzavírá kanály 1-7), nemáme-li otevřeno více kanálů (šetří paměť) nebo chceme-li zavřít všechny kanály (samozřejmě mimo kanál číslo 8).

Pozor! CLOSE uzavírá také kanál č.6, který potřebují grafické režimy. Proto zadáte-li po příkazu třeba GRAPHICS 8 příkaz CLOSE, nebudete moci kreslit na obrazovku! Jediným východiskem je zadat GRAPHICS 8+32 (nemaže obrazovou paměť).

## INPUT

Syntax příkazu:

INPUT A,B,C\$ je známé z AB

INPUT #3;A je známé z AB

INPUT #3,A je známé z AB

INPUT "text";A,B,C\$ jen TB

INPUT "text",A,B,C\$ jen TB

Příkaz INPUT slouží v TB stejně jako v AB ke čtení hodnot proměnných, ať už z obrazovky či jiného zařízení. Avšak na rozdíl od AB umožňuje vložit před zadání text (musí být napsán v uvozovkách), který se bude tisknout.

Oddělování příkazu INPUT od seznamu proměnných čárkou nebo středníkem řídí tisk otazníku před vlastním vstupem dat. Při použití varianty se středníkem se tiskne otazník, varianta s čárkou otazník netiskne. Pokud použijete jako "text" prázdný znak, tj. "", vytiskne se znak "\*" (= control - K). Otazník se také netiskne v případě, že zadáváme data pomocí INPUT #k... .

Př. 32:

```
10 OPEN #3,12,0,"E":LIST
20 INPUT A
30 INPUT #3;A
40 INPUT #3,A
50 INPUT #0;A
60 INPUT #0,A
70 INPUT "A=";A
80 INPUT "A=",A
90 INPUT "";A
100 INPUT "",A
```

## TEXT

Syntax příkazu:

```
TEXT x,y,ř
TEXT x,y,v
```

Slouží ke vpisování textu na obrazovku. Čísla  $x,y$  určují pozici levého horního rohu prvního znaku textu.  $x,y$  musí být uvedeny tak jako pro PLOT, POSITION apod. v pixelech (obrazových bodech) použitého grafického režimu. Hodnoty  $x, y$  mohou být z intervalu  $\langle 0;65535.5 \rangle$ , jsou zaokrouhlovány podle vztahu  $x = \text{INT}(x + 0.5)$ . V příkazu TEXT může být obsažen pouze jeden výraz (na rozdíl od ? (PRINT)). Na konci obrazového řádku text skončí a nepokračuje na řádku následujícím (oproti příkazu ? (PRINT)). Barva textu (při použití příkazu TEXT v grafickém režimu), případně ATASCII kód znaků, z nichž je text složen (v textovém režimu), se volí příkazem COLOR.

```
Př. 33: 10 GRAPHICS 0:FOR Q=0 to 255:COLOR Q:
        TEXT 4,8,"AHOJ":PAUSE 5:NEXT Q
        20 TEXT 50,10,"Turbo-Basic XL 1.5"
        30 TEXT 235,154,"NEBUDE CELE"
        40 TEXT 100,30,123
        50 TEXT 100,40,5*EXP(20)
```

Př. 34:

```
10 GRAPHICS 0:FOR Q=1 TO 255:
    TEXT 4,8,"AHOJ":PAUSE 5:NEXT Q
```

Další příkazy I/O operací jsou popsány také v odstavci 3.4.13.

### 3.4.13 Příkazy pro disketovou jednotku

#### DIR

Syntax příkazu:

DIR

DIR "Dx:JMENO.EXT", kde x je číslo požadované disketové jednotky (1-8).

Odpovídá v DOSu příkazu A. V AB se pro výpis adresáře musí použít např. podprogram uveřejněný ve Zpravodaji ATARI klubu Praha č.4/87, str.18.

Slouží k výpisu adresáře (seznamu souborů) diskety na obrazovku. Příkazy DIR "D1:\*.\*)" a DIR vypisují celý adresář diskety v mechanice číslo 1. Příkaz DIR "D2:\*.\*)" provede totéž s disketou v disketové jednotce zapojené jako číslo 2. Příkaz DIR "D:\*.BAS" vypíše všechny soubory typu .BAS z diskety v mechanice č.1. Příkaz DIR "D:A\*.\*" vypíše všechny soubory začínající na A z diskety v mechanice č.1.

V názvu souboru lze použít i "?", který je chápán jako libovolný znak (ale jen jeden na rozdíl od \*). Příkaz DIR "D:OBR?.FIL" vypíše všechny soubory, které začínají na OBR, jméno mají dlouhé maximálně 4 znaky a jsou typu .FIL. Vypíše tedy např. soubory:

```
OBR1.FIL
OBR2.FIL
OBR3.FIL
OBR.FIL
```

ale nevypíše:

```
OBRAZ.FIL
OBR1.FIX
OB.FIL
```

Na konci výpisu je vždy uváděn počet volných sektorů na disketě.

U ostatních příkazů je "hvězdičková" konvence použitelná stejně, jako u příkazu DIR.

Ve všech těchto příkazech je přípustné používání řetězců:

Př. 35:

```
10 DIM D$(15)
20 D$="D1:PROGRAM1.BAS"
30 SAVE D$:DIR D$
40 DELETE D$:DIR D$
```

Toto je výhodné, např. používáme-li stejný název souboru vícekrát v jednom programu.

Poznámka recenzenta: TB akceptuje ve spolupráci s diskovým operačním systémem DOS 4.2 a jemu podobných existenci tzv. větvených (stromových) adresářů. Tato skutečnost dovo-luje TB pracovat s libovolnou DOSem nadefino-vanou logickou disketovou stanicí. Například výpis podadresářů se provede následujícím způsobem: DIR "D:\SUBDIR1;SUBDIR2;\*./\*".

Další nemalou výhodou TB je fakt, že volá systémovou disketovou stanicí jako D: , nikoliv jako D1: , což umožňuje pod DOSem 4.2 definovat aktuální fyzickou nebo logickou disketovou stanicí, tedy je-li např. aktuální disk D8: , tj. D:=D8: , pak k výpisu adresáře D8 stačí psát pouze DIR. Podobně k uložení souboru "ASIPÉ.TBS" pod určitý podadresář, jestliže nadefinujeme DOSem 4.2 D:=D1:TEXTY, stačí psát SAVE "D:ASIPÉ.TBS". Podobně platí vše i u ostatních příkazů pro disketovou jed-notku.

## RENAME

Syntax příkazu:  
RENAME "D:STARÝ,NOVÝ"

RENAME (<=>) X10 32, #7, 0, 0, "D:STARÝ,NOVÝ",  
resp. odpovídá v DOSu příkazu E.

Tento příkaz přejmenuje soubor se jménem "STARÝ" na soubor se jménem "NOVÝ".

Př. 36:

```
10 SAVE "D:PREJMENO.VAT":DIR:PAUSE 100
20 RENAME "D:PREJMENO.VAT,UDELANO.REN"
30 DIR:PAUSE 100
40 DELETE "D:UDELANO.REN":DIR
```

## DELETE

Syntax příkazu:  
DELETE "D:JMENO.EXT"

DELETE (<=>) X10 33, #7, 0, 0, "D:JMENO.EXT",  
resp. v DOSu příkazu D.

Maže z diskety soubor s názvem typu JMENO.EXT. Soubor nesmí být ochráněn, jinak interpret hlásí chybu ERROR- 167 LOCKED.

## LOCK

Syntax příkazu:  
LOCK "D:JMENO.EXT"

LOCK (<=>) XIO 35,#7,0,0,"D:JMENO.EXT", nebo v DOSu příkazu F.

Po vykonání tohoto příkazu je daný soubor na disketě chráněn před přepsáním, smazáním nebo přejmenováním. Lze z něj pouze číst. Že je soubor ochráněn, poznáme při výpisu adresáře podle hvězdičky nalevo od názvu souboru.

## UNLOCK

Syntax příkazu:  
UNLOCK "D:JMENO.EXT"

UNLOCK (<=>) XIO 36,#7,0,0,"D:JMENO.EXT", nebo v DOSu příkazu G.

Zruší ochranu souboru na disketě. Po vykonání tohoto příkazu je soubor opět přístupný pro RENAME, DELETE, přepsání ap.

## BLOAD

Syntax příkazu:  
BLOAD "D:JMENO.EXT"

Odpovídá v DOSu příkazu L.

Načte do počítače z diskety binární soubor (tj. program psaný ve strojovém jazyce). Pokud tento soubor obsahuje inicializační adresu, zapíše se tato do registru INITAD a program se automaticky ihned spustí od této adresy. Pokud program obsahuje pouze startovací adresu nebo neobsahuje ani startovací, ani inicializační adresu, dojde po načtení k návratu zpět do TB. V této souvislosti si povšimněte poznámky u příkazu BRUN!

Příkaz BLOAD lze použít také ve spojení s kazetovým magnetofonem (BLOAD "C:").

## BRUN

Syntax příkazu:  
BRUN "D:JMENO.EXT"

## Odpovídá v DOSu příkazu L.

Načte do počítače z diskety binární soubor a odstartuje jej, pokud obsahuje startovací nebo inicializační adresu. Pokud neobsahuje ani jednu z nich, předá po načtení OS počítače jeho řízení Turbo-BASICu.

Příkaz BRUN lze použít také ve spojení s kazetovým magnetofonem (BRUN "C:").

Pozn.: Existuje několik způsobů, jak odstartovat program ve strojovém jazyce:

a) Z Basicu pomocí funkceUSR.

b) Z DOSu pomocí funkce M.

c) Prostřednictvím OS. Po ukončení načítání souboru spustí CIO program od adresy uložené v registru RUNAD (\$2E0,\$2E1). Pokud je zde 0, vrátí CIO řízení tam, odkud byl tento podprogram volán, tj. do DOS, Basic, ... .

(CIO = Central Input / Output routine - řídí podprogramy pro vstupní/výstupní operace.)

Startovací adresu lze do RUNAD uložit kdykoliv během načítání souboru.

Kdykoliv během zavádění souboru do paměti lze okamžitě spustit program, uložíme-li jeho startovací adresu (v tomto případě se jí říká inicializační adresa) do INITAD, což je opět 2-bytový registr na adrese \$2E2. Řízení (tj. načítání) vrátíme CIO ukončením inicializačního programu instrukcí RTS. CIO potom pokračuje v načítání souboru do počítače (není-li již načítání ukončeno), případně na konci spustí program od startovací adresy (obsahuje-li ji). Je-li v souboru inicializační adresa až na konci souboru, tj. program je spuštěn, když je již jeho načítání ukončeno, funguje instrukce RTS na konci inicializačního programu stejně jako na konci programu odstartovaného pomocí RUNAD, čili program pokračuje tam, odkud bylo načítání voláno (TB, DOS, ...).

### Př. 37:

Příklad ukazuje rozdíly mezi programy ve strojovém jazyce s a bez startovací či inicializační adresy. Programy jsou napsány pomocí makroassembleru MAC-65, je ovšem možné napsat je i pomocí assemblerů jiných (samozřejmě je třeba respektovat rozdíly mezi MAC-65 a příslušným assemblerem).

Opíšte zdrojový program a přeložte jej příkazem uvedeným pod výpisem (ASM...). Až budete mít zapsány a přeloženy všechny tři strojové programy (a, b, c), načtete do počítače TB, napišete v něm program e), do disketové jednotky č. 1 vložte disketu s přeloženými programy a program spusťte (Basic!).

Pokud nemáte možnost napsat strojové programy v assembleru, napište a spusťte nejprve program d), který vytvoří strojové programy z Basicu, a potom napište a spusťte program označený e).

Po spuštění programu e) se načítají strojové programy příkazy BLOAD a BRUN, a pokud se po načtení spustí, změní se barva pozadí a znaků.

a) bez run a init adresy

```

10 START = $0600
20 COLOR1 = 709
30 COLOR2 = 710
40      *= START
50      LDA #255
60      EOR COLOR1
70      STA COLOR1
80      LDA #255
90      EOR COLOR2
0100    STA COLOR2
0110    RTS
ASM, #D: NOTHING.COM

```

b) s run adresou

```

10 START = $0600
20 COLOR1 = 709
30 COLOR2 = 710
40 RUNAD = $02E2
50      *= START
60      LDA #255
70      EOR COLOR1
80      STA COLOR1
90      LDA #255
0100    EOR COLOR2
0110    STA COLOR2
0120    RTS
0130    *= RUNAD
0140    .WORD START

```

ASM, #D: RUNAD.COM

c) s init adresou

```

10 START = $0600
20 COLOR1 = 709
30 COLOR2 = 710
40 INITAD = $02E2
50      *= START
60      LDA #255
70      EOR COLOR1
80      STA COLOR1
90      LDA #255
0100    EOR COLOR2
0110    STA COLOR2
0120    RTS

```

```
0130      *= INITAD
0140      .WORD START
```

ASM, ,#D: INITAD.COM

d) Program na vytvoření strojových programů a, b, c z TB

```
10 DIM W$(29), D$(2)
20 FOR Q=1 TO 29
30 READ D$: W$(Q)=CHR$(DEC(D$))
40 NEXT Q
50 OPEN #1, 8, 0, "D: NOTHING.COM"
60 BPUT #1, ADR(W$), 23
70 CLOSE
80 OPEN #1, 8, 0, "D: RUNAD.COM"
90 BPUT #1, ADR(W$), 29
100 CLOSE
110 W$(24, 24)=CHR$(E2)
120 W$(26, 26)=CHR$(E3)
130 OPEN #1, 8, 0, "D: INITAD.COM"
140 BPUT #1, ADR(W$), 29
150 CLOSE
200 DATA FF, FF, 0, 6, 10, 6, A9, FF, 4D, C5, 2,
      8D, C5, 2, A9, FF, 4D, C6, 2, 8D, C6, 2, 60,
      E0, 2, E1, 2, 0, 6
```

e) ukázkový program na BLOAD a BRUN

```
10 DPOKE 709, 12
20 ? "Zacinam !": PUT 253
30 ? "BLOAD - NIC": BLOAD
  "D: NOTHING.COM": PUT 253: PAUSE 50
40 ? "BRUN - NIC": BRUN "D: NOTHING.COM":
  PUT 253: PAUSE 50
50 ? "BLOAD - RUNAD": BLOAD
  "D: RUNAD.COM": PUT 253: PAUSE 50
60 ? "BRUN - RUNAD": BRUN "D: RUNAD.COM":
  PUT 253: PAUSE 50
70 ? "BLOAD - INITAD": BLOAD
  "D: INITAD.COM": PUT 253: PAUSE 50
80 ? "BRUN - INITAD": BRUN
  "D: INITAD.COM": PUT 253: PAUSE 50
```

### 3.4.14 Ostatní příkazy

**\*B (\*B+)**

Po tomto příkazu způsobí stisknutí klávesy /BREAK/ přerušeni programu, ale interpret chybu neohlásí. Program lze před přerušeni chránit, a to ošetřením TRAPem (stejně jako obyčejnou chybu).

```
PR.38: 10 TRAP 100:*B
        20 DO :LOOP
        100 ? "BREAK"
```

\*B-

Ruší příkaz \*B (\*B+), interpret se vrací do normálního režimu. Příkaz RUN provádí automaticky i \*B-.

PAUSE v

Nahrazuje prázdné čekací smyčky FOR/NEXT v AB (FOR I=0 TO 100:NEXT I). Přerušuje běh programu na dobu v/50 sekundy. Tento čas ovšem není přesný (stejně tak jako u příkazů TIMES apod.). Hodnota výrazu v musí být z intervalu (0;65535.5).

V AB se ke krátkému zdržení používá mj. příkazu Q=1^1. V TB je však tento výpočet mnohem rychlejší než v AB, neboť umocňování s celočíselným exponentem menším než 100 se provádí opakovaným násobením a ne jako v AB podle vzorce  $a^b=10^{(b*\log(a))}$ . Nejjednodušší Q=1^1 (v AB trvá asi 180 ms, v TB jen 21 ms) je možno nahradit příkazem PAUSE 0. Rovněž prázdné čekací smyčky FOR/NEXT je možné nahradit příkazy PAUSE, a to zhruba v poměru: 10 cyklů v AB odpovídá 1 v argumentu příkazu PAUSE (platí jen přibližně!), u malých hodnot argumentu je přesnější ještě odečíst z argumentu 1, viz dále).

Při používání příkazu PAUSE s velmi malými hodnotami argumentu, či potřebujete-li docílit poměrně přesného zdržení programu, musíte mít na zřeteli, že vykonání samotného příkazu PAUSE zabere nějaký čas! Konkrétně vykonání příkazu:

ČÍ PAUSE 0

trvá 19.3 ms. Každé další zvětšení hodnoty argumentu o 1 už prodlouží čekání o 20.0 ms. Tj. PAUSE 1 trvá 39.3 ms, PAUSE 2 trvá 59.3 ms, ....

TIMES=

Syntax příkazu:  
TIMES= n

Slouží k nastavení vnitřních hodin RTCLCK na určitou hodnotu. Má stejný formát jako funkce TIMES.

Př. 39:

```
CLR:DIM D$(6):D$="123456":TIMES= D$:  
? TIMES
```

Tyto příkazy nastavují čas RTCLCK na 12 hodin 34 minut a 56 sekund.

TIMES= "000000" nastavuje čas RTCLCK na nulu.

### 3.5 Funkce TB

#### DPEEK(v)

Analógie ke dvojici příkazů POKE-DPOKE je PEEK-DPEEK, tj. argumentem funkce je šestnáctibitové číslo  $(LO+HI*256)$  uložené na adrese  $v$  a  $v+1$ . Na adrese  $v$  je nižší byte hodnoty argumentu (LO-byte), na adrese  $v+1$  vyšší byte (HI-byte). Argument  $v$  může být výraz, jehož hodnotou je celé číslo z intervalu  $\langle 0;65535 \rangle$ .

DPEEK a DPOKE se používají především při práci s ukazovátky (Basic, OS, DOS, ...) a se šestnáctibitovými čísly.

Př. 40:

```
A=DPEEK(88):? A:DPOKE 88,A+300:  
? DPEEK(88) (návrat DPOKE 88,A)
```

#### TIME

$TIME \langle \Rightarrow \rangle PEEK(20)+PEEK(19)*256+PEEK(18)*65536$   
Hodnotou této funkce je čas vnitřních hodin RTCLCK, který je na adresách 18-20 v nulté stránce paměti, a to v padesátinách sekundy. Čas v sekundách získáme jako  $TIME/50$ . Pozor!, nelze změnit čas hodin přiřazením, například  $TIME=18000$ . Po tomto příkazu se hodnota 18000 přiřadí proměnné TIME, čas vnitřních hodin se však nezmění. TIME je funkce a ne proměnná! Ke změně hodnoty hodin (nastavení na určitou hodnotu, např. vynulování) je nutno použít příkazu  $TIMES=$ , případně příkazů POKE. Viz příklad 41.

Jednou z možností využití je měření doby vykonávání příkazů.

## TIMES

Hodnotou této funkce je čas ve formě šestiznakového ř formátu "hhmmss":  
hh=hodiny, mm=minuty, ss=sekundy;  
hh=0-23, mm=0-59, ss=0-59.  
TIMES není proměnná, není ji proto třeba deklarovat.

Př. 41:

```
? TIMES → 153040, což znamená čas 15  
hodin 30 minut 40 sekund.  
? TIME → 2816000  
? TIME/50 → 56320
```

Příkaz TIMES vychází stejně jako TIME z RTCLCK na adresách 18-20. Čas RTCLCK ovšem není přesný, neboť není odvozen z frekvence přesně 50 Hz.

## TRUNC(v)

Výsledkem funkce je celé číslo vzniklé "odřiznutím" části čísla vpravo od desetinné tečky.

Př. 42:

I	TRUNC(I)	FRAC(I)	INT(I)
3.1	3	0.1	3
3.5	3	0.5	3
3.8	3	0.8	3
-3.1	-3	-0.1	-4
-3.5	-3	-0.5	-4
-3.8	-3	-0.8	-4

## FRAC(v)

Hodnotou funkce je desetinná část čísla (viz př. 42).

## HEX\$(v)

Funkce slouží k převodu čísel z desítkové do šestnáctkové soustavy. Argument musí být číslo z intervalu <0;65535,5). Je-li INT(v) menší než 256, poskytuje funkce dvojmístné číslo, jinak je čtyřmístné.

Př. 43:

? HEX\$(48192), HEX\$(4\*53)

DEC(ř)

Je opakem funkce HEX\$ a slouží k převodu čísel z šestnáctkové do desítkové soustavy. Hodnotou této funkce je číslo z intervalu <0;65535>. Má-li ř více než 4 znaky, bere funkce jen poslední 4. Vyskytuje-li se v ř jiný znak než číslice nebo znaky A, B, C, D, E, F (znaky šestnáctkové soustavy), vypočte funkce desítkovou hodnotu šestnáctkového čísla od začátku ř do tohoto znaku.

Př. 44:

```
10 ? DEC("FFFF"), DEC("FF"), DEC("EFFFF")
20 ? DEC("EE/FF"), DEC("4C")
30 DIM A$(4): A$="A000": ? DEC(A$)
```

RND (RND(v))

Stejně jako v AB je výsledkem pseudonáhodné číslo z intervalu <0;1>. V TB stačí psát RND místo RND(0). Hodnota funkce RND na hodnotě výrazu v nezávisí.

RAND(v)

RAND(n) odpovídá TRUNC(RND\*n). Funkce RAND poskytuje celé pseudonáhodné číslo z intervalu <0;v> pro v=0 resp. z intervalu (v;0), je-li v<0 (jako argument bere TRUNC(v)).

Místo RAND(256) bývá výhodnější používat PEEK(53770). Funkce RAND je totiž odtud vypočítávána TB interpretem, PEEK(53770) je tudíž rychlejší.

Př. 45:

Potřebujeme získat náhodné racionální číslo X ( $0 \leq X < 2.3$ ), a celé náhodné číslo Y ( $10 \leq Y \leq 100$ ):

```
10 X=RND*2.3
20 Y=10+RAND(91)
```

RAND(91) (!!!), protože výsledkem jsou celá čísla od 0 do 90!

## INSTR

Syntax funkce je:  
INSTR(Ř1,Ř2,k)  
INSTR(Ř1,Ř2)

Funkce ke hledání řetězce Ř2 v řetězci Ř1. Nalezne-li její, výsledkem funkce je jeho poloha v řetězci Ř1, jinak je výsledkem nula. Hledat se začíná od začátku řetězce Ř1, je-li uveden 3. parametr (k), začíná se hledat od (k+1)-tého znaku (k-tý znak se neporovnává!). Je-li k > LEN(Ř1), výsledkem je vždy 0. k musí být z intervalu <0;65535.5>.

Př. 46:

```
10 DIM A$(20), B$(10)
20 A$="POUTSTCTPASONNE"
30 D=INSTR(A$, "ST"): ?D, (D+1)/2
40 A$="123456789ABCDEF"
50 ? INSTR(A$, "C")
60 A$="KALAFUNA": B$="A"
70 I=0: J=0
80 DO
90   I=INSTR(A$, B$, I)
100  IF I
110    ? I, : J=J+1
120  ELSE
130    EXIT
140  ENDIF
150 LOOP
160 ? : ? J
```

## UINSTR

Syntax funkce:  
UINSTR(Ř1,Ř2,k)  
UINSTR(Ř1,Ř2)

Pracuje podobně jako INSTR, pouze se neberou v úvahu při porovnání bity 5 a 7 ATASCII kódu (tj. 6. a 8. bit), tj. nerozlišují se velká, malá a inverzní písmena, ... (viz tabulka ATASCII kódů).

## INKEY\$

Tato funkce má hodnotu řetězce (jednoho znaku), odpovídajícího právě stisknuté klávese. Není-li stisknuta žádná klávesa, přiřadí se jí prázdný řetězec (tj. ""). INKEY\$ není třeba deklarovat, nejedná se o proměnnou.

```
Př. 47:
100 ? INKEY$;" ";
110 GOTO 100
```

```
Př. 48:
100 IF INKEY$="" THEN 100
110 ? "110"
```

ERR

Je ekvivalentní PEEK(195), používá se ke zjištění obsahu registru, do něhož TB ukládá číslo chyby pro chybové hlášení.

```
Př. 49:
10 CLS:TRAP 30
20 A=3/0
30 ? ERR
40 TRAP 60
50 POSITION 300,300
60 POSITION 2,2: ? ERR
```

ERL

Je ekvivalentní PEEK(186)+PEEK(187)\*256, resp. DPEEK(186), tj. výsledkem je čí, na kterém byla indikována poslední chyba, nebo na kterém byl přerušen program. Oba příkazy se používají při ošetření chyb pomocí příkazů TRAP. Doporučuji načíst hodnotu ERL i ERR ihned po skoku prostřednictvím TRAP, než se změní (ERL se např. mění stisknutím klávesy BREAK).

```
Př. 50:
100 INPUT A,B,C,D,E,F:TRAP 1000
110 G=A/B
120 H=C/D
130 I=E/F
140 ? G,H,I:TRAP 40000:END
1000 TRAP 1000:IF ERR<>11 THEN 1100
1020 ON (ERL-100)/10 GOTO 1050,1060,
1070
1050 G=0:GOTO 10+ERL
1060 H=0:GOTO 10+ERL
1070 I=0:GOTO 10+ERL
1100 ? "Chyba c. ";ERR;" na radku c. ";
ERL:STOP
```

### 3.6 Operátory TB

#### 3.6.1 Aritmetické operátory

##### DIV

Funkce DIV je funkcí celočíselného dělení. Výsledkem operace a DIV b je podíl ekvivalentní TRUNC(a/b). Další popis u MOD.

Př. 51:

```
10 ? 17 DIV 5      → 3
20 ? 23.456 DIV 9.3 → 2
```

##### MOD

Vyčísluje zbytek po celočíselném dělení. a MOD b  $\langle \Rightarrow \rangle$  a-b\*TRUNC(a/b)  $\langle \Rightarrow \rangle$   $\langle \Rightarrow \rangle$  a-b\*(a DIV b).

Př. 52:

```
10 ? 17 MOD 5      → 2
20 ? 23.456 MOD 9.3 → 4.656
```

Příkazy DIV a MOD nejlépe pochopíte, když se vrátíte do dětských let na základní školu. Při dělení celých čísel poskytují DIV a MOD podíl a zbytek dělení:

```
17 : 5 = 3 a zbytek 2
      DIV      MOD
```

Příkazy DIV a MOD lze s výhodou použít při rozkladu 16-bitových čísel na 8-bitová (LO = dolní byte, HI = horní byte).

Př. 53:

```
SLOVO=LO+HI*256
zpětný rozklad:
LO=SLOVO MOD 256; HI=SLOVO DIV 256
na rozdíl od:
HI=INT(SLOVO/256); LO=SLOVO-HI*256
```

Výsledky aritmetických operací zobrazuje TB na rozdíl od AB na 10 platných míst (někdy jen na 9, např. ? 10/7 ). Výpočty jsou však několikrát rychlejší, než v AB. TB totiž využívá vlastní matematické podprogramy uložené v paměti RAM. Např. výpočet umocňování na celočíselný exponent menší než 100 je prováděn opakovaným násobením a ne výpočtem podle vzorce  $a^b = \exp(b \cdot \log(a))$  jako v AB.

### 3.6.2 Logické operátory

& ! EXOR
----------------

& = binární AND  
! = binární OR  
EXOR = binární exkluzivní OR

Tyto operátory pracují s 16-bitovými celými čísly, tj. s čísly z intervalu <0;65535>, nikoliv však s booleovskými hodnotami pravda (<>0) a nepravda (=0) jako operátory AND, OR a NOT.

Př. 54:

157 = 10011101  
240 = 11110000

157 & 240 → 10010000 = 144  
157 ! 240 → 11111101 = 253  
157 EXOR 240 → 01101101 = 109

na rozdíl od:

157 AND 240 → 1  
157 OR 240 → 1

Těchto příkazů se využívá např. při nulování některých bitů ap.

Př. 55:

```
10 GRAPHICS 0:LIST:B=PEEK(710)
20 FOR I=0 TO 100
30 POKE 710,RAND(256) & 247:PAUSE 0
40 NEXT I:POKE 710,B
```

Pak zkuste na ř.č. 30 umazat "& 247"  
(pozor na barevné televizory!).

Užše uvedený program způsobí blikání pozadí tím, že posílá náhodnou barvu a jas na registr COLOR2 (= barva pozadí v grafice 0). "Chmurnější atmosféra" (lze využít v programu např. pro bouři, resp. výbuch) je dosažena zmenšením jasu tím, že je vždy použitím & 247 nulován bit 3 registru, čímž se jas sníží na 1/2. Jas totiž určují bity 3, 2 a 1, barvu bity 7, 6, 5 a 4, bit 0 je nepoužit (proto nelze u ATARI programově nastavit barevných odstínů 256, ale jen 128!).

/7 6 5 4/3 2 1/0/  
/ barva / jas /-/

Způsob nulování bitu 3 v minulém příkladu:

$a \& 1 = a$  (protože  $1\&1=1$ ;  $0\&1=0$ )  
 $a \& 0 = 0$

RAND(256) = aaaaaaaaaa  
247 = 11110111  
RAND(256) & 247 = aaaa0aaa

Další příklady využití logických operátorů TB najdete například v ZAK 4/87, na straně 19 (autor Ing. V. Friedrich).

Pro úplnost uvádím ještě tabulku pravdivostních hodnot logických operátorů AND, OR, NOT:

A, B = hodnoty výrazů

A	B	A AND B	A OR B	NOT(A)
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

### 3.6.3 Priorita operátorů v TB

Priorita operátorů v TB je uvedena v sestupném pořadí. Nejprve se vykonávají operace uvedené nejdříve, jsou-li 2 operátory na stejném řádku (mají stejnou prioritu), vykonávají se zleva doprava tak, jak jsou zapsány v příkazovém řádku.

- (unární) znaménko mínus  
& ! EXOR  
= <= >= < > <>  
^  
\* /  
DIV MOD  
+ -  
OR AND  
NOT

### 3.7 Speciální příkazy, funkce, operátory a výrazy

\$abcd

Znak \$ před číslem znamená, že jde o číslo v šestnáctkové soustavě. Tak například číslo \$600=1536, \$E=14. Číslo musí být kladné celé z intervalu <0;FFFF>, tj. <0;65535>. Číselná konstanta definovaná tímto způsobem je stejně jako normální konstanta uchovávána jako BCD konstanta, zaujímá tudíž 6 bytů.

Př. 56:

POKE \$CB40, \$41: ? \$89AB

%a

Čísla 0, 1, 2 a 3 jsou při použití se značkou % (%0, %1, %2, %3) definována jako 1-bytové konstanty. Nejsou zaznamenána v tabulce jmen ani hodnot proměnných. Uzhledem k tomu, že jsou tato čísla velmi často používána a zaujímají pouze 1 byte (úspora 6 bytů na každou náhradu 0, 1, 2, 3 → %0, %1, %2, %3), lze při jejich důsledném používání u delších programů dosáhnout značné úspory paměti! (Číselné konstanty potřebují 7 bytů paměti - 6 bytů pro uložení hodnoty konstanty v BCD kódu + 1 byte pro označení, že jde o BCD konstantu.)

Př. 57:

OPEN #1, 0, 0, "E:": ? FRE(0) → 33989 bytů  
OPEN #%1, 0, %0, "E:": ? FRE(0) → 34001 byte  
=> úspora 12 bytů!

Takto zjistíte, že při ? FRE(%0) poskytuje TB 34027 bytů místo 34021 při ? FRE(0).

Užše uvedené hodnoty pro FRE(0) platí při použití TB pod DOS 2.5. Při použití jiné systémove konfigurace, jiného DOSu či kazetové verze TB mohou být hodnoty jiné, ale rozdíl bude vždy stejný.

Protože Basic chápe znaménko mínus před číslem jako operátor a ne jako součást číselné konstanty, lze značky % použít i u záporných čísel, tj. -1, -2, -3, o čemž se snadno přesvědčíte:

Př. 58:

```
A=-2: ? FRE(0)
A=-%2: ? FRE(0)
```

Za, nesmí být použito za příkazem DATA!

..

Uvozovky lze v TB vytisknout v řetězci tak, že je napíšeme 2x vedle sebe uvnitř řetězce. Je-li mezi uvozovkami jeden pár uvozovek, jsou vytištěny jedny uvozovky. Pro další uvozovky je třeba přidat další dvojici.

Př. 59:

```
? "JEDNA""DVA" → JEDNA"DVA
<=> ? "JEDNA"; CHR$(34); "DVA"
? "A""""B" → A""B
<=> ? "A"; CHR$(34); CHR$(34); "B"
```

-

Ve jménu proměnné, procedury nebo návěští lze použít znak - (SHIFT -) stejně jako alfanumerické znaky, nesmí však být použit jako první znak jména.

Př. 60:

```
POCET-KUSU=60: GO# SMER-VLEVO
EXEC VYPOCET-SUMY: PROC VYPOCET-SUMY
```

### 3.6 Chybová hlášení TB

Chybová hlášení jsou v TB doplněna krátkými hesly označujícími chybu a mají tvar: ERROR - ch?HESLO, kde ch je číslo chyby, například: ERROR - 25?D0

Př. 61:

```
10 GOSUB 30
20 END
30 DEL 0,19:RETURN
```

Př. 62:

```
10 WHILE I<10
20   FOR J=1 TO 10
30     I=I+1
40   WEND
50 NEXT J
```

Usvětlivky k chybovým hlášením jsou uvedeny v kapitole č.5.

### 3.9 Paměť a TB

TB XL 1.5 není v paměti ROM počítače, proto je ho zapotřebí načítat do paměti RAM. TB zabírá asi 16 kB, je tedy podstatně delší, než AB, který má 8 kB. Větší část interpretu (12 kB) je umístěna v paměti RAM pod ROM. Zbýlých 6 kB je uloženo v paměti od adresy \$2000, tedy za D05em.

Interpret si přepíná mezi RAM a ROM podle toho, zda potřebuje podprogramy 05 v ROM, nebo své, umístěné v paměti RAM.

To, že TB zabírá paměť v RAM od adresy \$C000 po \$FFFF (mimo adresy \$D000 - \$D7FF, které jsou obsazeny čipy GTIA, POKEY, PIA a ANTIC), je důvod, proč nelze v TB použít RAM-disk pro ATARI 800 XL a XE, který používá právě tuto oblast paměti.

TB při načítání odpojuje 8 kB ROM, v níž je uložen Basic. Proto poskytuje TB v grafickém módu 8 téměř 34 kB volných pro program v Basicu, (tj. o 2 kB více než AB). U místě, kde je v AB adresována ROM Basicu (\$A000 - \$BFFF), je v TB paměť RAM a jsou zde umístěna obrazová data a DL.

Příkazy POKE do oblasti, kde je v RAM interpret TB (\$2000 - \$3628), by mohly vést ke zhroucení systému, v lepším případě ke špatné práci interpretu. Vyvarujte se proto změn v paměti v níž leží TB ! Během provádění příkazů POKE, DPOKE, MOVE a -MOVE je sice většinou oblast RAM, v níž je TB, blokována, ale ... Při vykonávání těchto příkazů se automaticky odpojuje RAM od \$C000 výše a připojuje se ROM, kde nemohou tyto příkazy napáchat žádnou škodu (opět to samozřejmě neplatí pro čipy na adresách \$D000 - \$D7FF). Kromě toho je oblast ROM volána ještě funkcemi PEEK, DPEEK, I/O příkazy jako PRINT, INPUT, BPUT, BGET atd. (ne ale při PAINT), stejně jako při vykonávání aritmetických operací, funkcí STR\$, VAL, ...

Následující tabulka znázorňuje schematicky paměť TB ve srovnání s AB při použití diskového operačního systému DOS 2.5:

Atari-Basic			Turbo-Basic		
FFFF D800	OS-ROM	RAM volná	FFFF D800	OS-ROM	RAM inter- pret
D7FF D000	HW-registry		D7FF D000	HW-registry	
CFFF C000	OS-ROM	RAM volná	CFFF C000	OS-ROM	RAM inter- pret
BFFF A000	Basic ROM	RAM volná	BFFF	video-RAM	
9FFF	video-RAM .....			.....	
	Basic-zásobník .....			Basic-zásobník .....	
	Pole a řetězce .....			Pole a řetězce .....	
	Program v Basicu		3629	Program v Basicu	
	Program v Basicu		3628	interpret	
2080			2080		
207F 0700	DOS 2.5		207F 0700	DOS 2.5	
06FF 0600	6. stránka		06FF 0600	6. stránka	
05FF 0000	Používá OS		05FF 0000	Používá OS	

### 3.10 Kompatibilita AB a TB

U programu napsaného v AB mohou při jeho spuštění v TB nastat nejrůznější komplikace. V následujícím přehledu jsou shrnuty oblasti nekompatibility AB s TB:

1. DL a umístění obrazových dat v paměti
2. PMG
3. obsazení paměti (adresy \$2000 - \$3620, \$C000 - \$FFFF, ...)
4. klíčová slova
5. rychlost, případně způsob vykonávání určitých příkazů a operací v AB a TB

Jak tyto nedostatky řešit?

#### Ad 1

U DL i obrazových dat nám pomůže přeadresování nebo vztažení adresování relativně k paměťové buňce dané např. ukazovátkem na adrese 560 (ukazuje na 1. byte DL) nebo na adrese 88 (ukazovátka na 1. byte obrazové paměti). Použitím relativního adresování se většina programů v Basicu stane použitelnými v AB a v TB zároveň.

#### Ad 2

U PMG jde o přesun dat do volného paměťového prostoru, kdy je třeba respektovat uložení TB v paměti. Opět pomůže např. adresování vztažené k některému z ukazovátek tj. relativní adresování.

ne → POKE 40000,1 , ale → POKE DPEEK(88),1  
ne → POKE 106,152, ale → POKE 106,PEEK(106)-8

Poznámka recenzenta: Pro výpočet správných hodnot pro relativní adresování je dobré znát adresy umístění SM a DL při používání AB. Jinak také pomůže přičtení konstanty 6192 či 32 (dvoubytovou pro vektory, 32 pro RAMTOP aj. registry, obsahující 1 byte). Program potom však nebude fungovat v AB!

#### Ad 3

Posunutí video-RAM v TB vůči AB lze obejít zařazením příkazů POKE 106,160: GRAPHICS 0 na začátek programu, pro Basic však máte nyní o 6 kB méně než v AB.

O uložení TB v paměti pojednává blíže kapitola 3.9.

#### Ad 4

TB obsahuje některá klíčová slova, která nejsou AB definována, ale která jsou v AB často používána jako jména proměnných. Jde např. o HEX\$, ERR, ERL, DEL, TIME, ...

Tato klíčová slova nelze obvykle použít jako jména proměnných. Při načtení programu a spuštění se tyto chyby nemusí projevit ihned, program však může dávat špatné výsledky. Např. při ERR=10 se použije proměnná ERR, ale při A=ERR nebo ? ERR se použije funkce ERR. Jediná možnost, jak těmto kolizím zabránit, je tyto proměnné přejmenovat. Pomoci si můžete například příkazem DUMP.

#### Ad 5

Případnou příliš vysokou rychlost vykonávání programu je nutné snížit přidáním či prodloužením zdržovacích podprogramů nebo příkazů. Některé rady jsou uvedeny u příkazu PAUSE.

### 3.11 Překladač TB

K TB existuje také překladač Turbo-BASIC XL COMPILER Version 1.1 z roku 1985, autorem je Frank Ostrowski.

Program napsaný v TB se přeložením TB překladačem (Compilerem - dále označován také jako CTB) obvykle zkrátí a 2-6 x zrychlí.

Nebudu se zde rozepisovat o rozdílech mezi interpretem a překladačem a o tom, jak který pracuje. Stačí, budeme-li vědět, jak přeložit zdrojový program napsaný v TB a jak překlad spustit, případně jak opravit chyby.

Překladač TB se sestává ze dvou samostatných programů - překladače (Compileru), který překládá zdrojový program z TB do strojového kódu a z programu runtime, kterým se přeložené programy spouští a který obsahuje knihovnu podprogramů používanou přeloženými programy.

Postup při překládání zdrojového programu je následující:

Předpokládáme, že máte uložený program v TB na disketě pod názvem PROGRAM.TBS. Program musí být uložen příkazem SAVE!

1. Vypněte počítač.
2. Vložte disketu s DOSem do disketové jednotky.
3. Stiskněte klávesu /OPTION/ na konzole, podržte ji a zapněte počítač.

4. Načtou se soubory DOSu a vypíše se MENU.
5. Do disketové jednotky vložte disketu, na níž máte uložen překladáč.
6. Stiskněte klávesy /L/ (=BINARY LOAD) a /RETURN/.
7. DOS se Vás zeptá na jméno souboru, který chcete načíst. Zadejte jméno, pod kterým je uložen překladáč, např. D1:COMPILER.COM a odešlete jej stlačením klávesy /RETURN/.
8. Po načtení překladače se vypíše krátký návod k použití.
9. Nyní máte několik možností:
  - a) stisknutím kombinace kláves /CONTROL/ a /D/ s následným /J/ (ja) načtete zpět DOS (/N/ (nein) vrací do MENU);
  - b) /CONTROL/ /R/ s následným /J/ (ja) provede studený start počítače (/N/ (nein) vrací do MENU);
  - c) pokračovat v překládání - přejděte k bodu č.10).
10. Stiskněte klávesu 1 až 8, podle toho, ve které disketové jednotce máte disketu s programem, který budete překládat (8 v případě, že jej máte uložen v RAMdisku). V levé horní části obrazovky se napíše zařízení, ze kterého je prováděn výpis, pod ním se ve třech sloupcích vypíše adresář.
11. Na názvu prvního souboru (vlevo nahoře) se objeví kurzor (název je v inverzi). Tímto kurzorem lze pohybovat pomocí kláves se šipkami. Není při tom nutno mačkat tyto klávesy přes /CONTROL/. Pomocí šipek nastavte kurzor na název souboru, který chcete přeložit a stiskněte /RETURN/.
12. Za názvem zařízení (D1: apod.) se napíše název překládaného programu. Napravo od něj se vypisují čísla překládaných řádků.
13. Pokud proběhl překlad bez chyb, napíše program:

```
Keine Fehler (= bez chyby)
Programmlaenge = X Bytes
(= délka programu = X bytů)
```

- a zeptá se Vás na jméno přeloženého programu. Přitom je povinný typ souboru .CTB. Název napíše (.CTB se automaticky odsouvá) a odešlete klávesou /RETURN/.
14. Přeložený program se pod tímto názvem uloží na disketu.
  15. Dále se Vás překladáč zeptá, zda chcete ještě jednu kopii. Pokud ji chcete, stiskněte klávesu /J/ (ja), pokud ne, stiskněte /N/ (nein).
- Pokud zadáte N, program se vrátí do MENU překladače, tj. do bodu 9., nyní však už máte

- zdrojový program přeložený, jde jen o to jej spustit.
16. Do systémové disketové jednotky vložte disketu, na níž je DOS.
  17. Stiskněte současně /CONTROL/ a /D/.
  18. Načte se Vám DOS a vypíše se DOS-MENU.
  19. Zadejte /L/ a /RETURN/.
  20. Do disketové jednotky vložte disketu s programem runtime.
  21. Zadejte jméno, pod nímž máte uložený program runtime, např. RUNTIME.COM a stiskněte /RETURN/.
  22. Načte se program runtime, který automaticky vyhledá a spustí program s názvem AUTORUN.CTB, obdobně jako TB spouští program AUTORUN.BAS. Na rozdíl od TB však runtime ohlásí chybu, pokud program AUTORUN.CTB nenajde. Pokud tedy máte na téže disketě s programem runtime program AUTORUN.CTB, automaticky se tento program spustí. Není-li tam, ohlásí po načtení runtime chybu 170 (soubor nenalezen) na řádku 0 (\$297F) a vypíše jednorádkové MENU s volbami DOS, RUN a LOAD.
  23. Stiskněte klávesu odpovídající prvnímu písmenu funkce podle toho, zda chcete načíst DOS (D), spustit načtený program (R) nebo načíst přeložený program do počítače (L).
  24. Volba /D/ (DOS) Vás vrátí do DOSu.
  25. Po volbě /L/ (LOAD) jste dotazováni na název programu. Zadejte název přeloženého programu, který chcete spustit a odešlete ho klávesou /RETURN/. Extendr ".CTB" není třeba zadávat. Po načtení je program automaticky spuštěn.
  26. Vykonávání programu lze přerušit stisknutím klávesy /BREAK/ (není-li programově zablokována). Poté se Vám objeví MENU stejné jako v bodě 22. Program lze znovu spustit volbou RUN, tj. stisknutím klávesy /R/.

Máte-li již vytvořený překlad programu v TB, lze jej spustit vykonáním bodů 1-4 a 19-25.

Je-li DOS, překladač, runtime i zdrojový program na téže disketě, není zapotřebí diskety vyměňovat. Pokud nazvete překladač nebo runtime AUTORUN.SYS, spustí se tento program po studeném startu počítače automaticky.

Pokud někde narazí překladač nebo runtime na chybu, ohlásí její číslo a na kterém řádku zdrojového programu k ní došlo. Opravit chybu lze pouze v TB přímo ve zdrojovém programu. Pak je nutno provést překlad celý znovu, tj. od bodu 1.

## Poznámka recenzenta:

U kazetové verze překladače Turbo - BASICU (u TURBO 2000) dodržujte následující postup:

1. Nahrajte příslušným způsobem operační systém (pro CTB lze použít jen TOS XL 2.4 nebo TOS 4.1).
2. Pomocí direktivy L /RETURN/ a zadáním jména souboru nahrajte překladač.
3. Odstartujte jej stiskem 'Y'.
4. Nyní se nabídnou možnosti překladače. Jsou obdobné možnostem disketové verze.
5. Stiskem 1, 2 nebo 3 si vyberte příslušné zařízení pro načítání (platí pro TOS XL 2.4)

- 1 ..... zařízení "T:"
- 2 ..... RAMdisk (na 000 XL verzi TOSu není možné použít)
- 3 ..... zařízení "C:"

a po dvojím RETURN se provede nahrání prvního nalezeného souboru ve formátu příslušného zařízení.

6. Po ukončení kompilace se vypíše stav (bez chyb, chyby; délka) a zadáme jméno pro přeložený program. Zde postupujeme až do opuštění CTB jako u disketové verze.

7. Při opuštění CTB samozřejmě nemusíme vkládat disketu s DOSem.

8. RUNTIME nahrajeme z TOSu pomocí L /RETURN/ a jména, a odstartujeme jej stiskem 'Y'.

9. RUNTIME se snaží najít soubor AUTORUN.CTB na zařízení "T:". To přerušíme stiskem BREAK.

10. Zadání jména, nahrání souboru a další operace jsou stejné jako u disketové verze.

Nejčastější zdroje chyb při překládání a spouštění programu jsou:

### 1. Nestrukturované programování

Chyby tohoto druhu vznikají nedodržením zásad strukturovaného programování. Nej důležitějšími, často porušovanými zásadami jsou:

a) Každá struktura má mít jeden začátek a jeden konec. Chybu při překladu vyvolávají například tyto konstrukce:

Př. 63:

```
10 FOR I=1 TO 10
20   J=RND
30   IF J<0.1 THEN NEXT I
40   K=J
50 NEXT I
```

Př. 64:

```
10 EXEC PODIL
20 END
30 PROC PODIL
40   IF K=0 THEN P=1E10: ENDPROC
50   P=J/K
60 ENDPROC
```

Př. 65:

```
10 IF K=1 THEN 30
→ 20 FOR I=1 TO 10
30 GOTO 50
→ 40 FOR I=1 TO 20
50 REM
60 NEXT I
```

b) Začátek struktury má být v programu před koncem struktury. Chybu např. vyvolá:

Př. 66:

```
10 GOTO 30
20 NEXT I:GOTO 40
30 FOR I=1 TO 5: I:GOTO 20
40 REM
```

c) Nepoužívat proměnné jako parametry v příkazech DIM, COM, GOTO, TRAP, THEN, RESTORE a GOSUB, neboť překladač potřebuje u těchto příkazů konstantu, konkrétní číslo:

Př. 67:

```
10 DIM A(N)
20 CR=100:GOTO CR
```

d) Vyvarovat se křížení struktur:

Př. 68:

```
10 FOR I=1 TO 10 <-----+
20   DO <-----!-+
30   NEXT I <-----+ !
40   IF I=10 THEN EXIT !
50 LOOP <-----+ !
```

Chcete-li se vyhnout mnoha nepříjemnostem, snažte se nepoužívat nepodmíněné skoky GOTO a příkaz GOSUB, tj. nestrukturované příkazy. U TB lze téměř vždy tyto příkazy nahradit

konstrukcemi s IF/ELSE/ENDIF, WHILE/WEND, REPEAT/UNTIL, DO/EXIT/LOOP, PROC/ENDPROC, případně GOTO.

## 2. Použití nepřeložitelného příkazu

Existuje řada příkazů, které CTB nepřeloží. Jsou to následující příkazy:

LIST, ENTER, \*L, DEL, RENUM, DUMP, TRACE, CONT, LOAD, SAVE, CLOAD, CSAVE a NEW.

Jedná se vesměs tedy o příkazy operující s čísly řádků a s proměnnými (tabulkami proměnných), které v přeloženém programu být nemohou, neboť ten již nemá při spuštění k dispozici interpret TB. ERROR se objeví v programu, pokud odešlete syntakticky špatný řádek programu v Basicu. Takový řádek také samozřejmě nelze přeložit.

Jediným řešením k odstranění chyb způsobených těmito příkazy je nepoužívat je.

### Poznámka recenzenta:

Polský časopis Bajtek uveřejnil v čísle 10 z roku 1988 na straně 8 článkeček o zjištěné chybě v CTB. V kompilátoru nefunguje správně funkce DEC; počítá asi takto:

```
A=DEC(A$) ...
```

```
A=LEN(A$):A=128+256*A*(A<5)+1024*(A>4)
```

Celý efekt je způsoben tím, že kompilátor zapomíná přidat sestavu

```
A5 D4 LDA $D4
```

```
A4 D5 LDY $D5
```

do kompilovaného programu. Protože oprava kompilátoru by byla obtížná, byl zveřejněn krátký prográmeček, který modifikuje RUNTIME. Aby nenastala chyba, program udělá kopii na disk pod názvem RUNTIME2.EXE. Pro kazetovou verzi však tento prográmeček nelze použít.

V dalším je uveden výpis tohoto programu:

```

5 GRAPHICS 0
10 -----
15 TRAP 15:? :? "WLOZ DYSKIETKE ZAWIER
AJACA ZBIOR"? ""D1:RUNTIME.EXE"", WC
ISNIJ /RETURN/";GET X
20 CLOSE :OPEN #1,4,0,"D:RUNTIME.EXE"
25 BGET #1,$4000,$2A7C:CLOSE
30 -----
35 POKE $4EBC,$C5:DPOKE $5E92,$FFBE
40 NPOKE $6A7C,$20:DPOKE $6A7E,$A5C8
45 DPOKE $6A80,$A4D4:DPOKE $6A82,$60D5
50 -----
55 TRAP 75:? :? "PRZYGOTUJ DYSKIETKE,
WCISNIJ /RETURN/";GET X
60 OPEN #1,0,0,"D:RUNTIME2.EXE"
65 BPUT #1,$4000,$2A84:CLOSE :END
70 -----
75 CLOSE :? :? "BLAD ";ERR:GOTO 55

```

### 3.12 DOS a TB

TB existuje jak v kazetové, tak v disketové verzi. S disketovou verzí lze samozřejmě pracovat pouze pod DOSem. TB XL 1.5 a CTB mohou pracovat například pod DOSem 2.0, DOSem 2.5, DOSem 4.2 (MY-DOSem), OS/A+ (DOSem XL) a jinými příbuznými diskovými operačními systémy. TB nelze používat pod DOSem 3.0 a pod SpartaDOSem!

Mnoho systémových programů je vytvořeno tak, že po načtení do počítače testují, zda je na disketě uložen soubor s určitým názvem. Pokud je tomu tak, je načten tento soubor a obvykle automaticky spuštěn. Tak je tomu i u DOSu, TB a CTB.

DOS hledá a automaticky spouští soubor s názvem AUTORUN.SYS. Máte-li tedy pod tímto názvem uložen na disketě s DOSem Turbo-BASIC, automaticky se načte a spustí. Není-li soubor AUTORUN.SYS na disketě přítomen, objeví se DOS-MENU.

TB hledá a spouští soubor AUTORUN.BAS, což musí být program napsaný v TB nebo v AB. Není-li tento program na disketě, vypíše se READY a řízení je předáno Basic-editoru.

Program runtime hledá a spouští program s názvem AUTORUN.CTB. Pokud ho nenajde, ohlásí chybu (viz kap. 3.11).

Z TB lze stejně jako z AB přejít do DOS odesláním příkazu DOS, přičemž se do paměti počítače načte program DUP.SYS a objeví se DOS-MENU.

Pokud nemáte na disketě (nebo v RAMdisku), odkud je čten DUP.SYS, soubor MEM.SAV, přepíše se nenávratně ta část paměti, odkud se ukládá DUP.SYS. Tak přijdete o svůj program v paměti počítače. U AB se však můžete vrátit do Basicu zvolením B (RUN CARTRIDGE). Toto v TB nelze, neboť TB vypíná vestavěnou Basic-cartridge, pokud ji nepotřebuje. U TB DOS vypíše NO CARTRIDGE, čili nezbyde nic jiného, než znovu načíst TB ze zdrojové diskety.

Pokud je na disketě s DOsem i soubor MEM.SAV (nesmí být chráněn!), zapíše se část paměti přepisovaná daty ze souboru DUP.SYS do tohoto souboru a při zadání B v DOS-MENU se přenesou zase zpět z tohoto souboru do počítače. Program tedy zůstane v paměti počítače i po návratu z DOSu, který je za těchto okolností možný i při práci s TB pomocí skoku na adresu \$2000 (odpovídá volbě M /RETURN/ 2000 /RETURN/ z DOSu).

Nevýhodou v tomto případě je, že se při každém volání DOSu ukládá na disketu a při návratu do Basicu zase do paměti počítače téměř 6 kB dat, což trvá zhruba 25-30 sekund při volání DOSu a asi 7 sekund při návratu do Basicu (vyjma případu, kdy je využíván RAMdisk u ATARI 130XE). Navíc soubor MEM.SAV zabírá 45 sektorů.

Poznámka recenzenta: Výše popsaná metoda je použitelná pod DOSy 2.x. Některé DOSy však umožňují startovat TB i bez pomocného souboru MEM.SAV (umí to BIBM-DOS, OS/A+, Action! DOS, Happy DOS II+/D) přímo přes RUN \$2000.

Soubor MEM.SAV lze vytvořit přímo z DOSu (volba N v DOS-MENU) nebo z Basicu. Druhého způsobu lze s výhodou využít, potřebujeme-li nutně do DOSu, na disketě nemáme MEM.SAV a nechceme či nemůžeme uložit program, který máme v paměti počítače a o který nechceme přijít. Proveďte se to následujícím způsobem:

```
OPEN #1, 0, 0, "D:MEM.SAV"; CLOSE #1: DOS
```

Tato posloupnost příkazů vytvoří soubor MEM.SAV a zavolá DOS. Úbpec nevedí, že takto vytvořený MEM.SAV nemá původně 45 sektorů, důležité je pouze, aby těchto 45 sektorů bylo na disketě volných. Návrat do TB je nutné provést nikoliv pomocí volby B, ale pomocí M (RUN AT ADDRESS), a to od adresy \$2000 (znak \$ se v DOS-MENU nezadává!).

#### 4. R A D Y A N Ā P A D Y

a) Jak vymazat staré proměnné z tabulky jmen a názvů proměnných?

Tento úkon se provádí ve dvou krocích:

1. LOAD "D:PRG":LIST "D:QQ":NEW
2. ENTER "D:QQ":SAVE "D:PRG":DELETE "D:QQ"

U kazetového magnetofonu se postupuje obdobným způsobem:

1. CLOAD
2. LIST "C:"
3. NEW
4. ENTER "C:"
5. CSAVE

b) Měření rychlosti vykonávání příkazu nebo části programu

Nejprve napište tento program

```
1000 B=B-1:IF B>0 THEN 1000
1010 A=TIME
1020 A=A*20/C
1030 ? "1 cyklus trval ";A;" ms"
1040 END
1234 CLR:INPUT "pocet cyklu =",B:C=B
1300 TIME$="000000":GOTO 1000
```

a spusíte jej příkazem GOTO 1234 (stačí G.1234). Pak zadejte počet opakování cyklu (alespoň 500). Tím zjistíte, jak dlouho trvá na Vašem počítači prázdný cyklus tohoto programu. Tuto mrtvou dobu je zapotřebí vždy odečíst, Změřenou hodnotu tedy odečtete od výrazu na řádce 1020, např.:

```
1020 A=A*20/C-2.66
```

Tuto podobu programu doporučuji uložit. Nyní napište na řádky 0 až 999 Váš program, který chcete zkoumat, a přepište na řádcích 1000 a 1300 číslo 1000 za THEN (GOTO) na číslo prvního řádku Vašeho programu, např. na nulu. Nyní už jen zbývá spustit program opět příkazem G.1234 a zadat přiměřený počet cyklů (zadáte-li příliš málo opakování, výsledek nemusí být přesný, zadáte-li příliš mnoho cyklů, budete na výsledek dlouho čekat). Po skončení výpočtu můžete měřené řádky měnit a zkoumat dále.

Program může být užitečný při optimalizaci programových úseků, dostanete-li se při výpočtech do časové tísně apod.

V dalším jsou uvedeny doby vykonávání některých operací:

```
REM <0.01 ms>  
Q=PEEK(Q) <1.44 ms>  
FOR Q=1 TO 10:NEXT Q <8.28 ms>  
Q=1:REPEAT :Q=Q+1:UNTIL Q=10 <20.96 ms>  
Q=1:WHILE Q<10:Q=Q+1:WEND <23.2 ms>  
Q=INT(Q/256) <1.72 ms>  
Q=Q DIV 256 <1.4 ms>
```

c) Chcete-li po napsání a uložení programu v TB program zkompilovat, není potřeba při načítání CTB přecházet do DOSu. Kompilátor lze přímo z TB nahrát a spustit pomocí příkazu BLOAD "D:COMPILER.COM".

d) Pomocí běžných diskových editorů lze celkem bez problémů přepsat v Turbo-BASICu (chápejme v souboru) hlášení 'READY' například za hlášení 'TURBO'. Tato úprava pomůže uživateli na první pohled rozlišit, zda má pro svou práci připraven "v počítači" AB nebo TB.

## 5. D O D A T K Y A T A B U L K Y

Od roku 1988 je k dispozici kazetová verze TB - Turbo-BASIC XL 2.0, rozšířená o možnost ukládání dat ve formátu TURBO 2000. V této verzi je využito příkazu DOS, který pro operace s kazetovým magnetofonem postrádá smysl. Po odeslání tohoto příkazu se na obrazovce objeví následující menu:

LOAD SAVE QUIT VERIFY

Význam jednotlivých příkazů:

LOAD načtení programu do počítače  
SAVE zápis programu na kazetu  
QUIT návrat do TB editoru  
VERIFY kontrola správnosti nahrávky

### a) Čtení programu (LOAD)

Stiskněte L (ozve se 1 tón), /RETURN/ a na kazetovém magnetofonu tlačítko /PLAY/. Počítač hledá hlavičku programu v TB. Program na kazetě musí být uložen systémem TURBO 2000 pomocí této verze TB nebo jiným slučitelným způsobem. Po nalezení hlavičky se záznamový magnetofon zastaví. Nyní můžete zvolený program načíst stisknutím klávesy /RETURN/.

### b) Zápis programu na kazetu (SAVE)

Stiskněte S, vypíše se ; PROGRAMNAME:. Napište název programu (maximálně 10 znaků) a odešlete klávesou /RETURN/. Je-li první znak názvu \*, program se po načtení LOADem sám spustí. Po ukončení zápisu můžete požadovat ještě jednu kopii (stiskněte /RETURN/) nebo přejít do menu (stiskněte /BREAK/).

### c) Návrat zpět do TB editoru (QUIT)

Stiskněte klávesu Q.

### d) Kontrola správnosti nahrávky (VERIFY)

Stiskněte V a přetočte pásku na začátek programu. Pak stiskněte /RETURN/ a tlačítko /PLAY/. Až se načte hlavička, stiskněte ještě jednou /RETURN/. Pokud je program v pořádku, vypíše se O.K., v opačném případě počítač ohlásí chybu - BOOT ERROR. V tom případě program uložte znovu příkazem SAVE. Pozor: VERIFY je možno provést pouze bezprostředně po uložení programu (SAVE)!

Turbo-BASIC XL 2.0 má nadefinované nové zařízení, I:, které pracuje jako standardní IOCB, tedy je možné používat následujících

```
LOAD "T:jmeno"  
SAVE "T:jmeno"  
RUN "T:jmeno"  
LIST "T:jmeno"  
ENTER "T:jmeno"
```

Podobně lze také pracovat s daty:

```
OPEN #1, 8, 0, "T:jmeno"  
FOR I=0 TO 255  
PUT #1, I  
NEXT I  
CLOSE #1
```

Poznámka:

Soubory jsou přenositelné mezi TB XL 2.0 a TOS XL verze 2.01 a vyšší (TOS 4.0 a vyšší, CTOS 1.0 a vyšší), nejsou ale přenositelné mezi TB XL 2.0 a TTDOS všech verzí.

Máte-li disketovou jednotku, popřípadě používáte-li TTDOS, můžete k zajištění přenosu dat mezi diskem a zařízením "T:" použít i novou verzi TB XL 2.0 s názvem TTBASIC 2.1, kterou nabízí softwareová skupina ATARI klubu Praha. Ta umožňuje instalaci zařízení "T:" bez konfliktů s DOSem.

Pozor! Základní verzi TB XL 2.0 není možné spustit z DOSu ani z TOSu.

Tabulka not pro příkazy SOUND a DSOUND

oktáva	nota	DSOUND	SOUND
1	C	27357	
	C#	25821	
	D	24372	
	D#	23003	
	E	21712	
	F	20493	
	F#	19342	
	G	18256	
	G#	17231	
	A	16264	
A#	15351		
H	14489		
2	C	13675	
	C#	12907	
	D	12182	
	D#	11498	
	E	10852	
	F	10243	
	F#	9668	
	G	9125	
	G#	8612	
	A	8128	
A#	7672		
H	7241		
3	C	6834	243
	C#	6458	238
	D	6088	217
	D#	5746	204
	E	5423	193
	F	5118	182
	F#	4838	172
	G	4559	162
	G#	4303	153
	A	4061	144
A#	3832	136	
H	3617	128	
4	C	3414	121
	C#	3222	114
	D	3048	108
	D#	2869	102
	E	2708	96
	F	2555	91
	F#	2412	85
	G	2276	81
	G#	2148	76
	A	2027	72
A#	1913	68	
H	1805	64	

oktáva	nota	DSOUND	SOUND
5	C	1703	60
	C#	1607	57
	D	1517	53
	D#	1431	50
	E	1350	47
	F	1274	45
	F#	1202	42
	G	1134	40
	G#	1070	37
	A	1010	35
	A#	953	33
H	899	31	
6	C	848	30
	C#	800	28
	D	755	26
	D#	712	25
	E	672	23
	F	634	22
	F#	596	21
	G	564	19
	G#	532	18
	A	501	17
	A#	473	16
H	446	15	
7	C	421	14
	C#	397	
	D	374	
	D#	353	
	E	332	
	F	313	
	F#	295	
	G	276	
	G#	262	
	A	247	
	A#	233	
H	219		
8	C	207	
	C#	195	
	D	183	
	D#	173	
	E	163	
	F	153	
	F#	144	
	G	136	
	G#	128	
	A	120	
	A#	113	
H	106		

Použití: např. C5 = DSOUND A,1703,10,D

Tabulka grafických módů

kód ANTIC	2	3	4	5	6
kód BASIC					
bez text. o. →	8	-	28	29	17
text. okénko →	-	-	12	13	1
graf.-text.	T	T	T	T	T
počet barev	2	2	5	5	5
svisle →	8	10	8	16	8
rozměr Pix.	x	x	x	x	x
vodorovně →	8	8	8	8	16
počet řádků na obrazovku bez text. o. →	24	19	24	12	24
text. okénko →	-	-	20	10	20
počet sloupců na řádek	40	40	40	40	20
počet bytů na řádek	40	40	40	40	20
počet bytů na obrazovku bez text. o. →	960	760	960	480	480
text. okénko →	-	-	800	400	400
spotřeba pa- měti (i s DL) bez text. o. →	992	787	1152	668	672
text. okénko →	-	-	1154	664	674

kód ANTIC	7	8	9	A	B
kód BASIC					
bez text. o. →	18	19	20	21	22
text. okénko →	2	3	4	5	6
graf.-text.	T	G	G	G	G
počet barev	5	4	2	4	2
svisle →	16	8	4	4	2
rozměr Pix.	x	x	x	x	x
vodorovně →	16	8	4	4	2
počet řádků na obrazovku					
bez text. o. →	12	24	48	48	96
text. okénko →	10	20	40	40	80
počet sloupců na řádek	20	40	80	80	160
počet bytů na řádek	20	10	10	20	20
počet bytů na obrazovku					
bez text. o. →	240	240	480	960	1920
text. okénko →	200	200	400	800	1600
spotřeba pa- měti ( $i \leq DL$ )					
bez. text o. →	420	432	696	1176	2184
text. okénko →	424	434	694	1174	2174

kód ANTIC	C	D	E	F	F
kód BASIC					
bez text. o.→	30	23	31	24	9
text. okénko→	14	7	15	8	-
sraf.-text.	G	G	G	G	G
počet barev	2	4	4	1/2	1/16
svisle →	1	2	1	1	1
rozměr Pix.	x	x	x	x	x
vodorovně →	2	2	2	1	4
počet řádků na obrazovku					
bez text. o.→	192	96	192	192	192
text. okénko→	160	80	160	160	160
počet sloupců na řádek	160	160	160	320	80
počet bytů na řádek	20	40	40	40	40
počet bytů na obrazovku					
bez text. o.→	3840	3840	7680	7680	7680
text. okénko→	3200	3200	6400	6400	6400
spotřeba pa- měti (i s DL)					
bez text. o.→	4296	4200	8138	8138	8138
text. okénko→	4270	4190	8112	8112	-

kód ANTIC	F	F			
kód BASIC					
bez text. o. →	26	27			
text. okénko →	10	11			
graf.-text.	G	G			
počet barev	9	16			
svisle →	1	1			
rozměr Pix.	x	x			
vodorovně →	4	4			
počet řádků na obrazovku					
bez text. o. →	192	192			
text. okénko →	160	160			
počet sloupců na řádek	80	80			
počet bytů na řádek	40	40			
počet bytů na obrazovku					
bez text. o. →	7680	7680			
text. okénko →	6400	6400			
spotřeba pa- měti (i s DL)					
bez text. o. →	8136	8136			
text. okénko →	-	-			

Poznámka: Rozlišení GRAPHICS 8, 9, 10 a 11 je provedeno mimo DL.

Tabulka barev

barva	číslo barvy (2. parametr příkazu SETCOLOR)	hodnota pro příkaz POKE
šedá	0	0
světle oranžová (zlatá)	1	16
oranžová	2	32
červenooranžová (světle červená)	3	48
růžová	4	64
nachová	5	80
nachově modrá	6	96
azurově modrá	7	112
blankytně modrá	8	128
světle modrá	9	144
tyrkysová	10	160
zelenomodrá	11	176
zelená	12	192
žlutozelená	13	208
oranžovozelená	14	224
světle oranžová	15	240

Tabulka paměti PMG

dvojitě	nastavené řádkování	jednoduché
vzdálenost od začátku PMG paměti		
0	nepoužitá paměť	0
+ 384	střely 0   1   2   3	+ 768
+ 512	hráč 0	+ 1024
+ 640	hráč 1	+ 1280
+ 768	hráč 2	+ 1536
+ 896	hráč 3	+ 1792
+ 1023		+ 2047

## Tabulka chybových hlášení TB

### Chyby programové:

číslo chyby	označení chyby	popis chyby
2	MEM	Nedostatek paměti. Program přesahuje kapacitu paměti.
3	VALUE	Nesprávná hodnota. Číselná hodnota je větší než maximální možná nebo je záporná tam, kde musí být kladná.
4	>#VARS	Příliš mnoho proměnných a jmen. V AB lze použít maximálně 128 proměnných, v TB maximálně 256 proměnných a jmen v jednom programu.
5	\$LEN	Chyba v délce řetězce. Pokus o použití části řetězce, která leží mimo deklaraci. Např. po DIM A\$(5).
6	?DATA	K dispozici je méně dat, než kolik jich bylo čteno příkazem READ.
7	>32767	Číslo větší než 32767 tam, kde tomu tak být nesmí.
8	INPUT	Chybný vstup dat. Neodpovídající proměnná po příkazu INPUT a vložená hodnota. Např. vložíme-li po INPUT A řetězec.
9	DIM	Chyba v deklaraci. Pokus o deklaraci již deklarovaného pole, deklaraci pole (ř) zaujímajícího více než 32767 bytů nebo pokus o použití dosud nedeklarovaného pole (ř). Např. ? A\$, nepředcházel-li příkaz DIM A\$(5).

číslo chyby	označení chyby	popis chyby
10	STACK	Přetečení zásobníku pro výrazy, výraz je příliš složitý.
11	OVERFLOW	Přetečení FP registru. Pokus o dělení nulou nebo se vyskytlo číslo, jehož absolutní hodnota je větší než 9.99999999E105.
12	LINE	Řádek nenalezen. Bylo použito číslo řádku, který se v programu nevyskytuje.
13	?FOR	K NEXT chybí FOR. Interpret narazil na příkaz NEXT n, kterému nepředcházel příkaz FOR n=.... Tato chyba vzniká při skoku do smyčky, či pokud chybí příslušný příkaz FOR n=....
14	TOO LONG	Řádek delší než 256 bytů, což je maximální délka řádku v TB.
15	?DEL	Byl vymazán příkaz GOSUB k odpovídajícímu RETURN, NEXT k FOR, REPEAT k UNTIL, ....., takže návrat (podprogramu ap.) na něj není možný.
16	?GOSUB	K RETURN chybí GOSUB. Obdobu chyby č. 13.
17	GARBAGE	Výrok není proveditelný. Jde o syntaktickou chybu v programu vzniklou buď odesláním syntakticky nesprávného řádku (s hlášením ERROR - ...), nebo změnou vlastního programu příkazem POKE či programem ve strojovém kódu.
18	?CHR	Nesprávný znak. Došlo k použití funkce VAL na řetězec, v němž se vyskytuje nepřevoditelný znak.

číslo chyby	označení chyby	Popis chyby
19	MEM	Načítaný program je příliš dlouhý.
20	#	Špatné číslo kanálu. Čísla kanálů jsou 0 až 7, používat lze i čísla zvětšená o celé násobky 16.
21	?LOAD	Soubor nelze načíst příkazem LOAD. Zřejmě jde o program ve strojovém kódu nebo data.
22	?NEST	K WHILE není nalezeno odpovídající WEND, k IF ENDIF, ... nebo po *F+ není k FOR n=... nalezeno NEXT, má-li se smyčka přeskočit. Obvykle jde o vnoření cyklů.
23	?WHILE	K UNTIL chybí REPEAT. Obdoba chyby č. 13.
24	?REPEAT	K WEND chybí WHILE. Obdoba chyby č. 13.
25	?DO	K LOOP chybí DO. Obdoba chyby č. 13.
26	?EXIT	Příkaz EXIT byl použit mimo cyklus.
27	XPROC	Procedura nebyla vyvolána příkazem EXEC.
28	?EXEC	K ENDPROC chybí EXEC. Obdoba chyby č. 13.
29	?PROC	Byla volána neznámá procedura, procedura tohoto jména není v programu deklarována.
30	?#	Volané návěští není definováno.

Poznámka: Předkládaný výpis chybových hlášení TB není úplný; s významem chybějících kódů se běžný uživatel buď vůbec nesetká, nebo mu není znám.

Chyby vzniklé při I/O operacích:

číslo chyby	označení chyby	Popis chyby
128		Přerušeni klávesou /BREAK/ během I/O operace.
129	IS OPEN	Kanál již otevřen. Pozn.: V grafických módech je již otevřen kanál číslo 6 na obrazovku, kanál číslo 7 bývá obvykle otevřen na klá- vesnici, kanál číslo 8 na editor.
130	?DEV	Voláno neznámé zařízení, není definováno v HATABAS.
131	WR ONLY	Kanál je otevřen pouze pro zápis.
132	CMD	Chybný I/O příkaz.
133	NOT OPEN	Kanál není otevřen.
134	##	Chybné číslo IOCB kanálu. (v TB nemůže nastat).
135	RD ONLY	Kanál je otevřen pouze pro čtení.
136	EOF	Pokus o čtení souboru poté, co bylo dosaženo jeho konce.
137	TRUNC	Nahrávka delší než buffer. Objevuje se například při čtení dat příkazem ENTER, když byla uložena příkazem SAVE.
138	TIME OUT	Zařízení neodpovídá. Vypršel čas, během kterého se mělo zařízení ohlásit Pravděpodobně není zapnuté nebo je chyba v připojení.
139	NAK	Neproveditelný příkaz nebo zařízení nepracuje bez závad. Vzniká například při pokusu o přístup do sektoru 0. Zkuste zopakovat příkaz, případně zkontrolovat při- pojení periférií.

číslo chyby	označení chyby	Popis chyby
140	!FRAME	Chyba na sériové sběrnici.
141	CURSOR	Kursor mimo rozsah v daném grafickém režimu.
142	!OVERRUN	Špatný formát při přenosu dat po sériové sběrnici.
143	CHKSUM	Chyba parity (kontrolního součtu) při přenosu dat po sériové sběrnici. Kazeta nebo disketa je pravděpodobně poškozená.
144	DONE	Chyba diskety. Disketa je chráněna proti zápisu, není v disketové jednotce nebo byl čten vadný sektor.
145	MODE	Špatný mód zobrazování nebo chyba při zápisu na disketu.
146	NOT IMPL	Funkce není proveditelná. Pokus provést neproveditelnou funkci, například zavést data na klávesnici, číst data z tiskárny, a pod.
147	RAM	Pro požadovaný grafický mód je nedostatek volné RAM.
160	D?:	Špatné číslo disketové jednotky.
161	>#FILES	Otevřeno více souborů, než je konfigurováno DOSem.
162	DSK FULL	Disketa je plná.
163	FATAL I/O	Neidentifikovatelná systémová chyba při I/O operaci.
164	FILE#	Chyba v čísle souboru. Pravděpodobně jsou porušena spojení mezi jednotlivými sektory souboru (link).
165	NAME	Chybný název souboru.
166	POINT	Příkaz POINT chybný. Pokus využít příkaz POINT pro neexistující byte.

číslo chyby	označení chyby	Popis chyby
167	LOCKED	Soubor ochráněn. Pokus smazat, přepsat, či otevřít pro zápis ochráněný soubor.
168	DCMD	Špatný příkaz pro dané zařízení.
169	>DIR	Prostor pro Seznam souborů na disketě je zaplněn. Na jedné straně diskety může být v DOS 2.5 maximálně 64 souborů.
170	?FILE	Soubor nenalezen.
171	POINT	Příkaz POINT nelze provést Daný sektor nenáleží k otevřenému souboru.
173	BAD SECTORS	Vadný sektor. Například není vložena disketa při formátování.

Poznámka: Ustupně/výstupní chybová hlášení jsou závislá na typu používaného DOSu.

Tabulka kódů a čísel

DEC	HEX	BIN	ATASCII	Intern	komb. kl.			
0	\$00	%00000000	␣	:	á	CTL-	,	
1	\$01	%00000001	␣	:	â	CTL-	A	
2	\$02	%00000010	␣	:	ã	CTL-	B	
3	\$03	%00000011	␣	:	ä	CTL-	C	
4	\$04	%00000100	␣	:	å	CTL-	D	
5	\$05	%00000101	␣	:	æ	CTL-	E	
6	\$06	%00000110	␣	:	ç	CTL-	F	
7	\$07	%00000111	␣	:	¸	CTL-	G	
8	\$08	%00001000	␣	:	é	CTL-	H	
9	\$09	%00001001	␣	:	ê	CTL-	I	
10	\$0A	%00001010	␣	:	ë	CTL-	J	
11	\$0B	%00001011	␣	:	¸	CTL-	K	
12	\$0C	%00001100	␣	:	¸	CTL-	L	
13	\$0D	%00001101	␣	:	¸	CTL-	M	
14	\$0E	%00001110	␣	:	¸	CTL-	N	
15	\$0F	%00001111	␣	:	¸	CTL-	O	
16	\$10	%00010000	␣	:	ü	0	CTL-	P
17	\$11	%00010001	␣	:	û	1	CTL-	Q
18	\$12	%00010010	␣	:	ü	2	CTL-	R
19	\$13	%00010011	␣	:	ü	3	CTL-	S
20	\$14	%00010100	␣	:	¸	4	CTL-	T
21	\$15	%00010101	␣	:	¸	5	CTL-	U
22	\$16	%00010110	␣	:	¸	6	CTL-	V
23	\$17	%00010111	␣	:	¸	7	CTL-	W
24	\$18	%00011000	␣	:	¸	8	CTL-	X
25	\$19	%00011001	␣	:	¸	9	CTL-	Y
26	\$1A	%00011010	␣	:	¸		CTL-	Z
27	\$1B	%00011011	␣	:	¸		ES/ES	
28	\$1C	%00011100	␣	:	¸	<	ES/CTL-	-
29	\$1D	%00011101	␣	:	¸	=	ES/CTL-	=
30	\$1E	%00011110	␣	:	¸	>	ES/CTL-	+
31	\$1F	%00011111	␣	:	¸	?	ES/CTL-	*
32	\$20	%00100000			@		SPC	
33	\$21	%00100001	!		A		SH-	1
34	\$22	%00100010	"		B		SH-	2
35	\$23	%00100011	#		C		SH-	3
36	\$24	%00100100	\$		D		SH-	4
37	\$25	%00100101	%		E		SH-	5
38	\$26	%00100110	&		F		SH-	6
39	\$27	%00100111	'		G		SH-	7
40	\$28	%00101000	(		H		SH-	9
41	\$29	%00101001	)		I		SH-	0
42	\$2A	%00101010	*		J		*	
43	\$2B	%00101011	+		K		+	
44	\$2C	%00101100	,		L		,	
45	\$2D	%00101101	-		M		-	
46	\$2E	%00101110	.		N		.	
47	\$2F	%00101111	/		O		/	

Tabulka kódů a čísel

DEC	HEX	BIN	ATASCII	Intern	komb. kl.
48	\$30	%00110000	0	P	0
49	\$31	%00110001	1	Q	1
50	\$32	%00110010	2	R	2
51	\$33	%00110011	3	S	3
52	\$34	%00110100	4	T	4
53	\$35	%00110101	5	U	5
54	\$36	%00110110	6	V	6
55	\$37	%00110111	7	W	7
56	\$38	%00111000	8	X	8
57	\$39	%00111001	9	Y	9
58	\$3A	%00111010	:	Z	SH- ;
59	\$3B	%00111011	;	[	;
60	\$3C	%00111100	<	\	<
61	\$3D	%00111101	=	^	=
62	\$3E	%00111110	>	^	>
63	\$3F	%00111111	?	_	SH- /
<hr/>					
64	\$40	%01000000	a	Ⓜ	SH- 0
65	\$41	%01000001	A	Ⓜ	A
66	\$42	%01000010	B	Ⓜ	B
67	\$43	%01000011	C	Ⓜ	C
68	\$44	%01000100	D	Ⓜ	D
69	\$45	%01000101	E	Ⓜ	E
70	\$46	%01000110	F	Ⓜ	F
71	\$47	%01000111	G	Ⓜ	G
72	\$48	%01001000	H	Ⓜ	H
73	\$49	%01001001	I	Ⓜ	I
74	\$4A	%01001010	J	Ⓜ	J
75	\$4B	%01001011	K	Ⓜ	K
76	\$4C	%01001100	L	Ⓜ	L
77	\$4D	%01001101	M	Ⓜ	M
78	\$4E	%01001110	N	Ⓜ	N
79	\$4F	%01001111	O	Ⓜ	O
<hr/>					
80	\$50	%01010000	P	Ⓜ	P
81	\$51	%01010001	Q	Ⓜ	Q
82	\$52	%01010010	R	Ⓜ	R
83	\$53	%01010011	S	Ⓜ	S
84	\$54	%01010100	T	Ⓜ	T
85	\$55	%01010101	U	Ⓜ	U
86	\$56	%01010110	V	Ⓜ	V
87	\$57	%01010111	W	Ⓜ	W
88	\$58	%01011000	X	Ⓜ	X
89	\$59	%01011001	Y	Ⓜ	Y
90	\$5A	%01011010	Z	Ⓜ	Z
91	\$5B	%01011011	[	Ⓜ	SH- ,
92	\$5C	%01011100	\	Ⓜ	SH- +
93	\$5D	%01011101	]	Ⓜ	SH- .
94	\$5E	%01011110	^	Ⓜ	SH- *
95	\$5F	%01011111	_	Ⓜ	SH- -

Tabulka kódů a čísel

DEC	HEX	BIN	ATASCII	Intern	komb. kl.
96	\$60	%01100000	♦ : i	♦ : i	CTL- .
97	\$61	%01100001	a	a	A
98	\$62	%01100010	b	b	B
99	\$63	%01100011	c	c	C
100	\$64	%01100100	d	d	D
101	\$65	%01100101	e	e	E
102	\$66	%01100110	f	f	F
103	\$67	%01100111	g	g	G
104	\$68	%01101000	h	h	H
105	\$69	%01101001	i	i	I
106	\$6A	%01101010	j	j	J
107	\$6B	%01101011	k	k	K
108	\$6C	%01101100	l	l	L
109	\$6D	%01101101	m	m	M
110	\$6E	%01101110	n	n	N
111	\$6F	%01101111	o	o	O
112	\$70	%01110000	P	P	P
113	\$71	%01110001	q	q	Q
114	\$72	%01110010	r	r	R
115	\$73	%01110011	s	s	S
116	\$74	%01110100	t	t	T
117	\$75	%01110101	u	u	U
118	\$76	%01110110	v	v	V
119	\$77	%01110111	w	w	W
120	\$78	%01111000	x	x	X
121	\$79	%01111001	y	y	Y
122	\$7A	%01111010	z	z	Z
123	\$7B	%01111011	♦	♦	CTL- ;
124	\$7C	%01111100			SH- =
125	\$7D	%01111101	↵	↵	ES/SH- <
126	\$7E	%01111110	⏪	⏪	ES/BKS
127	\$7F	%01111111	⏩	⏩	ES/TAB
128	\$80	%10000000	[␣]: [á]	[ ]	␣/CTL- ,
129	\$81	%10000001	[␣]: [à]	[!]	␣/CTL- A
130	\$82	%10000010	[␣]: [ã]	["]	␣/CTL- B
131	\$83	%10000011	[␣]: [ä]	[#]	␣/CTL- C
132	\$84	%10000100	[␣]: [å]	[\$]	␣/CTL- D
133	\$85	%10000101	[␣]: [æ]	[%]	␣/CTL- E
134	\$86	%10000110	[␣]: [ç]	[&]	␣/CTL- F
135	\$87	%10000111	[␣]: [d]	[']	␣/CTL- G
136	\$88	%10001000	[␣]: [e]	[ ( ]	␣/CTL- H
137	\$89	%10001001	[␣]: [f]	[ ) ]	␣/CTL- I
138	\$8A	%10001010	[␣]: [g]	[ * ]	␣/CTL- J
139	\$8B	%10001011	[␣]: [h]	[ + ]	␣/CTL- K
140	\$8C	%10001100	[␣]: [i]	[ , ]	␣/CTL- L
141	\$8D	%10001101	[␣]: [j]	[ - ]	␣/CTL- M
142	\$8E	%10001110	[␣]: [k]	[ . ]	␣/CTL- N
143	\$8F	%10001111	[␣]: [l]	[ / ]	␣/CTL- O

Tabulka kódů a čísel

DEC	HEX	BIN	ATASCII	Intern	komb. kl.
144	\$90	%10010000	[+]: [ü]	[0]	☐/CTL- P
145	\$91	%10010001	[r]: [á]	[1]	☐/CTL- Q
146	\$92	%10010010	[~]: [ú]	[2]	☐/CTL- R
147	\$93	%10010011	[+]: [í]	[3]	☐/CTL- S
148	\$94	%10010100	[0]: [é]	[4]	☐/CTL- T
149	\$95	%10010101	[m]: [è]	[5]	☐/CTL- U
150	\$96	%10010110	[ ]]: [á]	[6]	☐/CTL- V
151	\$97	%10010111	[r]: [é]	[7]	☐/CTL- W
152	\$98	%10011000	[+]: [à]	[8]	☐/CTL- X
153	\$99	%10011001	[# ]]: [à]	[9]	☐/CTL- Y
154	\$9A	%10011010	[+]: [á]	[:]	☐/CTL- Z
155	\$9B	%10011011	[é]	[;]	RETURN
156	\$9C	%10011100	[+]	[<]	ES/SH-BKS
157	\$9D	%10011101	[+]	[=]	ES/SH- >
158	\$9E	%10011110	[<]	[>]	S/CTL-TAB
159	\$9F	%10011111	[+]	[?]	ES/SH-TAB
160	\$A0	%10100000	[ ]	[a]	☐/SPC
161	\$A1	%10100001	[!]	[A]	☐/SH- 1
162	\$A2	%10100010	["]	[B]	☐/SH- 2
163	\$A3	%10100011	[#]	[C]	☐/SH- 3
164	\$A4	%10100100	[\$]	[D]	☐/SH- 4
165	\$A5	%10100101	[%]	[E]	☐/SH- 5
166	\$A6	%10100110	[&]	[F]	☐/SH- 6
167	\$A7	%10100111	[']	[G]	☐/SH- 7
168	\$A8	%10101000	[ ( ]	[H]	☐/SH- 9
169	\$A9	%10101001	[ ) ]	[I]	☐/SH- 8
170	\$AA	%10101010	[*]	[J]	☐/ *
171	\$AB	%10101011	[+]	[K]	☐/ +
172	\$AC	%10101100	[, ]	[L]	☐/ ,
173	\$AD	%10101101	[-]	[M]	☐/ -
174	\$AE	%10101110	[. ]	[N]	☐/ .
175	\$AF	%10101111	[/]	[O]	☐/ /
176	\$B0	%10110000	[0]	[P]	☐/ 0
177	\$B1	%10110001	[1]	[Q]	☐/ 1
178	\$B2	%10110010	[2]	[R]	☐/ 2
179	\$B3	%10110011	[3]	[S]	☐/ 3
180	\$B4	%10110100	[4]	[T]	☐/ 4
181	\$B5	%10110101	[5]	[U]	☐/ 5
182	\$B6	%10110110	[6]	[V]	☐/ 6
183	\$B7	%10110111	[7]	[W]	☐/ 7
184	\$B8	%10111000	[8]	[X]	☐/ 8
185	\$B9	%10111001	[9]	[Y]	☐/ 9
186	\$BA	%10111010	[: ]	[Z]	☐/SH- ;
187	\$BB	%10111011	[; ]	[_]	☐/ ;
188	\$BC	%10111100	[<]	[\]	☐/ <
189	\$BD	%10111101	[=]	[^]	☐/ =
190	\$BE	%10111110	[>]	[`]	☐/ >
191	\$BF	%10111111	[?]	[_]	☐/SH- /

Tabulka kódů a čísel

DEC	HEX	BIN	ATASCII	Intern	komb. kl.
192	\$C0	%11000000	[0]	[0]: [á]	☐/SH- 0
193	\$C1	%11000001	[A]	[1]: [à]	☐/ A
194	\$C2	%11000010	[B]	[2]: [ã]	☐/ B
195	\$C3	%11000011	[C]	[3]: [ä]	☐/ C
196	\$C4	%11000100	[D]	[4]: [å]	☐/ D
197	\$C5	%11000101	[E]	[5]: [ä]	☐/ E
198	\$C6	%11000110	[F]	[6]: [è]	☐/ F
199	\$C7	%11000111	[G]	[7]: [é]	☐/ G
200	\$C8	%11001000	[H]	[8]: [ê]	☐/ H
201	\$C9	%11001001	[I]	[9]: [ÿ]	☐/ I
202	\$CA	%11001010	[J]	[A]: [ü]	☐/ J
203	\$CB	%11001011	[K]	[B]: [ã]	☐/ K
204	\$CC	%11001100	[L]	[C]: [ö]	☐/ L
205	\$CD	%11001101	[M]	[D]: [ó]	☐/ M
206	\$CE	%11001110	[N]	[E]: [ô]	☐/ N
207	\$CF	%11001111	[O]	[F]: [ö]	☐/ O
208	\$D0	%11010000	[P]	[7]: [ü]	☐/ P
209	\$D1	%11010001	[Q]	[8]: [á]	☐/ Q
210	\$D2	%11010010	[R]	[9]: [ö]	☐/ R
211	\$D3	%11010011	[S]	[A]: [í]	☐/ S
212	\$D4	%11010100	[T]	[B]: [é]	☐/ T
213	\$D5	%11010101	[U]	[C]: [è]	☐/ U
214	\$D6	%11010110	[V]	[D]: [ã]	☐/ V
215	\$D7	%11010111	[W]	[E]: [ö]	☐/ W
216	\$D8	%11011000	[X]	[F]: [á]	☐/ X
217	\$D9	%11011001	[Y]	[7]: [à]	☐/ Y
218	\$DA	%11011010	[Z]	[8]: [ä]	☐/ Z
219	\$DB	%11011011	[0]	[9]: [é]	☐/SH- ,
220	\$DC	%11011100	[1]	[A]: [í]	☐/SH- +
221	\$DD	%11011101	[2]	[B]: [ö]	☐/SH- .
222	\$DE	%11011110	[3]	[C]: [á]	☐/SH- *
223	\$DF	%11011111	[4]	[D]: [ä]	☐/SH- -
224	\$E0	%11100000	[5]: [i]	[5]: [i]	☐/CTL- .
225	\$E1	%11100001	[a]	[a]	☐/ A
226	\$E2	%11100010	[b]	[b]	☐/ B
227	\$E3	%11100011	[c]	[c]	☐/ C
228	\$E4	%11100100	[d]	[d]	☐/ D
229	\$E5	%11100101	[e]	[e]	☐/ E
230	\$E6	%11100110	[f]	[f]	☐/ F
231	\$E7	%11100111	[g]	[g]	☐/ G
232	\$E8	%11101000	[h]	[h]	☐/ H
233	\$E9	%11101001	[i]	[i]	☐/ I
234	\$EA	%11101010	[j]	[j]	☐/ J
235	\$EB	%11101011	[k]	[k]	☐/ K
236	\$EC	%11101100	[l]	[l]	☐/ L
237	\$ED	%11101101	[m]	[m]	☐/ M
238	\$EE	%11101110	[n]	[n]	☐/ N
239	\$EF	%11101111	[o]	[o]	☐/ O

## Tabulka kódů a čísel

DEC	HEX	BIN	ATASCII	Intern	komb. kl.
240	\$F0	%11110000	[P]	[P]	☐/ P
241	\$F1	%11110001	[q]	[q]	☐/ Q
242	\$F2	%11110010	[r]	[r]	☐/ R
243	\$F3	%11110011	[s]	[s]	☐/ S
244	\$F4	%11110100	[t]	[t]	☐/ T
245	\$F5	%11110101	[u]	[u]	☐/ U
246	\$F6	%11110110	[v]	[v]	☐/ V
247	\$F7	%11110111	[w]	[w]	☐/ W
248	\$F8	%11111000	[x]	[x]	☐/ X
249	\$F9	%11111001	[y]	[y]	☐/ Y
250	\$FA	%11111010	[z]	[z]	☐/ Z
251	\$FB	%11111011	[⌘]	[⌘]	☐/CTL- ;
252	\$FC	%11111100	[ ]	[ ]	☐/SH- =
253	\$FD	%11111101	[⌥]	[⌥]	ES/CTL- 2
254	\$FE	%11111110	[⌧]	[⌧]	ES/CTL-BKS
255	\$FF	%11111111	[⌨]	[⌨]	ES/CTL->

## Zkratky použité v tabulkách

CTL	CONTROL
SH	SHIFT
BKS	BACK SPACE
ES	ESCAPE
RET	RETURN
SPC	mezerník
☐	INVERSE
-	současné stisknutí
/	postupné stisknutí kláves

Kód klávesy odpovídá ? PEEK(764)

Kód ATASCII odpovídá ? ASC(ř)

Kód Intern odpovídá ? PEEK(a)

Znaky uzavřené do hranatých závorek jsou v inverzi!

Tam, kde je sloupec ATASCII nebo Intern rozdělen dvojtečkou, platí levý sloupec pro standardní znakovou sadu (POKE 756,224) a pravý sloupec pro mezinárodní znakovou sadu (POKE 756,204).

Tabulka strojových kódů a kódů kláves

DEC	HEX	MNEMONICS	kód klávesy
0	\$00	BRK	L
1	\$01	ORA (z, X)	J
2	\$02		;
3	\$03		nepoužito
4	\$04		nepoužito
5	\$05	ORA z	K
6	\$06	ASL z	+
7	\$07		*
8	\$08	PHP	0
9	\$09	ORA #n	nepoužito
10	\$0A	ASL A	P
11	\$0B		U
12	\$0C		RETURN
13	\$0D	ORA Q	I
14	\$0E	ASL Q	-
15	\$0F		=
16	\$10	BPL d	V
17	\$11	ORA (z), Y	nepoužito
18	\$12		C
19	\$13		nepoužito
20	\$14		nepoužito
21	\$15	ORA z, X	B
22	\$16	ASL z, Y	X
23	\$17		Z
24	\$18	CLC	4
25	\$19	ORA Q, Y	nepoužito
26	\$1A		3
27	\$1B		6
28	\$1C		ESCAPE
29	\$1D	ORA Q, X	5
30	\$1E	ASL Q, X	2
31	\$1F		1
32	\$20	JSR Q	,
33	\$21	AND (z, X)	mezerník
34	\$22		.
35	\$23		N
36	\$24	BIT z	nepoužito
37	\$25	AND z	M
38	\$26	ROL z	/
39	\$27		inverse
40	\$28	PLP	R
41	\$29	AND #n	nepoužito
42	\$2A	ROL A	E
43	\$2B		Y
44	\$2C	BIT Q	TAB
45	\$2D	AND Q	T
46	\$2E	ROL Q	W
47	\$2F		0

Tabulka strojových kódů a kódů kláves

DEC	HEX	MNEMONICS	kód klávesy
46	\$30	BMI d	9
49	\$31	AND (z), Y	nepoužito
50	\$32		0
51	\$33		7
52	\$34		BACK SPACE
53	\$35	AND z, X	8
54	\$36	ROL z, Y	<
55	\$37		>
56	\$38	SEC	F
57	\$39	AND Q, Y	H
58	\$3A		D
59	\$3B		nepoužito
60	\$3C		CAPS
61	\$3D	AND Q, X	G
62	\$3E	ROL Q, X	S
63	\$3F		A
64	\$40	RTI	SH- L
65	\$41	EOR (z, X)	SH- J
66	\$42		SH- ;
67	\$43		nepoužito
68	\$44		nepoužito
69	\$45	EOR z	SH- K
70	\$46	LSR z	SH- +
71	\$47		SH- *
72	\$48	PHA	SH- 0
73	\$49	EOR #n	nepoužito
74	\$4A	LSR A	SH- P
75	\$4B		SH- U
76	\$4C	JMP Q	SH- RETURN
77	\$4D	EOR Q	SH- I
78	\$4E	LSR Q	SH- -
79	\$4F		SH- =
80	\$50	BVC d	SH- U
81	\$51	EOR (z), Y	nepoužito
82	\$52		SH- C
83	\$53		nepoužito
84	\$54		nepoužito
85	\$55	EOR z, X	SH- B
86	\$56	LSR z, Y	SH- X
87	\$57		SH- Z
88	\$58	CLI	SH- 4
89	\$59	EOR Q, Y	nepoužito
90	\$5A		SH- 3
91	\$5B		SH- 6
92	\$5C		SH- ESCAPE
93	\$5D	EOR Q, X	SH- 5
94	\$5E	LSR Q, X	SH- 2
95	\$5F		SH- 1

Tabulka strojových kódů a kódů kláves

DEC	HEX	MNEMONICS	kód klávesy
96	\$60	RTS	SH- ,
97	\$61	ADC (z, X)	SH- SPC
98	\$62		SH- .
99	\$63		SH- N
100	\$64		nepoužito
101	\$65	ADC z	SH- M
102	\$66	ROR z	SH- /
103	\$67		SH- inverse
104	\$68	PLA	SH- R
105	\$69	ADC #n	nepoužito
106	\$6A	ROR A	SH- E
107	\$6B		SH- Y
108	\$6C	JMP (Q)	SH- TAB
109	\$6D	ADC Q	SH- T
110	\$6E	ROR Q	SH- W
111	\$6F		SH- Q
112	\$70	BUS d	SH- 9
113	\$71	ADC (z), Y	nepoužito
114	\$72		SH- 0
115	\$73		SH- 7
116	\$74		SH- BKS
117	\$75	ADC z, X	SH- 6
118	\$76	ROR z, Y	SH- <
119	\$77		SH- >
120	\$78	SEI	SH- F
121	\$79	ADC Q, Y	SH- H
122	\$7A		SH- D
123	\$7B		nepoužito
124	\$7C		SH- CAPS
125	\$7D	ADC Q, X	SH- G
126	\$7E	ROR Q, X	SH- S
127	\$7F		SH- A
128	\$80		CTL- L
129	\$81	STA (z, X)	CTL- J
130	\$82		CTL- ;
131	\$83		nepoužito
132	\$84	STY z	nepoužito
133	\$85	STA z	CTL- K
134	\$86	STX z	CTL- +
135	\$87		CTL- *
136	\$88	DEY	CTL- 0
137	\$89		nepoužito
138	\$8A	TXA	CTL- P
139	\$8B		CTL- U
140	\$8C	STY Q	CTR- RETURN
141	\$8D	STA Q	CTL- I
142	\$8E	STX Q	CTL- -
143	\$8F		CTL- =

Tabulka strojových kódů a kódů kláves

DEC	HEX	MNEMONICS	kód klávesy
144	\$90	BCC d	CTL- U
145	\$91	STA (z), Y	nepoužito
146	\$92		CTL- C
147	\$93		nepoužito
148	\$94	STY z, X	nepoužito
149	\$95	STA z, X	CTL- B
150	\$96	STX z, Y	CTL- X
151	\$97		CTL- Z
152	\$98	TYA	CTL- 4
153	\$99	STA Q, Y	nepoužito
154	\$9A	TXS	CTL- 3
155	\$9B		CTL- 6
156	\$9C		CTL- ESCAPE
157	\$9D	STA Q, X	CTL- 5
158	\$9E	ASL Q, X	CTL- 2
159	\$9F		CTL- 1
160	\$A0	LDY #n	CTL- ,
161	\$A1	LDA (z, X)	CTL- SPC
162	\$A2	LDX #n	CTL- .
163	\$A3		CTL- N
164	\$A4	LDY z	nepoužito
165	\$A5	LDA z	CTL- M
166	\$A6	LDX z	CTL- /
167	\$A7		CTL- INV
168	\$A8	TAY	CTL- R
169	\$A9	LDA #n	nepoužito
170	\$AA	TAX	CTL- E
171	\$AB		CTL- Y
172	\$AC	LDY Q	CTL- TAB
173	\$AD	LDA Q	CTL- T
174	\$AE	LDX Q	CTL- W
175	\$AF		CTL- Q
176	\$B0	BCS d	CTL- 9
177	\$B1	LDA (z), Y	nepoužito
178	\$B2		CTL- 0
179	\$B3		CTL- 7
180	\$B4	LDY z, X	CTL- BKS
181	\$B5	LDA z, X	CTL- 8
182	\$B6	LDX z, Y	CTL- <
183	\$B7		CTL- >
184	\$B8	CLU	CTL- F
185	\$B9	LDA Q, Y	CTL- H
186	\$BA	TSX	CTL- D
187	\$BB		nepoužito
188	\$BC	LDY Q, X	CTL- CAPS
189	\$BD	LDA Q, X	CTL- G
190	\$BE	LDX Q, Y	CTL- S
191	\$BF		CTL- A

Tabulka strojových kódů a kódů kláves

DEC	HEX	MNEMONICS	kód klávesy
192	\$C0	CPY #n	SH-CTL- L
193	\$C1	CMP (z, X)	SH-CTL- J
194	\$C2		SH-CTL- ;
195	\$C3		nepoužito
196	\$C4	CPY z	nepoužito
197	\$C5	CMP z	SH-CTL- K
198	\$C6	DEC z	nepoužito
199	\$C7		nepoužito
200	\$C8	INY	SH-CTL- 0
201	\$C9	CMP #n	nepoužito
202	\$CA	DEX	SH-CTL- P
203	\$CB		SH-CTL- U
204	\$CC	CPY 0	SH-CTL-RET
205	\$CD	CMP 0	SH-CTL- I
206	\$CE	DEC 0	SH-CTL- -
207	\$CF		SH-CTL- =
208	\$D0	BNE d	SH-CTL- U
209	\$D1	CMP (z), Y	nepoužito
210	\$D2		SH-CTL- C
211	\$D3		nepoužito
212	\$D4		nepoužito
213	\$D5	CMP z, X	SH-CTL- B
214	\$D6	DEC z, Y	SH-CTL- X
215	\$D7		SH-CTL- Z
216	\$D8	CLD	SH-CTL- 4
217	\$D9	CMP 0, Y	nepoužito
218	\$DA		SH-CTL- 3
219	\$DB		SH-CTL- 6
220	\$DC		SH-CTL-ES
221	\$DD	CMP 0, X	SH-CTL- 5
222	\$DE	DEC 0, X	SH-CTL- 2
223	\$DF		SH-CTL- 1
224	\$E0	CPX #n	SH-CTL- ,
225	\$E1	SBC (z, X)	SH-CTL-SPC
226	\$E2		SH-CTL- .
227	\$E3		SH-CTL- N
228	\$E4	CPX z	nepoužito
229	\$E5	SBC z	SH-CTL- M
230	\$E6	INC z	SH-CTL- /
231	\$E7		SH-CTL- INU
232	\$E8	INX	SH-CTL- R
233	\$E9	SBC #n	nepoužito
234	\$EA	NOP	SH-CTL- E
235	\$EB		SH-CTL- Y
236	\$EC	CPX 0	SH-CTL-TAB
237	\$ED	SBC 0	SH-CTL- T
238	\$EE	INC 0	SH-CTL- W
239	\$EF		SH-CTL- 0

## Tabulka strojových kódů a kódů kláves

DEC	HEX	MNEMONICS	kód klávesy
240	\$F0	BEQ d	SH-CTL- 9
241	\$F1	SBC (z), Y	nepoužito
242	\$F2		SH-CTL- 0
243	\$F3		SH-CTL- 7
244	\$F4		SH-CTL-BK5
245	\$F5	SBC z, X	SH-CTL- 8
246	\$F6	INC z, Y	SH-CTL- <
247	\$F7		SH-CTL- >
248	\$F8	SED	SH-CTL- F
249	\$F9	SBC Q, Y	SH-CTL- H
250	\$FA		SH-CTL- 0
251	\$FB		nepoužito
252	\$FC		SH-CTL-CAPS
253	\$FD	SBC Q, X	SH-CTL- G
254	\$FE	INC Q, X	SH-CTL- S
255	\$FF		SH-CTL- A

## Způsob zápisu adresování strojového kódu:

Q	absolute	absolutní
(Q)	indirect	absolutní nepřímé
Z	zero page	nulová stránka
d	relative	relativní
A	accumulator	akumulátor
#n	immediate	přímé
	implied	implicitní
Q, X	absolute, index X	absolutní indexované (X)
Q, Y	absolute, index Y	absolutní indexované (Y)
Z, X	zero page, index X	nulová stránka indexovaná (X)
Z, Y	zero page, index Y	indexovaná (X)
(z, X)	index, indirect	indexované nepřímé (X)
(z), Y	indirect, index	nepřímé indexované (Y)

## Seznam klíčových slov TB

klíčové slovo	zkratka	AB?	kapitola
ABS	---	+	
ADR	---	+	
AND	---	+	
ASC	---	+	
ATN	---	+	
BGET	BG.	-	3.4.12
BLOAD	BL.	-	3.4.13
BPUT	BP.	-	3.4.12
BRUN	BR.	-	3.4.13
BYE	B.	+	
CHRS	---	+	
CIRCLE	CI.	-	3.4.10
CLOAD	CLOA.	+	
CLOG	---	+	
CLOSE	CL.	+-	3.4.12
CLR	---	+	
CLS	---	-	3.4.10
COLOR	C.	+	
COM	---	+	
CONT	CON.	+	
COS	---	+	
CSAVE	CS.	+	
DATA	D.	+	
DEC	---	-	3.5
DEG	DE.	+	
DEL	---	-	3.4.8
DELETE	DEL.	-	3.4.13
DIM	DI.	+	
DIR	---	-	3.4.13
DIV	---	-	3.6.1
DO	---	-	3.4.4
DOS	DO.	+	
DPEEK	---	-	3.5
DPOKE	DP.	-	3.4.9
DRAWTO	DR.	+	
DSOUND	DS.	-	3.4.11
DUMP	DU.	-	3.4.8
ELSE	EL.	-	3.4.2
END	---	+	
ENDIF	END.	-	3.4.2
ENDPROC	ENDP.	-	3.4.3
ENTER	E.	+	
ERL	---	-	3.5
ERR	---	-	3.5
EXEC	EXE.	-	3.4.3, 3.4.7
EXIT	EX.	-	3.4.5
EXOR	---	-	3.6.2
EXP	---	+	

klíčové slovo	zkratka	AB?	kapitola
FCOLOR	FC.	-	3.4.10
FILLTO	FI.	-	3.4.10
FOR	F.	+	3.4.4
FRAC	---	-	3.5
FRE	---	+	
GET	GE.	+ -	3.4.12
GET#	---	-	3.4.12
GOSUB	GOS.	+	
GO TO	GO .	+	
GOTO	G.	+	
GO#	---	-	3.4.6, 3.4.7
GRAPHICS	GR.	+	
HEX\$	---	-	3.5
IF	---	+	3.4.2
INKEY\$	---	-	3.5
INPUT	I.	+ -	3.4.12
INSTR	---	-	3.5
INT	---	+	
LEN	---	+	
LET	LE.	+	
LIST	L.	+ -	3.4.8
LOAD	LO.	+	
LOCATE	LOC.	+	
LOCK	---	-	3.4.13
LOG	---	+	
LOOP	LOO.	-	3.4.4
LPRINT	LP.	+	
MOD	---	-	3.6.1
MOVE	M.	-	3.4.9
NEW	---	+	
NEXT	N.	+	3.4.4
NOT	---	+	
NOTE	NO.	+	
ON	---	+ -	3.4.7
OPEN	O.	+	
OR	---	+	
PADDLE	---	+	
PAINT	PAI.	-	3.4.10
PAUSE	PA.	-	3.4.14
PEEK	---	+	
PLOT	PL.	+	
POINT	P.	+	
POKE	POK.	+	
POP	---	+	
POSITION	POS.	+	

klíčové slovo	zkratka	AB?	kapitola
PRINT	PR. (?)	+	
PROC	PRO.	-	3.4.3
PTRIG	---	+	
PUT	PU.	+ -	3.4.12
PUT#	---	-	3.4.12
RAD	RA.	+	
RAND	---	-	3.5
READ	REA.	+	
REM	R. ( , )	+	
RENAME	REN.	-	3.4.13
RENUM	RENU.	-	3.4.6
REPEAT	REP.	-	3.4.4
RESTORE	RES.	+ -	
RESTORE#	---	-	3.4.6
RETURN	RET.	+	
RND	---	+ -	3.5
RUN	RU.	+	
SAVE	S.	+	
SETCOLOR	SE.	+	
SGN	---	+	
SIN	---	+	
SOUND	SO.	+ -	3.4.11
SQR	---	+	
STATUS	ST.	+	
STEP	---	+	3.4.4
STICK	---	+	
STOP	STO.	+	
STRIG	---	+	
STR\$	---	+	
TEXT	TE.	-	3.4.12
THEN	---	+	
TIME	---	-	3.5
TIMES	---	-	3.5
TIMES=	---	-	3.4.14
TO	---	+	
TRACE	TRAC.	-	3.4.6
TRACE+	TRAC. +	-	3.4.6
TRACE-	TRAC. -	-	3.4.6
TRAP	T.	+ -	
TRAP#	---	-	3.4.6
TRUNC	---	-	3.5
UINSTR	---	-	3.5
UNLOCK	UNL.	-	3.4.13
UNTIL	U.	-	3.4.4
USR	---	+	
VAL	---	+	
WHILE	W.	-	3.4.4
WEND	WE.	-	3.4.4

klíčové slovo	zkratka	AB?	kapitola
%	---	-	3.7
%PUT	%.	-	3.4.12
%GET	%G.	-	3.4.12
-MOVE	-.	-	3.4.9
*B	---	-	3.4.14
*B+	---	-	3.4.14
*B-	---	-	3.4.14
*F	*,	-	3.4.4
*F+	*,	-	3.4.4
*F-	---	-	3.4.4
*L	---	-	3.4.6
*L+	---	-	3.4.6
*L-	---	-	3.4.6
\$abcd	---	-	3.7
&	---	-	3.6.2
!	---	-	3.6.2
"	---	-	3.7
-	---	-	3.7
--	---	-	3.4.6
# jméno	---	-	3.4.6
?	---	+	

### Vysvětlivky:

Znaky --- ve sloupci "zkratka" znamenají, že klíčové slovo nelze psát zkratkou. Má-li klíčové slovo více možností zkrácení, jsou tyto možnosti uvedeny v závorkách.

Ve sloupci AB? je vyznačeno, zda se jedná o slovo z AB. Znak + znamená ano, znak - ne, tj. dané klíčové slovo lze použít pouze v TB. Jsou-li v tomto sloupci znaky +-, jde o klíčové slovo známé z AB, které však má v TB více možností použití.

# O B S A H

1.	ÚVOD .....	3
2.	ZAVEDENÍ DO POČÍTACE .....	3
3.	VÝKLAD TB .....	4
3.1	Zkratky, značky a symboly použité v textu .....	4
3.2	Editace programu .....	4
3.3	Jména proměnných, návěští a procedur v TB .....	5
3.4	Příkazy TB .....	6
3.4.1	Strukturované programování - - úvod .....	6
3.4.2	Příkazy větvení (podmíněného skoku) .....	6
3.4.3	Procedury .....	8
3.4.4	Příkazy cyklu .....	10
3.4.5	Příkazy opuštění struktur .....	13
3.4.6	Symbolická návěští .....	14
3.4.7	Příkazy násobného větvení .....	15
3.4.8	Příkazy pro editaci programu .....	16
3.4.9	Příkazy operující s pamětí .....	19
3.4.10	Příkazy pro grafiku .....	21
3.4.11	Příkazy zvuku .....	24
3.4.12	Příkazy I/O operací .....	25
3.4.13	Příkazy pro disketovou jednotku .....	31
3.4.14	Ostatní příkazy .....	36
3.5	Funkce TB .....	38
3.6	Operátory TB .....	43
3.6.1	Aritmetické operátory .....	43
3.6.2	Logické operátory .....	44
3.6.3	Priorita operátorů TB .....	45
3.7	Speciální příkazy, funkce, operátory a výrazy .....	46
3.8	Chybová hlášení TB .....	47
3.9	Paměť a TB .....	48
3.10	Kompatibilita AB a TB .....	50
3.11	Překladač TB .....	51
3.12	DOS a TB .....	57
4.	RADY A NÁPADY .....	59
5.	DODATKY A TABULKY .....	61
	Turbo-BASIC XL 2.0 .....	61
	Tabulka not pro příkazy SOUND a DSOUND .....	63
	Tabulka grafických modů .....	65
	Tabulka barev .....	69
	Tabulka paměti PMG .....	70
	Tabulka chybových hlášení TB .....	71
	Tabulka kódů a čísel .....	77
	Tabulka strojových kódů a kódů kláves .....	83
	Seznam klíčových slov TB .....	89

**Publikované zo súhlasom - vid' Prohlášení představitelů AK Praha.**