COMPILER

basic

C O M M O D O R E 128

Přeložil:

ing Bruno BREYL

PRAHA 17.4.1988

Pisemné materiály k přednášce.
Určeno výhradně pro potřebu členů klubu C 64
při 602. ZO Svazarmu v Praze 6.

K O M P I L E R

B A S I C 1 2 8

Vydalo: Nakladatelství DATA BECKER GmbH
Merowingerstrasse 30
4000 Düsseldorf

Přeložil: Ing Bruno B r e y l

Praha 14. 11. 1987

B A S I C - 1 2 8

ÚVOD

BASIC 128 je optimalizační basicový kompiler pro počítač Commodore 128, který vaše programy napsané v basicu urychlí a udělá výkonnější. BASIC 128 má všechny možnosti známého Basic-64-Compilers a současně obsahuje další významné inovace. Samozřejmě je plně kompatibilní k basicu 7.Ø a má vylepšený Code--generátor.

P-kódy generované BASICem 128 jsou až 15krát rychlejší a obzvláště u velkých programu zkracují je až o 30 % oproti nekompilovanému programu. BASIC 128 umí také váš program zvoleným způsobem přeložit do strojového kódu, který je až 35krát rychlejší a přirozeně je možné v programu přepínat mezi oběma způsoby. K tomu je třeba ještě vzít v úvahu FAST-modus počítače C-128. Tím se samozřejmě dvakrát zrychlí běh zkompilovaných programů.

Navíc obsahuje systém takové funkce, že programy, které po zkompilování nemohou ještě být optimální, přesto urychlují běh programu 4 ÷ 10 krát. Obzvláště účinné je to u programů s mnoha proměnnými s pohyblivou desetinnou čárkou a znakovými řetězci (stringy).

Funkce s pohyblivou desetinnou čárkou (pozor! - v dalším čs. textu bude výraz pohyblivá desetinná čárka uváděn pouze zkratkou PDC) jako TAN, ATN, SIN, COS, ↑ , SQR, EXP a LOG patří k několika operacím, které jsou proti dřívějším versím kompileru podstatně výkonnější. BASIC 128 používá k výpočtu těchto funkcí nové rutiny, které dříve byly používány pouze u podstatně dražších počítačů. Pomocí těchto rutin jsou tyto funkce 5 krát urychleny (některé až 11 krát). Právě toto činí BASIC 128 nepostradatelný pro vědecké výpočty, ale také komplikovaná jemná grafika bude pracovat s těmito funkcemi rychleji.

Samozřejmě kompiluje BASIC 128 programy libovolné velikosti a také Warm a Kalt Overlay - Pakety, přičemž rychlost kompilování obnáší podle typu floppy jednotky asi 1 ÷ 2 kbyty za minutu. Paměť, která není obsazena kompilovaným programem, může být použita pro uložení dat. K použití tedy je podle délky programu možnost uložit data s 60-ti až 100 kbyty.

BASIC 128 vám nabízí mnoho dalších možností, jako např.: dva optimalizační stupně, proměnný Code-start, proměnlivé použití paměti, automatické zapnutí nebo podle přání také vypnutí módu PDC (pohyblivé desetinné čárky), předefinování proměnných datového typu během kompilování, optimalizování a přeformulování vzorců, kontrolu syntaxe u programů, sestavení řádkového adresového výpisu, datový interfejs k assembleru, snadné použití, varování při výskytu chyby, nový start kompileru bez opakovaného natažení a jiné.

BASIC 128 vytváří společně s basicovým překladačem zabudovaným v počítači C-128 ideální vývojový systém, se kterým mohou být napsány rychlé a výkonné programy.

Thomas Helbig Říjen 1985

Poznámka: V dalším textu bude tento kompilační BASIC 128 psán velkými písmeny. Nezaměňovat za basic překladače, který je v ROM počítače C-128.

1. POUŽITÍ BASICU 128

Použití BASICU 128 je velmi jednoduché. Uložte si svůj basicový program na disketu. Není podmínkou, aby tato disketa byla jinak prázdná, ale musí mít dostatečnou volnou kapacitu (pro velké programy až 400 volných bloků). Není bezpodmínečně nutné pořizovat si záložní kopii vašeho programu, protože BASIC 128 si váš program pouze přečte a dále s ním nepracuje, nepřepisuje jej ani nemaže.

Nyní vložte do disku disketu s BASICem 128 a odstartujte kompilovací program pomocí povelu

RUN "BASIC 128"

Po natažení se kompiler ohlásí hlášením "BASIC 128 Compiler" a s číslem verse. Na obrazovce se objeví menu s možností volby funkcí kompileru. Vyjměte disketu BASIC 128 a vložte do disku disketu s vaším programem, který chcete kompilovat.

Stiskněte <1> nebo klávesu <RETURN>. Kompiler se vás zeptá na jméno vašeho programu. Zadejte je a pak stiskněte <RETURN>.

Kompiler nyní přeloží váš program do P-kódového programu. Na obrazovce se objeví číslo právě zpracovávaného řádku a tím podává kompiler informace o generovaném programu. Objeví-li kompiler ve vašem programu chyby, podá o tom odpovídající hlášení. Kompilování se nezastaví, ale pokračuje dále, aby byly odhaleny případné další chyby ve vašem programu.

Jestliže kompiler ukončil svoji práci, hlásí se "READY" a ozve se akustický signál. Stiskněte klávesu <N>, abyste sdělili kompileru, že Nechcete kompilovat další program. Stisknutím kterékoliv jiné klávesy můžete kompiler znovu odstartovat.

Jestliže byly ve vašem programu odhaleny chyby, musíte je nejprve odstranit a pak nechat váš program znovu zkompilovat.

Zkompilovaný program se nyní nachází na disketě. Aby byl v directory odlišen od normálních basicových programů, je zkompilovaný program označen předponou "P-". Tento program pak startujte příkazem

RUN "P - název programu"

Váš program, nyní zkompilovaný, se natáhne a odstartuje a poběží podstatně rychleji než původní, čistý basicový program.

Eventuální ještě existující chyby budou hlášeny přesně tak, jako v basicovém programu. Nebude ovšem oznámeno číslo řádku, nýbrž místo v paměti, ve kterém se chyba nachází. S pomocí výpisu řádků je možné snadno zjistit posici s chybou v originálním programu. Výpis řádků provede kompiler na vaše přání.

Jestliže vám rychlost programu stále ještě nevyhovuje, pak se nabízí další možnost úpravy programu (volbou optimalizačního stupně č. 2 bude vygenerován strojový program, použity celočíselné proměnné (integer a j.). To vše bude vysvětleno v následujících kapitolách.

Ke zkompilování několika málo povelů basicu 7.0 je nutné změnit nastavení kompileru pomocí menu, které kompiler nabízí (např. Overlay - Pakete). V těchto případech vydá kompiler odpovídající upozornění (viz 4.2).

Pozor! Přesto, že obsluha BASICu 128 je velmi jednoduchá, je třeba pro plné pochopení a využití kompileru bezpodmínečně přečíst tento návod až do konce.

2. ZÁKLADY

2.1 Volba možností v menu

Po odstartování BASICu 128 se objeví na obrazovce hlavní menu. Čtyři možné funkce, které jsou postupně očíslovány 1 až 4, je možné zvolit stisknutím odpovídající klávesy. Nejdůležitější bod menu je <1>, který odstartuje kompilování programu s nastaveným stupněm. (Rozdíl mezi oběma stupni je popsán v kapitole 7.) Volbu <1> můžete také provést nejen klávesou <1>, ale také samotnou klávesou <RETURN>.

Volba <2> natáhne všechna uložená nastavení kompileru z disketového souboru a zobrazí je v podmenu, do kterého se také můžete dostat volbou <3>. Ke zkompilování Overlay-Paket slouží v menu bod <4>. Toto je popsáno v kapitole 6.

2.2 Rozšířený vývojový paket

K vývoji programů se speciálním použitím nabízí BASIC 128 možnost pozměnit většinu vlastností generovaných programů. K tomu je třeba zvolit v menu bod 3. Na obrazovce se pak objeví všechny měnitelné hodnoty. Všechny body menu jsou značeny písmeny v abecedním pořadí. Stisknutím odpovídající klávesy s písmenem můžete dosáhnout požadované nastavení, u některých programů se dostanete do dalšího podmenu. Přesnější popis jednotlivých možností menu naleznete v odpovídajících kapitolách tohoto návodu a v kapitole č. 9. Jednoduchý příklad pro použití vývojového paketu je volba Code-generátoru.

Po volbě <3> v hlavním menu se objeví druh kompilerem generovaného kódu pod bodem menu A. Po startu kompileru je nastaven P-code generátor. Stisknutím klávesy <A> můžete kompileru přikázat, aby váš program přeložil do strojového kódu. Pomocí <RETURN> se dostanete zpět do hlavního menu. Dalším stisknutím <RETURN> odstartujete zvolený kompilační průběh.

Jiné změny přednastavených hodnot můžete provést snadno jednoduchým způsobem. Přirozeně můžete měnit také více hodnot. Kvůli zjednodušení zápisu budou možnosti volby vývojového paketu označovány v následujících kapitolách pouze volbou "A" až volbou "C". Možnosti volby v hlavním menu napsaném tomu pouze číslem "1" až "4".

2.3 Kompilační příkazy

Volba všech kompilačních možností může být provedena pouze před vlastním kompilováním. V mnoha případech je požadováno předat kompileru některé příkazy v průběhu kompilování. Tyto příkazy musí pak být poznačeny uvnitř kompilovaného programu. Aby však nebyl uveden do zmatku basicový překladač C-128, musí příkaz pro kompiler následovat za povelem REM, který basicový překladač přeskakuje. Symbol k označení těchto příkazů je "zavináč" - @ (v DIN pak §). Formát zápisu příkazu pro kompiler pak je následující:

REM @ příkaz

Výpis všech možných příkazů pro kompiler naleznete v kapitole 9.

Například:

REM @ I = proměnná, proměnná, ...

Tento příkaz způsobí, že všechny za ním vyjmenované proměnné budou kompilerem interpretovány jako celočíselné proměnné (proměnné typu integer). Tento příkaz je vhodný při vývoji obzvláště rychlých programů.

2.4 Příklad vývoje programu

Jednoduchý příklad programu, jehož průběh bude zkompilováním urychlen, je počítání s prvočísly. Následující program, který je znám jako Benchmarkův test rychlosti počítačů, programovacích jazyků a kompilování:

```
 1Ø  REM SIEB DES ERATOSTHENES
 2Ø  DIM Z% (1Ø ØØØ)
 3Ø  FOR I=1 TO 1ØØØØ
 4Ø  Z% (I)=1
 5Ø  NEXT
 6Ø  PRINT 2
 7Ø  FOR I=1 TO 1ØØØØ
 8Ø  IF Z% (I)=Ø THEN 15Ø
 9Ø  PRINT I*2+1
1ØØ  K=1
11Ø  IF I+K(I*2+1)  1ØØØØ THEN 15Ø
12Ø  Z% (I+K*(I*2+1))=Ø
13Ø  K=K+1
14Ø  GOTO 11Ø
15Ø  NEXT
```

Budete-li nyní měřit čas, který program potřebuje k provedení, pak to bude z velké části čas, který je třeba pro výpis prvočísel na obrazovku. Aby bylo možné porovnávat počítací rychlost programu, provádí se obvykle následující změna v programu na řádku 9Ø:

```
9Ø  Q = I* 2+1
```

Přes tuto změnu potřebuje program ještě stále mnoho počítacího času. Provede-li se však na něm zkompilování postupem popsaným v kapitole 1, bude průběh tohoto programu asi 7 krát zrychlen. V kap. 1 bylo uvedeno, že je možné další urychlení. Na tomto programu si ukážeme více možností, jak to provést.

Napište tento program ještě jednou, ale tak, aby v něm byly pouze celočíselné proměnné. To provedete přidáním "%" za každou proměnnou. Pozor na to, že basicový překladač v C-128 nezná celočíselné (integer) proměnné ve smyčce FOR--NEXT. Proto byste museli celý program přeformulovat. Abyste si tuto námahu ušetřili, použijte následujícího příkazu pro kompiler v řádku 5:

```
5  REM @ I = I,K,Q
```

Obzvláště je tento příkaz účinný na smyčkovou proměnnou I, což je také možné vyzkoušet s basicovým překladačem, a to bez přeformulování programu.

Příkaz pro kompiler není nutný, pokud program počítá výlučně s celými čísly. Můžete jej také zkompilovat volbou stupně 2 (volba "O" ve vývojovém paketu).

Navíc můžete u každého programu přikázat kompileru, aby vytvořil program ve strojovém kódu. Toto provádí bod "A" menu. Přirozeně je možné tento příkaz dát kompileru následujícím příkazem v řádku č. 1

```
1 REM O M
```

Na řádek 2 pak dalším příkazem

```
2 REM O Ø2
```

přikážeme kompileru pracovat se stupněm kompilace č. 2. Program po zkompilování běží více než 18 krát rychleji než původní basicový program. Velmi se tedy vyplatí všímat si pravidel pro programování a kompilování.

Další příklad pro účinné zvýšení rychlosti je třídění dat.

```
 1 REM O M
 2 REM O Ø2
 4 REM QUICKSORT
 5 DIM SR%(2Ø),SL%(2Ø)
1Ø DIM A%(1ØØØ)
2Ø INPUT "POCET CISEL"; N
3Ø FOR I=1 TON: A%(I)=RND(1)*1ØØØØ:NEXT
4Ø TI$="ØØØØØØ":PRINT "START"
5Ø GOSUB 1ØØØ
6Ø A$=TI$:PRINT "HOTOVO"
7Ø FOR I=1 TON:PRINT A%(I),:NEXT:PRINT:PRINT
   "CAS TRIDENI=, "A$:END
```

```
1ØØØ  SP=Ø:L=1:R=N
1Ø1Ø  IF R<=L THEN 1Ø7Ø
1Ø15  X=A%((L+R)/2):LC=L-1:RC=R+1
1Ø2Ø  LC=LC+1:IF A%(LC)<X THEN 1Ø2Ø
1Ø3Ø  RC=RC-1:IF A%(RC)>X
1Ø4Ø  IF LC<=RC THEN HI=A%(RC):A%(RC)=
      A%(LC):A%(LC)=HI:GOTO 1Ø2Ø
1Ø5Ø  IF RC-L<R-LC THEN SL%(SP)=LC:SR%(SP)=
      R:SP=SP+1:R=RC:GOTO 1Ø1Ø
1Ø6Ø  SL%(SP)=L:SR%(SP)=RC:SP=SP+1:L=LC:GOTO 1Ø1Ø
1Ø7Ø  IF SP>=1 THEN SP=SP-1:L=SL%(SP):R=SR%(SP):GOTO 1Ø1Ø
1Ø8Ø  RETURN
```

Tento program pracuje dokonce 36 krát rychleji a třídí 1ØØØ hodnot čísel za 1Ø sekund, respektive za 5 sekund v módu FAST.

3. OPTIMALIZACE PROGRAMU

Již při sestavování programu, který má běžet s maximální možnou rychlostí, je užitečné přihlížet k možnostem kompileru a kritické části uvnitř vašeho programu pak přizpůsobit odpovídajícím způsobem kompileru. U programu, u kterého to bylo opomenuto, nebo původně se u nich nepočítalo s následnou kompilací, může se vše napravit volbou stupně 2. Toto bude popsáno v kapitole 7. Na druhé straně se často stává, že přizpůsobení, které vychází vstříc kompileru, není nutné, protože kompiler je provádí automaticky sám. Také z tohoto důvodu je dobré znát činnost a možnosti kompileru a tím si ušetřit námahu při psaní prvotního programu.

3.1 Všeobecně ke způsobu činnosti kompileru

Při dlouhodobé práci s basicovým překladačem získáte následující zkušenosti, které se vztahují k rychlosti programu:

- GOTO/GOSUB pracuje pomaleji než RETURN, ačkoliv oba skoky probíhají uvnitř programu
- smyčka, která je vytvořena pomocí IF-THEN pracuje pomaleji než smyčka s FOR-NEXT
- vzorce a výrazy, které obsahují konstanty, budou spočítány pomaleji, než stejné vzorce, kde jsou proměnné (které obsahují konstanty); to platí pro čísla a většinou také pro znakové řetězce
- proměnné, které program použije až později, pracují pomaleji než ty, které jsou deklarovány těsně před použitím
- rychlost skoku GOTO- a GOSUB je u skoků dopředu závislá na vzdálenosti skoku; u skoků zpět závisí rychlost na vzdálenosti cílového řádku od začátku programu (čím je cíl více vzdálen od řádku, tím více potřebuje vyhledávací rutina času pro nalezení)
- celočíselné proměnné (integer) budou zpracovány pomaleji než proměnné s PDC (pohyblivou desetinnou čárkou). Úspora paměti je opodstatněná pouze u integer proměnných při zpracování polí
- u velkých programů jsou skoky GOTO/GOSUB a zpracování proměnných až 4krát pomalejší
- zavedením mezer bude basicový program sice přehlednější, ale podstatně pomalejší
- strukturované programy jsou většinou pomalejší než programy napsané typem "špagety"
- bude-li na začátku programu dimensováno velké pole, pak pausíruje překladač při použití nějaké nové proměnné několik sekund.

Se všemi těmito body a mnoha dalšími se u zkompilovaných programů již nesetkáme. Není nutné přizpůsobovat programy na basicový interpreter (překladač). Přirozeně jsou také u kompilovaných programů operace, které pracují rychleji než jiné a často je užitečné je znát a používat je.

Někdy je položena otázka, kdy a proč kompilované programy pracují rychleji než nekompilované. Nejdůležitější důvody jsou:

- práce interpretovaného basic-programu bude zvýšena použitím jistého druhu pseudo-strojového jazyka (P-code), což podstatně urychluje průběh programu;
  BASIC 128 využívá navíc možnost vytvářet dobré strojové programy a tím šetřit interpretaci
- interpretaci vzorců (výrazů) přebírá kompiler; zkompilovaný program počítá vzorce značně rychlým postupem
- při zpracování proměnných nemusí být tyto vyhledávány v tabulce, nýbrž je možné na ně přímo přistupovat
- k lokalizaci cílů skoků GOTO-/GOSUB-není nutné prohledávat celý program, skok následuje přímo. Příklad: 1 skok GOTO ve strojovém programu trvá 3 mikrosekundy, u dlouhých basicových programů naproti tomu až půl sekundy
- konstanty musí být basicovým překladačem nejdříve převedeny do binárního tvaru, jelikož program je uložen v decimální formě. Podobně to platí pro čísla řádků pro GOTO-/GOSUB-. U zkompilovaného programu jsou konstanty naproti tomu uloženy již binárně a není je tedy nutné převádět.

Uveďme si příklad

```
10  FOR I=1 TO 1000:NEXT
20  PRINT "KONEC SMYCKY FOR-NEXT"
30  I=1
40  I=I+1:IF I<= 1000  THEN 40
50  PRINT "KONEC SMYCKY IF-THEN"
```

Spuštěním programu zjistíte, že je první smyčka rychlejší (dříve ukončena) než druhá smyčka. Po zkompilování tohoto programu budou obě smyčky pracovat přibližně stejně rychle. Rychlost zkompilovaného programu nezávisí na počtu použitých povelů ani na struktuře programu, nýbrž pouze na samotných prováděných operacích. Vnitřně totiž budou obě smyčky provedeny skoro stejnou metodou (nároky na zpracování smyčky FOR-NEXT jsou poněkud vyšší).

Uvedené výsledky kompilátoru jsou získány metodami, které používá většina basicových kompilerů, ale BASIC 128 navíc používá další postupy pro urychlení programů. Jedná se o optimalizaci vzorců, použití celočíselných operací, rychlé zpracování znakových řetězců (stringů) a vybraným způsobem generování strojového kódu.

3.2 Optimalizace vzorců

Při počítání velkých výrazů jsou mezivýsledky ukládány počítačem do zásobníku (stack). Kompiler se snaží snížit počet mezivýsledků na nejnižší možnou míru. Tím bude zkompilovaný program nejen kratší, ale může být také urychlen jeho průběh. Není tedy třeba si všímat velkých výrazů, neboť tyto budou interpretovány pokud možno co nejrychleji. Kompiler mění (upravuje) dokonce výrazy, aby se dosáhlo rychlejšího průběhu výpočtu. Tak například průběh násobení dvou čísel s PDC je rychlejší než dělení. V mnoha případech je možné dělení s PDC nahradit násobením (převrácenou hodnotou). Kompiler používá samozřejmě jen takové postupy, aby výsledek vzorce zůstal zachován.

Podstatné ulehčení práce při programování matematických programů je schopnost BASICu 128 přepočítat vzorce, které obsahují mnoho konstant již během kompilování, a to částečně nebo úplně.

Příklad: 1Ø S = 1.414

Toto přiřazení je nepřehledné, ale bude překladačem rychleji zpracováno jako

```
10  S = SQR(2)
```

Po zkompilování pracují obě verse stejně rychle, ale druhá verse je přesnější a jednodušší k programování. Spočítání konstantních operací provádí kompiler také tehdy, když jsou tyto obsaženy ve velkých výrazech.

Případ, při kterém výpočet konstant kompilerem není možný, jsou znakové řetězce. Ne všechny z 256 možných znakových kódů je možné znázornit uvnitř uvozovek. Tyto kódy jsou přesto velice často potřebné např. při práci s floppy jednotkou. Přístup na jednu paměťovou buňku paměti RAM v disku může být například proveden:

```
PRINT#2, "M-R" + CHR$(0)+CHR$(4)+CHR$(10)
```

Přepočet tohoto řetězce není jen příliš dlouhý, nýbrž také pomalý. Ve zkompilovaném programu obsadí jen nutných 5 bytů a nebude třeba jej dále přepočítat, protože to již provede kompiler.

Podstatné zvýšení rychlosti při výpočtu vzorců je umožněno rychlými matematickými funkcemi BASICu 128. To ale nepředstavuje žádnou vlastní optimalizaci a bude to dále vysvětleno v kapitole 4.4.

3.3 Zpracování znakových řetězců (stringů)

Už se vám jistě nejméně jednou stalo, že se program náhle přerušil, několik sekund se nic nedělo, a pak program pokračoval korektně dál? Podstata tohoto jevu spočívá ve zpracování paměti basicového překladače. Při každém přiřazení znakového řetězce některé proměnné nebude totiž její předchozí obsah smazán, nýbrž zůstává zachován v paměti. Toto plýtvání pamětí má za následek, že paměť se časem úplně zaplní. Aby mohl program pokračovat, odstartuje se jedna rutina, která vyhledá a odstraní všechny nepoužité znakové ře-

tězce (tzv. Garbage Collection - odstranění smetí). Kompiler v tomto případě používá rychlejší rutinu než překladač. V extrémních případech je Garbage Collection kompilovaného programu maximálně po 1 sekundě ukončena a zpoždění programu je tedy nepatrné. Jestliže i to vás příliš zdržuje, pak se nabízí možnost řídit Garbage Collection přímo programem, a to tak, že do několika kritických míst vložíte funkci FRE(1) - to je dotaz na volné místo v paměti, přičemž je paměť vyčištěna od smetí.

BASIC 128 provádí ještě další optimalizace, co se týče rychlého zpracování řetězcových operací. Obzvláště u komplexních řetězcových výrazů je velmi účinný, jako např.

A$ = LEFT$(A$,X%-1) + B$ + MID$(A$,X%)

Řetězec (string) B$ bude zaveden do strinku A$ od posice X%. Basicový překladač vytvoří nejdříve několik částí, které pak navzájem spojí. Zkompilovaný program naopak je tak optimalizován, že vytvoří rovnou výsledný string, mezivýsledky odpadají. Proto zpracuje složitý výraz skoro stejně rychle jako jednotlivé funkce v něm zahrnuté.

3.4 Celočíselná optimalizace

V Commodore basicu existují dva různé typy proměnných. Celočíselné (integer) a s pohylivou desetinnou čárkou (real) - PDC. Proměnná, do které se ukládá číslo s PDC, může mít delší název, přičemž překladač si všímá pouze prvních 2 znaků. Do proměnné, která jako poslední znak ve svém názvu má znak "%", může být naproti tomu uloženo pouze celé číslo (tedy ne např. 1,23, ale pouze 1). Také zde registruje překladač pouze první dva znaky názvu a znak %. Navíc může proměnná, která je označena %, obsahovat hodnoty pouze v rozsahu celých čísel od -32768 do +32768 (bez desetinných míst). Ačkoliv se takové proměnné normálně používají zřídka, je tento typ proměnné v zásadě důležitější než typ s PDC. Normálně si

programátor v basicu ani neuvědomuje, že basicový překladač oba typy proměnných mezi sebou převádí, pokud je to potřeba, a často dokonce, i když k tomu nemá důvod.

Dále jsou uvedeny všechny povely, operace a funkce, u kterých k jejich provedení basicový překladač vyžaduje pouze celá čísla, a proto provádí odpovídající transformaci z PDC na typ integer:

ON GOTO, ON GOSUB, WAIT, LOAD, SAVE, VERIFY, POKE, CMD, SYS, OPEN, CLOSE, TAB, SPC, NOT, AND, OR, FRE, POS, PEEK, LEN, ASC, CHR$, LEFT$, RIGHT$, MID$, Indexová pole, všechny grafické, spritové a diskové povely a ještě další.

U následujících operací naopak používá překladač PDC a transformuje z typu integer na PDC:

Smyčková čísla, INPUT, PRINT, READ, IF, DEF, +, -, *, /, >, <, =, <=, >=, <>, SGN, INT, ABS, USR, SQR, RND, LOG, EXP, COS, SIN, TAN, ATN, STR$, VAL.

U následujících operací není transformace do PDC vždycky požadována, a přesto ji basicový překladač vždycky provádí:

Smyčková čísla, INPUT, PRINT, READ, IF, +, -, *, /, >, <, =, <=, >=, <>, INT.

Basicový překladač provádí navíc všechny výpočty ve formátu PDC. Jedná-li se o celočíselnou operaci, pak je před operací formát přetransformován a po ukončení výsledek opět převeden zpět. Každopádně je velká část všech transformací basicového překladače zbytečná. Navíc k transformacím používá překladač rutiny, která pracuje pomalu a obšírně. Dále operace s PDC jsou několikrát pomalejší než s integer. Není třeba dodávat, že basicový překladač, pokud má možnost volby, vždy použije PDC. Basicový program by mohl tedy pracovat mnohem rychleji, kdyby používal PDC jen když je to nevyhnutelně nutné.

Program zkompilovaný BASICem 128 maximálně využívá vždy integer. Přitom je však kompiler závislý na vaší spolupráci:

- kompiler nemůže při překládání (kompilaci) zjistit, jaký typ dat u které proměnné bude použit
- běžící program to může sice zjistit, je však vždy pomalejší než zkompilovaný, optimalizovaný program.

Z těchto důvodů BASIC 128 předpokládá, že se u všech proměnných jedná o PDC s výjimkou následujících případů:

- proměnné, které jsou označeny %
- proměnné, které kvůli odpovídajícímu kompilerovému příkazu budou převedeny na typ integer
- všechny proměnné, které se vyskytnou v programu, který je kompilován optimalizačním stupněm 2 (viz kapitolu 7).

V kapitole 2 jsme se již setkali s tím, že se vyplatí používat integer proměnné. V průměru je 90 % všech proměnných v programu typu integer. Basicový překladač ovšem nepřipouští integer proměnnou ve smyčce FOR-NEXT. V tomto případě můžete tento problém obejít pomocí kompiler-příkazu. Proměnné často používané ve smyčkách (např. I,J,K) rezervovat a vyčlenit pro tyto účely jako celočíselné proměnné typu integer kompilerovým příkazem. Je mnoho dalších důvodů k použití integer proměnných:

- jedna integer proměnná obsazuje 2 byty v paměti, zatím co proměnná s PDC obsazuje 5 bytů. Obzvláště u polí je to rozhodující výhoda (s tím se setkáváme i u basicového překladače)
- mnoho, obzvláště jednoduchých integer operací patří k povelové sadě mikroprocesoru 8502. Ty mohou proto být provedeny v několika mikrosekundách. Patří k nim např.: I%+1, I%-1, I%=2, I%=J%, I%=-J%.

Všechny programovací jazyky, které jsou realizovány přes kompiler, přísně rozlišují mezi proměnnými typu integer a PDC. Jestliže někdy přejdete na výkonnější programovací jazyk, jako je např. PASCAL, pak bude pro vás výhodné, že se

tím už nyní zabýváte. Existují dokonce programovací jazyky, které pracují pouze s celými čísly, např. Forth a Assembler.

3.5 Generátor strojového kódu

V zásadě je možné rozdělit kompiler do dvou různých tříd podle způsobu, jaké vytváří kódy. Často používaný postup je generování pseudokódů (Speedcode), které jsou pak zpracovány odpovídajícím překladačem. Výhoda těchto kódů je v tom, že jsou krátké. Zkompilovaný program proto bude podle velikosti originálu až o 50 až 80 % kratší než jeho originál. To je zvláště důležité, jestliže kompiler připojuje k hotovému programu sbírku programů (Runtimemodul), které potřebuje zkompilovaný program, tím jej však prodlouží. Nevýhoda P-kódu je v tom, že je nepatrně snížena rychlost běhu oproti programu ve strojo. kódu. P-kódy generované BASICem 128 jsou vyvinuty speciálně pro počítač C-128.

Alternativa ke generování P-kódu je přeložení originálního programu do strojového kódu. Zde se nabízí výhoda běhu programu s maximální rychlostí. Je však také podstatně delší než program pracující s P-kódy. Normálně se toto používá pouze u velkých počítačů, aby kompiler generoval strojový program, protože u velkých počítačů se šetří výpočetní čas, naproti tomu většinou nezáleží na velikosti obsazené paměti.

Většina basicových programů pro Commodore 128 neobsazuje úplně paměť počítače. Z toho vychází BASIC 128, když generuje krátké P-kódy nebo dlouhé strojové programy, nebo dokonce oba typy dohromady (kapitola 7). Je-li přeložen program do strojového kódu, odpadá při průběhu programu překládání P-kódů. Mikroprocesor C-128 typu 8502 potřebuje ke zpracování nějakého povelu 1 až 4 mikrosekundy (FAST-módus). Tento povel je oproti basicovému povelu primitivní, protože k provedení basicového povelu je potřeba celá řada povelů k procesoru. V mnoha případech se musí dokonce zavolat celé

podprogramy, aby byl proveden basicový povel. Strojový program je jen tehdy podstatně rychlejší než program s P-kódy, jestliže překlad může být proveden s málo povely, jinak v případě překladu s P-kódy je program silně brzděn. Basicové povely mohou být přeloženy do několika strojových povelů jen tehdy, jestliže se jedná o jednoduché povely. Jednoduchými povely se rozumí takové, které poznáme podle vysoké prováděcí rychlosti, jako např. všechny operace s integer proměnnými a celými čísly. Většina integer operací se nechá provést s několika strojovými povely. V těchto případech bude program rychlejší, poněvadž odpadá překlad P-kódů, program poběží přímo ve strojovém kódu. Lze říci, že strojový program pracuje rychleji než program s P-kódy, jestliže samotný program s P-kódy pracuje rychle. U programů, které používají skoro výlučně komplexní operace s FDC (SIN, SQR aj.), nedosáhneme vygenerováním strojového programu žádného výrazného zvýšení rychlosti. Otázku, jak poznat, je-li lepší optimalizovat program pomocí strojového kódu, lze odpovědět tak, že nejlepší je to vyzkoušet. Jedině využitím všech možností se nechá většina programů znatelně urychlit. Dva malé příklady byly již ukázány v kap. 2.

Zapnutí generátoru strojového kódu se provede volbou "A" ve vývojovém paketu. Do vývojového paketu se dostanete stisknutím klávesy <3> po startu kompileru BASIC 128. Pomocí klávesy "A" můžete generovat kódy pro mikroprocesory 6502, 6510 a 8502 a potom pomocí <RETURN> se vrátit zpět do hlavního menu. Program, který byl kompilorem nyní vygenerován, obdrží v directory označení "M-" před svým jménem, podobně jako byla u P-kódů předpona "P-". Jinak je obsluhování kompileru identické jako u použití generátoru P-kódů.

3.6 Optimalizace časového průběhu

V mnoha programech je velmi těžké označit pro kompiler všechny proměnné v programu, které jsou pouze typu integer,

např. proto, že se původně nepočítalo s kompilováním, nebo použití proměnných je velmi komplexní. Kompiler pak už nemůže provést příliš mnoho optimalizací. Přesto může program běžet největší možnou rychlostí, bude-li optimalizace provedena systémem BASICu 128. Tento způsob optimalizace sice není tak docela efektivní jako přímá optimalizace pomocí kompileru, ale účinně působí na optimalizaci časového průběhu na každý program, i když tento nebyl pro kompilování přizpůsoben. Optimalizace časového průběhu není normálně u kompilerů prováděna, je to specialita BASICu 128.

4. DETAILY

4.1 Způsob práce kompileru

Po volbě bodu "1" v hlavním menu a zadání názvu programu, který má být kompilován, zahájí kompiler po natažení programu překládání programu (kap. 1). Kompiler zde rozlišuje dvě pracovní fáze, označené Pass 1 a Pass 2. Jsou prováděny postupně za sebou po odstartování kompilace.

Pass 1

Program bude přeložen, optimalizován a budou vytvořeny odpovídající kódy (P-kódy nebo strojový jazyk). Kompiler přitom udává na obrazovce proběhlá čísla řádků a dvojtečkou oddělené povely na jednom řádku. Narazí-li kompiler na povel, který mu patří (kompilerový povel), začínající REM @, pak napíše "R". Povely nebo funkce, které nejsou součástí Commodore basicu (např. různá basicová rozšíření), napíše "E".

Pass 2

Vygenerované kódy budou ještě jednou kompilerem zpracovány a doplněny, dále bude na program připojena část Runtimemodul

a do programu budou zavedeny DATA řádky. Během této činnosti podává kompiler na obrazovce následující hlášení:

Data-Code: Od této adresy v paměti Bank Ø leží Data řádky ve zkompilovaném programu.

Objekt-Code: V tomto rozsahu Bank Ø leží vlastní program.

Strings: Tento rozsah je úplně volný v Banku 1. Zkompilovaný program jej používá k uložení znakových řetězců (stringů).

Extentions: Jsou-li v programu použity povely nějakého basicového rozšíření, pak kompiler vyjmenuje jejich počet.

Errors: Jestliže se v programu vyskytují chyby, pak je kompiler vyjmenuje v odpovídajících řádcích a vydá před ukončením části Pass 2 výpis všech řádků, ve kterých se chyba vyskytuje.

Warnings: Kompiler umí vydat více druhů varování, na konci Pass 2 bude vyjmenován jejich počet.

4.2 Chybová hlášení

Chybová hlášení jsou rozdělena do dvou skupin: Programové chyby a varování.

Programové chyby: Chyby, které zapříčiňují, že kompiler nemůže povel přeložit, jsou hlášeny již v průběhu kompilování. Kompiler vydá to samé hlášení (popis chyby) jako basicový překladač, když narazí na tuto chybu.
Kompiler sděluje následující chyby:
Syntax - stejný význam jako v basicu
Redim'd Array - - " -
Type mismatch - - " -
Bod Subscript - přístup do pole obsahuje špatné indexové číslo oproti nadimensovanému poli
Undef'd statement - GOTO-/GOSUB- veden na neexistující řádek. Tato chyba je odhalena až v Pass 2.

Out of memory - program a proměnná se nevejdou do paměti. Normálně se to stává jen když budou s vývojovým paketem změněny hranice paměti

Runtime - kompiler zkouší přepočítat vzorce (tak daleko, jak je to možné) během kompilace (hlavně s konstantami). Nemůže-li tyto vzorce nebo výrazy vypočítat, pak kompiler vydá toto hlášení. Většinou se jedná o chyby, které se rozeznají až v průběhu činnosti programu. Například konstanta dělená nulou
1Ø A = 1/Ø

Varování:

Kompiler varuje před některými chybami obsluhy a programovými chybami. Mohou být vydána následující varování:

TRACE NOT FOUND	viz kapitola	4.8
TRACE NOT USED	- " -	4.8
BEND WITHOUT BEGIN	- " -	5.2
ILLEGAL BEGIN	- " -	5.2
ILLEGAL ELSE	- " -	5.2
ILLEGAL BEND	- " -	5.2
ILLEGAL OVERLAY	- " -	6.1
LOAD ONLY IN OVERLAY	- " -	6.1
POINTER WITHOUT BANK	- " -	8.3

System Error 1 až System Error 9:
Tato hlášení operačního systému vznikají např. při vypnutém disku a dalších hrubých chybách obsluhy. Je možné se setkat také v průběhu kompilování s chybovým hlášením disku, např. při přeplněné disketě.

System Error 10:
Stane-li se, že kapacita paměti C-128 nevystačuje, aby mohl být program kompilován, pak je vydáno toto hlášení.

System Error 11:
Toto hlášení bude vydáno, je-li počet odhalených chyb příliš velký, například jestliže se nejedná o basicový program (program, který byl předložen ke zkompilování, je např. napsán ve stroj. kódu).

System Error 2Ø:
Toto hlášení je vydáno, je-li disketa vašeho BASIC 128 poškozena, nebo disk pracuje nesprávně, nebo když kompiler nebyl odstartován s originální disketou. Eventuálně někdy pomůže znovu odstartovat kompiler. Skoro ve všech případech může kompiler po odhalení chyby dále pokračovat v činnosti, a tak objevit další chyby.

Pozor! Program, ve kterém se vyskytuje chyba, je přirozeně možné použít jen s omezením. To platí hlavně o řádcích, ve kterých se nachází chyba. Nalezené chyby je třeba odstranit a program pak nechat znovu kompilovat.

Průběhové chyby:

Mnoho chyb nemůže být kompilerem objeveno, protože se vyskytnou až při běhu programu. Tato chybová hlášení mají ten samý význam jako u basic-překladače.

Out of Memory - toto hlášení buď znamená, že rozsah pro znakové řetězce nepojme všechny stringy, nebo že je zásobník pro FOR, GOSUB a závorky úplně plný. Použití příliš mnoha proměnných bude již kompilerem odhaleno.

Bad Subscript - přístup do indexovaného pole leží mimo hranice vymezeného pole. Špatně zadané indexové číslo bude kompilerem odhaleno.

Formula too complex - toto hlášení vydá kompiler, je-li výraz znakového řetězce příliš hluboce škatulkován ("hnízděn"), s čímž se normálně nesetkáváme. Zkompilovaný program nikdy nevydá toto hlášení, jestliže toto hluboké hnízdění zpracoval.

Illegal Quantity- toto hlášení dává také basicový překladač, jestliže provedení povelu vlastně není možné, např.
1Ø DRAW, 1ØØ,1ØØ TO -5Ø,+5Ø
Překladač vydá chybové hlášení, protože třetí parametr obsahuje nedovolenou hodnotu (-5Ø).

Překladač rozeznává povolení na základě koncových souřadnic jako relativní souřadnice k výstupnímu bodu, podobně povel MOVSPR. Negativní hodnoty pak rovněž dávají smysl a budou pak kompilerem akceptovány.

Při převodu z PDC na integer hodnoty při počítání s celými čísly se může stát, že výsledek neleží v rozsahu celých čísel. Kompiler nastaví počítání s integer operacemi, jestliže nastane případ, že při běhu programu by mohlo dojít k hlášení překladače "Illegal Quantity". Nemůže-li tento stav u vašeho počítače nikdy nastat, pak si můžete být jisti, že také u zkompilovaného programu nedojde k překročení rozsahu. Z těchto důvodů neprovádí zkompilovaný program u integer výpočtů žádné přezkušování rozsahu (z důvodu rychlosti programu). Mělo-li by u mezivýsledků přece jen někdy dojít k překročení rozsahu a konečný výsledek opět leží v rozsahu integer, pak bude výsledek také s integer operací korektně sdělen. Toto může být účelné např. u povelu POKE (kap. 7).

Povel INPUT basicu 7.0 obsahuje několik zvláštností při chybových zadáních. Zadání stringu na místě, kde se očekává číslo, vede k hlášení "Redo from Start" a k úplnému opakování vstupního povelu. Zkompilovaný program pracuje stejně. Při neúplném zadání požaduje program další zadání údajů (se dvěma otazníky, kompiler i překladač). U chybějícího zadání (není nic napsáno a stiskne se RETURN) bude povel INPUT naopak ignorován a proměnná nebude změněna (jestliže před tímto průchodem obsahovala proměnná nějakou hodnotu, zůstává jí i potom). Toto je zvláštnost C-128 a C-64. U jiných Commodore počítačů tuto vlastnost použitý basic nemá. BASIC 128 byl vyvinut speciálně pro C-128 a uznává proto tuto někdy užitečnou možnost. U některých zadání, kdy je zadáno příliš mnoho hodnot, vydá překladač hlášení "Extra ignored". Zkompilovaný program ignoruje tyto hodnoty bez (často rušivého) hlášení.

Chyby disku:

Setká-li se kompiler při práci s chybou vyslanou z disku, ohlásí to a ukončí kompilování.

Význam diskových chybových hlášení je vysvětlen v manuálu k disku. Hlášení jako "Read error","Write error", "No channel" a "Write file open" ukazují na poškozenou disketu, která musí být vyměněna.

Při práci s BASICem 128 a basicovým překladačem se může stát, že původní basicový program musí být vylepšen a znovu uložen. To je možné většinou s následujícím povelem

DSAVE "(:) : název"

Na základě chyby v operačním systému disku 1541 může se někdy stát při použití tohoto povelu, že dojde ke ztrátě dat na použité disketě. Toto je přirozeně nezávislé na BASICu 128, je-li použit nebo ne. Můžeme vám pouze doporučit následující postup

SCRATCH "název": DSAVE "název"

(název = jméno souboru)

Došlo-li při práci s kompilerem k RESETu (tlačítkem RESET na pravé straně C-128 vedle vypínače) nebo vypnutím byl proces přerušen, pak se doporučuje provést na pracovní disketě (která byla v tomto okamžiku v disku) povel COLLECT, aby se zabránilo chybám na disketě.

4.3 Výpis řádků

Setká-li se při běhu program s chybou, pak bude vydáno příslušné chybové hlášení. Příkazem na řádek může program vydat pouze adresu v paměti, jelikož zkompilovaný program již není organizován systémem řádků (výjimka viz 4.8). Ke zjištění řádku, ve kterém je chyba, je potřeba výpis řádků, který vygeneruje kompiler. K tomu slouží bod "D" ve vývojovém paketu, kterým bude zapnuto vytvoření výpisu řádků.

U programu, který dosud nebyl vyzkoušen, by měl být vždy vygenerován výpis řádků.

K použití tohoto výpisu několik poznámek:

- poznačte si adresu v paměti, kde došlo k chybě
- natáhněte výpis řádků pomocí DLOAD "Z-název"
- vypište si list požadovaného místa pomocí LIST-adresa chyby
- na pravé straně výpisu naleznete odpovídající řádek, jemuž byla přiřazena adresa na levé straně stránky. Poslední vyjmenovaný řádek obsahuje chybu. Jestliže se chyba nenachází přímo na uvedené adrese, může ve velmi vzácných případech být na konci předchozího nebo na začátku následujícího řádku. Toto platí obzvláště pro programy, které byly přeloženy do strojového jazyku
- vylepšete (opravte) program a znovu jej zkompilujte.

4.4 Rychlé výpočty s pohyblivou desetinnou čárkou

BASIC 128 používá PDC pro funkce TAN, ATN, SIN, COS, ↑, EXP, LOG a SQR, které pracují rychleji než rutiny basicového překladače. Dříve byl tento druh rutin používán u výkonných 16- nebo 32 bitových počítačů s PDC. U BASIC 128 jsou tyto rutiny k dispozici také pro Commodore C-128, a je tedy možné s tímto počítačem řešit odpovídající problémy.

Výpočetní přesnost funkcí kompileru je vyšší než u basicového překladače. V následující tabulce je uvedena pro každou funkci její průměrná spotřeba taktovacích cyklů (hodinových impulsů) a dále urychlení proti basicovému překladači. Všechny hodnoty se vztahují jen na funkce samotné a ne na eventuální příslušné programové části.

FUNKCE	Takt cykly	Urychlení
TAN	11 6ØØ	4,6
ATN	8 7ØØ	4,9
SIN	16 6ØØ	1,7
COS	16 6ØØ	1,7
X↑Y	15 9ØØ	3,5
EXP	9 4ØØ	2,9
LOG	8 6ØØ	2,7
SQR	4 9ØØ	1Ø,8
SIN a COS současně	17 ØØØ	3,3

Pro srovnání:

Jedno jednoduché dělení spotřebuje asi 3ØØØ taktcyklů. C-128 provádí ve FAST módu 2 milióny taktcyklů za sekundu. Funkce SIN a COS jsou nejpomalejší. V mnoha případech jsou ale potřebné hodnoty sinus a cosinus téhož úhlu. V tomto případě je možné, bezprostředně po vypočtení sinusu nalézt cosinus ke stejnému úhlu pomocí funkce USR, což se děje ve velmi krátkém čase. Není možné však v tomto případě přirozeně použít USR jinak. Mezi SIN a USR (s cosinem) nesmí být provedeny žádné další operace (ani PRINT).

Příklad: S = SIN(X):C = USR(Ø):T = S/C
Nyní dostáváme v proměnné S sinus, v C cosinus a v T tangens od X.

Má-li být takový program před kompilováním otestován překladačem, pak je možné USR(Ø) nahradit USR(X), protože kompiler parametr USR funkce v tomto případě neinteresuje. Aby USR funkce překladače odpovídala kompileru, je před startem programu s překladačem nutné odeslat následující povely:

POKE 4633,9 : POKE 4634,148

Rutiny pro PDC zabírají asi 3 k paměti. Kompiler zavádí tyto rutiny jen do těch programů, které je skutečně při činnosti potřebují. Jestliže chcete kvůli úspoře místa v paměti zabránit, aby kompiler použil svoje rutiny, pak může používat

stejné rutiny jako používá basicový překladač. Stiskněte ve vývojovém paketu klávesu "E" a potom <6>. Nyní dojde k vypnutí (odpojení) aritmetického modulu. Pomocí <RETURN> se dostanete zase zpět.

Všeobecně ke znázornění s PDC v basicu 7.Ø:

Interpreter i kompiler pracují s přesností 32 bitů, čistě počtářsky to odpovídá decimální přesnosti asi 9,6 místa. Mezivýsledky jsou počítány s přesností na 4Ø bitů. Čísla jsou ukládána v binárním tvaru s PDC, což dovoluje vysokou výpočetní rychlost. Toto znázornění s PDC ovšem má také svoje nevýhody, které byste měli znát:

- zlomky, které v decimálním systému se lehce znázorní např. 1/1Ø, nemohou být v binárním systému znázorněny s konečným počtem míst, což vede k výpočetním chybám
- chyby vznikající zaokrouhlením mezivýsledků nelze srovnat s decimálním zaokrouhlením, protože není prováděno ani na 9 ani na 10 míst, nýbrž někde mezi tím. Obzvláště pro obchodnické výpočty jsou oba body nevýhodou
- čísla, která jsou tak malá, že překračují znázornitelný rozsah, budou zaokrouhlena na nulu. To je v mnoha případech rozhodující chyba, zvláště při následujícím násobení nějakým celým číslem
- u narůstající velikosti čísel bude také větší rozdíl mezi jedním číslem a následujícím velkým číslem. Například není možné korektně přičíst k číslu 1E1Ø hodnotu 1.
- zkompilovaný program počítá o něco přesněji než program překládaný basicovým překladačem.

Příklad PRINT 3↑4

4.5 Dimensování polí

Hranice polí musí být známé již během kompilování, a to z následujících důvodů:

- pole mohou ležet také v Banku Ø, odkud odeberou určitou kapacitu paměti. Na to následuje komplikované zpracování paměti, které může převzít pouze kompiler
- přístup do polí musí být pokud možno rychlý a přímý. Kompiler k tomu potřebuje adresu pole a číslo BANKu.

V mnoha případech jsou hranice pole v průběhu kompilování neznámé, např. u následujícího povelu DIM

```
1Ø   INPUT X : DIM A(X)
```

V tomto případě vydá kompiler na obrazovce název pole následovaný závorkou a otazníkem. Zadejte pak maximální velikost pole. V našem příkladě to bude očekávaná hodnota pro X.

U více dimensací se pokaždé kompiler zeptá na neznámé hodnoty. Oddělte přitom zadávané hodnoty vždy klávesou <RETURN>.

Ačkoliv pro kompilovaný program je normálně k dispozici dostatečné místo v paměti, je otázka na maximální index pole často také otázkou na potřebné místo v paměti. V kapitole 8 naleznete výpis různých typů dat a jejich nároky na paměť.

Při sestavení programů, které pracují s mnoha poli, se setkáme s tím, že překládanému basicovému programu je k dispozici méně místa v paměti (Bank 1) než zkompilovanému programu (Bank 1 a částečně Bank Ø). Je užitečné vyzkoušet program nejprve s malými poli a tento zkompilovat

```
1Ø   N = 5ØØØ : DIM X%(N)
```

Při kompilování se ptá kompiler na maximální index pole, kde se pak může zadat největší hodnota.

4.6 Povely v direkt módu

Některé basicové povely není možné normálně použít uvnitř basicového programu, neboť zde nedávají žádný smysl. Jedná se o následující povely:

LIST, SAVE, VERIFY, CONT, DSAVE, DVERIFY, AUTO, DELETE, RENUMBER

Povely DSAVE, SAVE, DVERIFY a VERIFY budou kompilerem správně přeloženy. Uložení programu s nimi však není možné, protože zkompilovaný program se sám v paměti posouvá a mění. Zbývající povely nemají v kompilovaném programu žádný smysl a budou kompilerem ignorovány. Například zkompilovaný program již nemůže být listován, protože to již není basicový program. Kompiler v tomto případě vydá hlášení "DIRECT MODE ONLY".

4.7 Integer - smyčky

Integer proměnné normálně nemohou být použity ve smyčkách FOR-NEXT, ale s odpovídajícím kompilerovým příkazem nebo optimalizačním stupněm č. 2 to je možné. Integer smyčky jsou nejen rychlejší, ale obsazují také méně místa v paměti, a proto mohou být hlouběji hnízděny. Kromě toho je zpracování a přístup na smyčkové proměnné uvnitř smyčky rychlejší. Následující příklad sice používá integer smyčku, ale přesto může být zpracován překladačem

```
 5   REM @ I = I,J
10   FOR I=1 TO 100:PRINT I
20   FOR J=1 TO 50
30   NEXT J,I
```

V integer smyčce může být zadána jako u normální smyčky hodnota pro STEP, ale postupujte při tom velmi opatrně, protože hodnota STEP bude v tomto případě zkompilována a převedena na celé číslo. To může vést např. pro STEP 0.5 na STEP 0 a smyčka přejde na tzv. nekonečnou smyčku, ze které není úniku.

4.8 Strukturové značení řádků a povelů

Při kompilování basicového programu do P-kódů nebo strojového jazyka jsou ztraceny informace o řádkové nebo povelové struktuře. Systém nemůže již zjistit, které řádky byly provedeny a kde nějaký povel končí a kde začíná. Některé povely basicu 7.Ø však právě tyto informace potřebují a jsou proto normálně nekompilovatelné. BASIC 128 umí tyto povely přesto zkompilovat. Kompiler ovšem musí znát strukturu značení těchto povelů, aby je mohl zavést do kompilovaného programu. Jedná se o povely:

TRON: řádkové a povelové označení v rozsahu, ve kterém má být hledáno

RESUME a RESUME NEXT: povelové označení v rozsahu, ve kterém se vyskytuje chyba.

RESUME a konstanta s číslem řádku: označení není nutné

EL-proměnná: označení řádku pro rozsah, ve kterém je chyba. Proměnná EL může být přečtena po přerušení programu. V tomto případě si ušetříte jeden výpis řádků.

COLLISION: označení povelu v rozsahu, ve kterém má být povel účinný

Jiné povely s nekonstantním číslem řádku, např.

TRAP N*1Ø : označení řádku v rozsahu, ve kterém možné číslo řádku leží

Výjimka: RESTORE nepotřebuje v žádném případě strukturové značení.

Existuje více možností, jak přimět kompiler ke strukturovému značení. Například kompilerový příkaz

```
1 REM @ TLC
```

Tento kompilerový příkaz zamezí všem problémům, které u shora jmenovaných povelů mohou vzniknout. Ovšem program díky tomuto příkazu bude pomalejší a delší. BASIC 128 nabízí možnost

strukturové značení pružně řídit. Pomocí několika stisknutí klávesy "J" ve vývojovém paketu dosáhnete následující postavení:

```
TRACE  :  OFF              ;  žádná značení, normální stav
TRACE  :  LINE             ;  pouze značení řádků
TRACE  :  COMMAND          ;  pouze značení povelů
TRACE  :  LINE & COMMAND   ;  obě značení (řádky + povel)
```

Aby obdržely označení jen ty rozsahy programu, ve kterých je to nutné, existují kompilerové příkazy k potřebnému řízení:

```
REM @ TB  nebo  REM @ TLC   ;  obě označení
REM @ TL                    ;  jen označení řádků
REM @ TC                    ;  jen označení povelů
REM @ TO                    ;  značení vypnout
```

Všechny kompilerové příkazy působí přirozeně teprve tehdy, až se na ně narazí v programu.

Příklad:

```
  1Ø   TRAP 1ØØØ
  2Ø   FOR I= -5 TO 5
  3Ø   PRINT 1/I
  4Ø   NEXT
  5Ø   END
 1ØØØ  PRINT "NENI DEFINOVANO": RESUME NEXT
```

Tento program používá povel RESUME NEXT, k jehož provedení systém potřebuje v řádku 3Ø povelové označení, protože chyba se vyskytne v řádku 3Ø (pro I=Ø). Můžete povelové značení například zapnout pomocí vývojového paketu. Uvnitř velkého programu by bylo efektivnější zavést následující kompilerové příklady:

```
25   REM @ TC
35   REM @ TO
```

Poznámka: Označení řádku může obsahovat označení povelu, jestliže v každém řádku ve zvoleném rozsahu stojí jen jeden povel.

Kompiler částečně varuje před chybovým použitím strukturového značení.

"TRACE NOT USED" : označení nebylo nutné (event. mimo TRON) a muselo být v celém programu vypnuto

"TRACE NOT FOUND": v celém programu nebylo zavedeno žádné značení, ačkoliv program obsahuje jeden nebo více povelů, které značení vyžadují. TRON nebude přezkoušen.

4.9 Uložení a natažení nastavení kompileru

Všechna nastavení kompileru, která byla zvolena pomocí vývojového paketu a jeho podmenu, mohou být jako soubor uložena na disketu a pak opět natažena. Toto se provede volbou bodu "I" ve vývojovém paketu. Pak stiskněte <S> pro uložení (Speichern) nebo <L> pro natažení (Laden) a pak zadejte název (jméno) souboru.

K uložení souboru, který má být pak možné natáhnout i z bodu "2" hlavního menu, musí být zadán název "Bl28". Aby byly rozlišeny jednotlivé soubory na disketě, které patří BASICu 128, obdrží tento soubor s nastavením kompileru předponu "E-" (Einstellung), toho však si nevšímejte.

4.10 Nový start kompileru a zkompilovaného programu

Byl-li zkompilovaný program odstartován, přesune se sám do místa v paměti, na které byl kompilerem generován. Poté co se běh tohoto programu ukončí, nalézá se stále ještě v tomto místě paměti, ale nemůže být znovu odstartován pomocí RUN, nýbrž musí být znovu natažen. Existuje však možnost odstartovat program bez nového natažení. K tomu slouží program "START" na vaší programové disketě BASIC 128. Uložte tuto disketu do disku a zadejte

RUN "START"

Aktuální, zkompilovaný program, který je v paměti, bude znovu odstartován. Přirozeně si můžete poznačit povely programu "START" a tyto zadat přímo.

Kompiler může být rovněž natažen pomocí programu "START". K tomu je potřeba splnit následující podmínky:

- Originál disketa BASIC 128 je vložena v disku.
- Kompiler byl již použit a korektně ukončen (tedy ne RESETem nebo vypnutím C-128).
- Po použití kompileru nebyl odstartován zkompilovaný program, jinak bude tento program odstartován místo kompileru.
- Před novým startem programu smí být basicový program natažen, změněn a opět uložen. Tento program nesmí podstatně překročit délku 15 kbyte (60 disketových bloků), jinak dojde k přepsání kompileru, který je uložen v paměti. Při použití grafického rozsahu smí mít tento program pouze délku 6 kbytů.

Jestliže tyto uvedené podmínky nebyly splněny, musí se kompiler znovu odstartovat podle postupu, který je uveden v kapitole č. 1.

4.11 FAST-módus

Povel FAST zvyšuje 2x rychlost počítače C-128. Samozřejmě bude tímto povelem také 2x zvýšena rychlost zkompilovaného programu, přičemž zůstává zachován poměr rychlosti mezi kompilerem a překladačem. Jestliže byl zkompilovaný program odstartován v aktivovaném 80-ti znakovém módu, pak se program automaticky přepne do režimu FAST. Pomocí povelu SLOW v programu může být zpětně přepnut do SLOW.

Kompiler sám může také pracovat v režimu FAST a při použití 80-ti znakového módu se to stane automaticky. Při použití 40-ti znaků na řádek je možné přepnout kompiler do módu FAST pomocí bodu "F" ve vývojovém paketu. V tomto případě zapne kompiler zobrazení na obrazovce pouze pro důležitá hlášení. Kompilační rychlost nebude při FAST módu nijak obzvlášť zvýšena,

protože překládací rychlost kompileru (pozor, nezaměňovat překladač basicu, který je umístěn v ROM paměti počítače C-128) závisí na rychlosti použitého disku.

4.12 Přerušení zkompilovaného programu

Přerušení zkompilovaného programu je jen zřídka možné pomocí klávesy <STOP>, např. při použití vstupního nebo výstupního povelu. Pokračování s CONT není možné. Má-li být zabráněno tomu, aby bylo možné program přerušit, může se to provést pomocí povelu TRAP. Přitom si všimněte, že uvnitř rutiny obsluhující chybu je TRAP povel inaktivní. Přerušený program pomocí <RESTORE> může být opět spuštěn pomocí povelů

POKE 792,51 : POKE 793,255

Přitom dojde ovšem k odpojení interfejsu RS-232.

5. ZVLÁŠTNÍ POVELY basicu 7.Ø

5.1 Obsloužení chyby

Povel TRAP basicu 7.Ø slouží k tomu, aby nalezl chyby, které se vyskytují v programu, a nikoliv k tomu, aby řídil průběh programu. Toto nebude u BASICu 128 ve všech případech zohledněno. Například kompilerem bude odhalena chyba a musí být odstraněna, a to tak, že se tato chyba v průběhu programu již nevyskytne. Není-li třeba po kompilaci žádných chybových hlášení, jsou části rutin obsluhující chyby přebytečné (viz také kap. 4.2).

Programy, ve kterých jsou použity povely RESUME a RESUME NEXT (ale ne RESUME s číslem řádku) musí být kompilovány se zapnutým strukturovým značením povelů (viz kap. 4.8).

5.2 IF ... THEN ... ELSE, BEGIN ... BEND

Překladač (basicový) obsluhuje strukturované příkazy ELSE, BEGIN a BEND ne vždy ve smyslu programové struktury. Kompiler se toho rovněž drží z důvodů kompatibility k překladači. V těchto případech ovšem kompiler vydá následující varování:

BEND WITHOUT BEGIN: příkaz BEND nenásleduje po BEGIN a je přebytečný nebo chybný.

ILLEGAL BEGIN: Na tomtéž řádku před BEGIN se nalézá THEN nebo ELSE a k těmto nenásleduje BEGIN. Obě struktury nejsou správně hnízděny, naopak se kříží, což je ve skoro všech případech hrubá programová chyba.

ILLEGAL ELSE: Jeden výraz ELSE se vztahuje na 2 nebo více předcházejících IF ... THEN příkazů. V těchto případech bude část po ELSE provedena vždy, když jedna z obou IF částí probíhá negativně. Tento postup skoro vždycky není ve smyslu požadavku programátora. Pomocí vložení BEGIN ... BEND může být vyznačeno správné hnízdění.

ILLEGAL BEND: BEND rozvíjí svoji činnost teprve na konci řádku nebo na následujícím ELSE. BEND, který není přímo před koncem řádku nebo před ELSE, je tím pádem špatně umístěn.

5.3 Nezadokumentované povely a povelové možnosti

Basic 7.Ø obsahuje některé povely a možnosti povelů, o kterých není žádná zmínka v manuálu C-128. Dále budou uvedeny některé povely ve formě, kterou neuvádí manuál. BASIC 128 se však i v těchto případech drží kompatibility s basico-

vým překladačem C-128. Ovšem není vyloučeno, že objevíte další nezadokumentované povely, které jsou pro kompiler neznámé.

- Skoro všechny grafické povely mohou pracovat s relativními souřadnicemi (viz také kap. 4.2), jako je to např. možné u MOVSPR.
- Grafické souřadnice mohou být zadávány také s mezerou nebo úhlem, namísto čárky je v těchto případech nutno použít středník.
- MID $ smí stát také vlevo od přiřazovacích znaků.
- SYS obsahuje ty samé parametry jako povel RREG (viz také kap. 8.3).

Některé povely basicu 7.Ø se nechovají vždy korektně. Kde je to nutné v zájmu kompatibility, přebírá kompiler toto nesprávné chování basicového překladače (viz. 5.2), ale nelze to ve všech případech zaručit.

5.4 Basicové rozšíření

Má-li být zkompilován program, který obsahuje povely, které nepochází z basicu 7.Ø, pak je to třeba sdělit kompileru volbou bodu "H" ve vývojovém paketu. Ovšem jsou nutná ještě další nastavení kompileru. Do jaké míry vámi použité rozšíření basicu také rozšíří kompiler a která nastavení kompileru jsou potřebná, to se dozvíte z manuálu k vašemu basicovému rozšíření.

K přizpůsobení basicového rozšíření na kompiler nabízí systém BASIC 128 následující postupy k použití:

Funkce:

Při řešení některé rozšiřující funkce (Token kód) volá systém rutinu na adrese $ ACE v BANKu Ø. Druhý byt Token se musí nalézat v Akku. Celočíselná (integer) hodnota funkčního parametru se musí nalézat v paměti $ 47, $ 48 a tam lze také očekávat výsledek.

Povely:

Setká-li se program s rozšiřujícím povelem (jako Token--kódem), pak předá systém řízení rutině, která začíná v paměti na adrese $ AC6 v BANKu Ø. Povelový parametr si může basicové rozšíření vyzvednout obvyklými rutinami překladače. Mohou být použity rutiny CHRGET a CHRGOT.

Basicové rozšíření je však zodpovědné za to, že pomětové buňky $ ACE a $ AC6 budou s rutinou správně obsazeny.

6. OVERLAY a RUNTIME MODUL

6.1 Kompilování s Overlay-pakety

V basicu Commodore 128 existuje možnost pomocí povelů LOAD nebo DLOAD natáhnout program a odstartovat jej, přičemž předchozí obsah všech proměnných zůstává zachován. Tato verse povelu LOAD bude provedena, jestliže je povel použit uvnitř programu a způsobí Warm-Overlay. Povel RUN "název" používá naproti tomu Kalt-Overlay a dojde ke smazání všech proměnných.

Kalt-Overlays mohou být kompilovány jako jednotlivé programy, ovšem nesmí být vypnut Runtime-modul, poněvadž bez tohoto modulu nemůže být program odstartován pomocí RUN. Povel RUN "název" může také odstartovat z kompilovaného programu nekompilovaný program.

Při kompilování programů, které používají mechanismus Warm-Overlay, je třeba dát pozor, že tyto programy nemohou být kompilovány obvyklým způsobem.

Ke kompilování nějakého programu z Overlay paketu potřebuje kompiler tabulku symbolů, ve které jsou uvedeny všechny použité proměnné programového paketu a jejich rozmístění v paměti. K sestavení této tabulky symbolů slouží

Overlay-Pass 1 (nezaměňovat s Kompilerem Pass 1).

Před provedením Overlay-Pass 1 musí být smazán soubor "S-OVERLAY" na vaší pracovní disketě, jestliže na ní existuje.

Overlay-Pass 1:

Kompiluje všechny programy programového paketu jednotlivě. Před startem kompilování s <1> nebo <RETURN> stiskněte klávesy <4> (OVERLAY) a potom <1> (Pass 1). Všechny možnosti vývojového paketu (bod menu č. 3) je možné nadále používat. Kompiler nyní nevytváří žádný hotový program, proto taky pracuje rychle. Bude pouze vygenerována tabulka proměnných a po každém kompilerovém průchodu bude doplněna. Dále uloží kompiler všechna kompilerová nastavení, tato si pak natahuje v OVERLAY-Pass 2.

Důležité je, aby první kompilovaný program byl také startovací program Overlay-paketu, protože tento program jako jediný může být odstartován v direkt módu. Dále může být odstartován startovací program jen pomocí RUN "název" (a ne s DLOAD "název") z jiného programu. V nouzovém případě musí být napsán odpovídající program pro Overlay-paket, např. menuprogram.

Poté co je tabulka symbolů doplněna, mohou být programy Overlay-paketu kompilovány. K tomu slouží Overlay-Pass 2.

Overlay-Pass 2:

Zkompiluje všechny programy paketu stisknutím (před kompilováním) klávesy <4> (Overlay) a pak <2> (Pass 2). Kompiler natáhne před kompilováním odpovídající Overlay-tabulku a kompilerové nastavení. Jakmile budou všechny programy s Overlay--Pass 2 zkompilovány, může být paket odstartován.

Při kompilování Overlay-paketů je třeba dát pozor na následující poznámky:

- Zkompilované programy musí být přirozeně ještě přejmenovány

na správný název, protože kompiler před název kompilovaného programu připojí "P-" resp. "M-".

- Mají-li být kompilovány velké Overlay-pakety, pak musí být jednotlivé programy rozděleny na více disket, protože kompilované programy potřebují místo na disketě. Kompiler natáhne Overlay-tabulku již před zadáním názvu programu. Tabulka může být natažena ke kompilovanému programu také z jiné diskety. Důležité přitom je, že Overlay-Pass 1 natáhne naposled generovanou tabulku (poslední průběh kompileru). V Overlay-Pass 2 je vždy natažena poslední generovaná tabulka v Overlay-Pass 1.
- Pomocí bodu "M" z menu může být odeslán povel do disku. Tímto způsobem můžete např. smazat již zkompilované programy (přirozeně jen ty programy, od kterých existují kopie) a tím získáte místo na disketě.
- Všechna pole musí být dimensována ve start programu, obzvláště když kompiler vydal odpovídající varování.
- Kompiler převede povely DLOAD a LOAD automaticky na odpovídající povel BLOAD. Ačkoliv kromě startovacího programu žádný program paketu neobsahuje Runtime-modul, není třeba si všímat příkazu v odstavci 6.2.
- Uvnitř Overlay-paketu smí být natažen start programu jen s RUN "název" a ne, jak je obvyklé, pomocí DLOAD nebo LOAD.
- Kompilování Warm-Overlay paketu vyžaduje trochu cviku v zacházení s kompilerem. Overlay pakety jsou většinou používány zkušenými programátory.

Při chybovém kompilování může kompiler vydat následující varování:

ILLEGAL OVERLAY:
Overlay paket byl nesprávně kompilován, např. nebyla dimensována všechna pole ve start programu. Nebylo-li vydáno toto hlášení, není to ještě záruka, že použití kompileru proběhlo správně.

LOAD ONLY IN OVERLAY:
Při použití povelů DLOAD a LOAD musí být program také jako Overlay kompilován.

6.2 Runtime-modul

Runtime-modul obsahuje všechny rutiny, nutné pro provoz zkompilovaného programu. Tyto rutiny obsahuje každý zkompilovaný program. Má-li být ušetřeno místo na disketě, pak můžete vygenerování Runtime-modulu vypnout (bod menu "G"). Při po sobě následujícím natahování více programů potřebuje jen první Runtime-modul. Ostatní musí být nataženy absolutně s BLOAD "název", BØ. Jeden jednotlivý Runtime-modul dostanete, když si zkompilujete následující program:

```
1Ø REM
```

Runtime-modul a program mohou být nyní jednotlivě absolutně nataženy. Program může běžet bez Runtime-modulu pouze tehdy, když bude natažen absolutně, a k tomu se již nalézá v paměti Runtime-modul. Start programu bez Runtime-modulu je možný pomocí programu "START" (viz 4.1Ø). Kompiler generuje program bez Runtime-modulu vždy tak, že by bylo možné jej také natahovat a odstartovat povelem BOOT "název", ale na základě chyby v basicovém překladači C-128 to bohužel není možné.

Kompiler umí generovat dva druhy Runtime-modulu. Druhý způsob vzniká jen tehdy, když se vypne při kompilerovém nastavení aritmetický modul. Automatické vypnutí pomocí kompileru nemá žádný vliv na modul. Oba druhy Runtime-modulu nesmí být zaměněny.

7. RYCHLOST

7.1 Optimalizační stupně

Kompiler používá dva optimalizační stupně, mezi kterými lze volit pomocí bodu "O" ve vývojovém paketu. Pod bodem "1" v hlavním menu bude pak ještě jednou ukázána zvolená optimalizace.

Optimalizace I.:

Při volbě tohoto optimalizačního stupně budou provedeny všechny možné optimalizace a změny programu jen tehdy, když je jisté, že v průběhu programu se již nic nebude měnit. Optimalizační stupneň 1 je tím plně kompatibilní k basicovému překladači. Výpočet s integer proměnnými bude prováděn jen tehdy jako celočíselné operace, pokud je jisté, že výsledek je celé číslo (v integer rozsahu). To je případ u většiny basicových operací.

Optimalizace II.:

Tento optimalizační stupeň obsahuje několik podstatných rozdílů ke stupni 1 a k basicovému překladači:

- Všechny proměnné, s výjimkou stringových proměnných, budou zavedeny jako integer proměnné, tzn. že kompiler předpokládá, že za každou proměnnou stojí znak "%". Zkompilovaný program pracuje rychleji, ovšem kompiler rozlišuje dále např. mezi I a I% - obě proměnné obsahují pouze svůj typ dat. Pole s proměnnými bude naproti tomu odstupňováno se správnými datovými typy.
- Dělení dvou integer-proměnných nebo celých čísel bude u optimalizačního stupně 1 vždy prováděno dělením s PDC. Stupeň 2 provádí dělení jako celočíselnou operaci a nedbá na místa za deset. čárkou. V málo případech to vede k rozdílu s překladačem. Většinou se to nestává, protože při operacích s integer proměnnými není dotaz na výsledek s PDC. Integer dělení probíhá podstatně rychleji než s PDC, obzvláště dělení dvěma.

- Funkce INT u stupně 2 nevyhlíží jako odtržení míst za desetinnou čárkou, nýbrž jako přetransformování na typ integer. Zpracování odpovídajících hodnot v nějakém vzorci probíhá s integer operacemi. Rozdíl oproti překladači spočívá v tom, že funkce INT není použitelná mimo rozsah celých čísel -32 768 až 32 768.

Optimalizační stupeň 2 má opodstatněné použití v následujících případech:

- v programech, u kterých se upřednostňuje rychlost před kompatibilitou k překladači
- v programech, u kterých je od začátku jasné, že všechny proměnné budou typu integer
- v programech, u kterých nebylo přihlíženo k použití integer proměnných a u nichž to má kompiler nyní napravit. V poněkud méně efektivní formě to probíhá také u stupně 1 při optimalizaci průběhového času.

Následující příklad ukazuje typické použití optimalizačního stupně 2:

```
10   A=INT (RND(1)*1000)
```

Proměnná A slouží k uložení celého čísla a přesto je použito PDC. Minimálně tento řádek v odpovídajícím programu může být kompilován optimalizačním stupněm 2. Programy, u nichž je nasazen optimalizační stupeň 2 jen proto, aby přenechaly převedení do integer-proměnných kompileru, potřebují většinou několik málo proměnných s PDC. Jestliže optimalizační stupeň 2 toto nepozná, je potřebné tyto proměnné kompileru zadat pomocí kompilerového příkazu.

Příkaz: REM @ R = proměnná, proměnná, ...

Vyjmenované proměnné budou v optimalizačním stupni 2 vzaty jako proměnné s PDC.

(Pomůcka I= integer R = real)

Příklad:
```
10   FOR I=1 TO 1000
20   A=SQR(I):PRINT A;
30   NEXT
```

Má-li být tento program kompilován s optimalizací 2 (OPTIMIZER II), pak je třeba zavést řádek

```
5   REM @ R=A
```

Optimalizační stupně mohou být měněny také uvnitř programu

```
REM @ 01   ; přepíná na stupeň 1
REM @ 02   ; přepíná na stupeň 2
```

Přepnutí optimalizačních stupňů má vliv jen na ten typ dat proměnných, které před přepnutím ještě nebyly použity. Optimalizační stupeň 2 nemá žádný vliv na typ dat v polích. Pole, která slouží pouze k uložení celých čísel, musí být vždy označena znakem "%"; basicový překladač pak šetří místo v paměti. Vyzkoušet si to můžete jen s překladačem. Po zkompilování je místo zabrané v paměti pro pole tak jako tak podstatně větší.

Pozor! Optimalizační stupeň 2 používejte jen u programů, které jste si sami napsali a u kterých znáte způsob činnosti programu a použitý typ dat. To platí také pro všechny další dodatečné možnosti kompileru oproti překladači (např. kompilerové příkazy).

7.2 P-kódy (Speed code) a strojový jazyk

Velmi dlouhé programy se většinou nedají úplně přeložit do strojového jazyka, protože přeložený program do strojového kódu je vždy delší než originál. Ve velkých programech jsou přirozeně také programové části, které je nutné převést do strojového jazyka. BASIC 128 umožňuje během kompilování přepínat dva druhy generovaných kódů:

```
REM @ M
```

Tento příkaz říká kompileru, aby všechny následující progra-

mové řádky přeložil do strojového kódu. Bude-li programový řádek s tímto příkazem proběhnut, pak bude příkaz odstartován.

REM @ P

Po tomto příkazu budou následující řádky přeloženy opět do P-kódů. Bude-li proběhnut řádek s tímto kompilerovým příkazem, bude odstartován překladač P-kódu a s ním budou přeloženy následující povely v programu.

Přepnutí, které nemá za následek nový stav (např. dva stejné příkazy za sebou) bude kompilerem ignorováno. Oba tyto kompilerové příkazy byste měli používat jen tehdy, když je vám jejich způsob činnosti úplně jasný. Při použití těchto povelů platí totiž následující omezení:

- mikroprocesor Commodore C-128 umí pracovat jen se strojovými kódy a nikoliv s P-kódy
- překladač P-kódu umí přeložit jen P-kódy a nikoliv strojové kódy.

Měla-li být provedena změna a kompilerový příkaz změny nebyl proběhnut, pak v žádném případě nebude program dále správně fungovat. Proto nesmí dojít k programovému skoku z jedné části do druhé, kde platí jiné kódy. Programové skoky provádí povely GOTO, GOSUB, RETURN, IF, ELSE, LOOP, EXIT, NEXT a RESUME. Dále příkaz COLLISION vede ke skoku na nepředvídané místo. Pomocí omezení strukturovým značením (kap. 4.8) si můžete cílový rozsah omezit. Příkaz TRAP vede ke skoku z chybového místa do rutiny k obsloužení chyby. Zde je druh kódu chybového místa nedůležitý, pouze TRAP-povel musí mít ten samý druh kódu jako rutina k obsloužení chyby. Dále představuje skok příkaz RESUME, u nějž je třeba dát pozor na cílový kód, což je jednoduché obzvláště u "RESUME číslo řádku". Kromě toho jsou RESUME NEXT a RESUME prováděny většinou jen pro známá programová místa.

Přepínání kódových generátorů může u programů s bloko-

vým strukturováním následovat bez nebezpečí na začátku a na konci bloku. Programový blok je význačný tím, že se do něj buďto skáče, nebo se z něj vyskakuje. Na začátku bude odstartován a na jeho konci opět opuštěn. Velké programy jsou většinou strukturovány do bloků a částečné přepnutí na strojní jazyk je tedy možné. U programů, které ani nejsou blokově strukturovány, ani nepoužívají jinou známou strukturu, by mělo být upuštěno od přepínání kódových generátorů uvnitř programu.

Velmi jednoduché je přepínání kódových generátorů u podprogramů. Podprogram může mít např. následující všeobecnou formu:

```
1000   REM @ M
 :      :
1980   REM @ P
1990   RETURN
```

Řádky od 1001 do 1979 budou v tomto případě přeloženy kompilerem do strojového jazyka. Podprogram může být volán pomocí GOSUB 1000 a po proběhnutí řádků 1980 a 1990 opět opuštěn, v žádném případě s GOTO 1980. U strukturovaných programů je to také tento případ. Má-li dojít uvnitř podprogramu k zavolání jiného podprogramu, pak musí být nejdříve kódový generátor opět přepnut. V našem příkladě by to vyhlíželo následovně:

```
1500   REM @ P
1510   GOSUB 5000
1520   REM @ M
```

Zde nehraje přirozeně žádnou roli, jaký typ kódu obsahuje volaný podprogram, pokud nejméně první řádek (s kompilerovým příkazem) bude přeložen v P-kódu.

Většina časově kritických podprogramů nevolá další podprogramy.

7.3 Integer-rozsah u povelu POKE

Několik málo možností C-128 může být použito v BASICu s pomocí povelů PEEK a POKE. Tyto povely slouží většinou k předávání dat do assemblerového programu a tím mají možnost rychlého zpracování.

Použití integer proměnné je v kombinaci s POKE přirozeně účelné jen tehdy, je-li adresa paměti znázorněna celým číslem. To je přesně 65 536 různých adres (Ø + 65 535) a integer proměnná může tedy obsahovat 65 536 různých číselných hodnot (-32 768 do 32 768). Bohužel nejsou odpovídající číselné rozsahy identické. Adresy v paměti nad 32 768 musí být tedy použity s proměnnou s PDC. Ve spojení s povelem POKE a několika dalšími povely má kompiler možnost integer-operace provádět nad hodnoty 32 767, ovšem když se upustí od negativních čísel. Uživatel kompileru získá větší rychlost průběhu programu.

```
 5   REM @ I = I
1Ø   FOR I= 1Ø24 TO 2Ø23
2Ø   POKE I,65
3Ø   POKE I + 54272,Ø
4Ø   NEXT
```

Rychleji poběží program po zavedení této změny

```
 6   OF%=3Ø ØØØ
3Ø   POKE I+OF%+24272,Ø
```

Nyní budou zpracovány integer-hodnoty a z jednoho sečítání s PDC jsou dvě sečítání integer. Tento trik by měl být používán jen v extrémně časově kritickém místě programu.

8. ADRESY PAMĚTI A STROJOVÝ KÓD

Tuto kapitolu byste si měli bezpodmínečně přečíst, jestliže vaše basicové programy mají pracovat společně s assemblerovými programy nebo když používáte ve vašem programu povel POKE.

8.1 Obsazení paměti

Stisknutím klávesy <E> ve vývojovém paketu se dostanete do dalšího menu, ve kterém pak můžete skoro libovolně manipulovat s částí paměti, kterou kompiler používá. Stiskněte odpovídající klávesu čísla <1> až <5> a pak zadejte adresu.

Kompiler vydá hodnoty o obsazení paměti kompilovaným programem a přednastavené a ještě nezměněné adresy v memory-menu. Hodnoty pro start Runtime modulu a Aritmetického modulu naleznete v programu "START" na disketě BASIC 128:

Ø až Start-Bank Ø:	systémová paměť
Start-Bank Ø až Data-Code-Start:	paměť pro jednotlivé proměnné a pole
Data-Code-Start až Top-Bank Ø:	programcode
Runtime-Modul resp. Arithmetik-modulstart až Speicherende:	Laufzeitsystem
$ Ø4ØØ až Start-Bank 1:	zásobník pro stringy-deskriptory v Banku 1
Start-Bank 1 až String-Ende:	volný rozsah paměti pro stringy v Banku 1
String-Ende až Top-Bank 1:	paměť pro pole a jednotlivé proměnné v Banku 1
Top-Bank 1 až Speicherende:	Laufzeitsystem v Banku 1

Laufzeitsystem v Banku 1

Pomocí změny přednastavení pro hodnotu Top dolů a pro hodnotu Start nahoru je možné vznikající paměťový rozsah rezervovat pro libovolné použití. Kompiler nebude tento rozsah

pak už používat a je možné zde například uložit assemblerové programy nebo novou znakovou sadu. Přitom dejte vždy pozor, aby ani Laufzeitsystem ani eventuálně Aritmetický modul nebyl posunut v paměti. Změnou nastavení Code-Start dolů pod kompilerem udaná DATA-Code-Start může být celý Program-Code posazen dolů a tím bude odpovídající paměťový rozsah mezi programem a Laufzeitsystemem (resp. Aritmetickým modulem) volný.

Integer-proměnné:	2 byty	LOW byt, HIGH byt
Proměnné s PDC:	5 bytů	
		první byt je exponent; 2.-5. byt Mantisa
	nebo:	první byt je Ø; 3. a 4. byt je integer-znázornění

K přenosu proměnné s PDC do formátu překladače pro assemblerový program může být např. použito funkce USR.

Stringové proměnné:	3 byty	
	délka, adresa stringu LOWbyt, adresa HGH byt	
Stringy:	2 byty	Trailer + 1 byt na znak

Assemblerový program může obsah proměnných měnit. Adresy proměnných dostanete pomocí tabulky symbolů. U stringů není možné ani stav stringu ani jeho délku měnit assemblerovým programem. To umí jen zkompilovaný BASICový program nebo u nekompilovaných programů překladač.

Kompilerem bude automatidky rezervována grafická paměť od $ 1ØØØ až $ 4ØØØ, pokud se v programu objeví nějaký povel, který tento rozsah používá. V tomto případě nesmí být hodnota pro Start-Bank Ø vůbec měněna, nebo nejvýše na hodnotu nad 16 383.

8.2 Paměťové adresy

Při použití povelů POKE a PEEK je třeba dát pozor, že zkompilovaný program některé adresové buňky používá jinak

jako překladač. Pro použitelnost povelu POKE platí:

- v Banku 2 + 15 se žádné změny neprovádí
- všechna paměťová místa od $ ØØØØ do $ 13ØØ v Banku Ø, která jsou používána operačním systémem nebo povely překladače, si ponechají svoji funkci (zásobník, klávesnice, stisknuté klávesy, I/O-vektory, obrazovková paměť, grafické hodnoty atd.). Dříve vůbec nepoužívané paměťové buňky v tomto rozsahu budou nyní použity Laufzeit systémem.

Změny jsou možné pouze u následujících paměťových buněk: rezervovaný rozsah od $ 13ØØ do $ 1CØØ je používán kompilovaným programem. Má-li být tento rozsah obsazen jinak, pak může být uvolněn pomocí kompilerového příkazu:

REM @ S adresa

K uvolnění celého zásobníku (buffer) slouží následující povel

REM @ S 7168 nebo při použití grafiky REM @ S 16 384.

Paměťové buňky, které používá BASICový překladač k svojí činnosti, ztrácejí částečně svůj význam. To je přirozený důsledek kompilování. Zachovány zůstávají jen ty paměťové buňky, které se normálním způsobem nechají použít s povelem POKE. K nim patří:

- registr PDC
- náhodné číslo
- CHRGET rutina a další rutiny pro přístup do paměti
- statusové slovo ST
- hodnota RND
- vektor na start programu (43 - 44)
- vektor na konec programu (45 - 46)
- vektor na String-stapel (53 - 54, 57 - 6Ø)
- vektor na následující DATA-Element (67 - 68)
- popis DS $
- Hires flag
- Modus flag

Vektor na zásobník (Stack pointer) $ 7D, $ 7E je používán kompilerem poněkud jiným způsobem než překladačem. Je možné číst paměťovou buňku $ 7D a podle potřeby ji opět uložit a tím kontrolovat hnízdění FOR- a GOSUB-. Ovšem přitom je třeba ještě korigovat buňku $ 82. Paměťová buňka $ 7E nesmí být měněna.

Příklad:
```
1Ø  A% = PEEK (125) : REM  uložit stav zásobníku
2Ø  FOR I=1 TO 1ØØØ
3Ø  POKE 125,A% : POKE 13Ø,A% : REM  stav zásobníku
                                     opět sestavit
4Ø  NEXT : REM  vytvoří NEXT bez FOR
```

Při použití adresových buněk (2 byty) je třeba dát pozor, že do těchto buněk možná budou uloženy jiné adresy než při použití překladače. To obzvláště platí pro vedení (spravování) paměti (45 - 58) a pro obsloužení chyby (768 - 769).

Vesměs jsou pozorované rozdíly mezi překladačem a kompilerem při obsazování paměťových adres tak nepatrné, že nemají prakticky skoro žádný význam.

8.3 Speciální povely

SYS: povel SYS basicu 7.Ø obsahuje stejné parametry jako povel RREG. Z tohoto důvodu je tento povel oproti basicu 2.Ø jen omezeně použitelný pro basicové rozšíření. Toto platí samozřejmě také pro kompiler.

BANK: Při startu nějakého programu je automaticky proveden povel BANK 15, protože se jedná o standardní nastavení překladače. Má-li být použit jiný BANK, pak musí být nastaven uvnitř programu.

STOP: Tento povel způsobí ukončení kompilovaného programu. Podobně jako při vydání chybového hlášení, nemůže být použit povel CONT.

POINTER: Funkce překladače POINTER udává vždy nějakou adresu v BANKu 1. Kompiler ukládá částečně proměnné také do

BANKu Ø. Na základě toho nabízí kompiler navíc možnost zjistit paměťový BANK pomocí PEEK (2). Program, který správně poběží, může například vyhlížet následovně:

```
1Ø  POKE 2,1 : REM BANK PRO PŘEKLADAČ
2Ø  A = POINTER (X)
3Ø  BANK PEEK(2) : REM BANK  pro kompiler a překladač
4Ø  E = PEEK(A) : REM  hodnotu vyčíst
```

Při nesprávném použití funkce POINTER vydá kompiler ve většině případů varování "POINTER WITHOUT BANK". Kompiler ukládá proměnnou vždy na pevnou adresu. Nemůže se tedy nic stát, jestliže funkce POINTER při běhu programu pošle více různých adres pro tutéž proměnnou, jak se to děje u překladače.

Má-li být uloženo datové pole na základě kompatibility bezpodmínečně do BANKu 1, pak to může sdělit kompileru pomocí následujícího příkazu, přímo po povelu DIM

REM @ B

8.4 Tabulka symbolů

BASIC 128 má možnost uložit a natáhnout tabulku symbolů. V této tabulce jsou uvedeny všechny proměnné programu a k nim vyjmenovány adresy, na kterých jsou tyto proměnné uloženy. Tabulka symbolů je uložena v takové formě, jak ji kompiler interně zpracoval.

Uložení tabulky symbolů

K uložení tabulky symbolů je třeba sdělit kompileru pouze jméno (název) tabulky. To se provede pomocí bodu menu "C". Název "OVERLAY" je rezervován pro Overlay tabulku symbolů. Uložení tabulky symbolů následuje po kompilování.

Natažení tabulky symbolů

Pomocí bodu menu "B" je možné zadat název tabulky symbolů, která má být natažena. Natažení tabulky se děje před

kompilováním. Všechny vyjmenované proměnné a jejich adresy v paměti, uvedené v tabulce, budou převzaty. Toto je užitečné, jestliže má více programů přistupovat do společných proměnných, aby bylo možné například použít ty samé assemblerové programy. Při kompilování s Overlay-pakety je kompiler odtud používá.

Zpracování tabulky symbolů

Na disketě BASIC 128 se nachází program SYMBOL. Po jeho natažení a odstartování se program ptá na název souboru tabulky symbolů. Po zadání názvu můžete volit mezi dvěma možnostmi.

Stisknutím klávesy <1> bude program přinucen přeměnit tabulku do formátu assembleru PROFI-ASS. Název přeformulované tabulky symbolů je identický s tím, který vytvořil kompiler. Přesto nebude stará tabulka smazána, protože kompiler připojí předponu "S-" před název tabulky. Toho si nemusíte všímat. Assembler PROFI-ASS může natahovat tabulky symbolů s Pseudo-opcody ".LST" (viz manuál PROFI-ASS). Název proměnné může nyní assemblerový program použít, jestliže byla v programu definována. Tímto je možné, že assemblerový program může přistupovat na proměnné zkompilovaného programu. K rozlišení jednotlivých typů dat mezi sebou a symbolů právě platného assemblerového programu přeformátuje program SYMBOL jména proměnných. V následujícím příkladu je "na" pro první dva byty názvu proměnné

Jednotlivé proměnné: na - na
na% - naIN
na$ - naST

pole: na - ARna
na% - ARnaIN
na$ - ARnaST

Znázornění je nezávislé na optimalizačním stupni. Je přímo odvozeno ze jména proměnné. Následující výsek z nějakého assemblerového programu vyměňuje 2 BASIC-integer proměnné:

```
 1ØØ   LST 8,2,"název,S,R"
 11Ø  ....
1ØØØ  LDA AIN
1Ø1Ø  LDX BIN
1Ø2Ø  STX AIN
1Ø3Ø  STA BIN
1Ø4Ø  LDA AIN+1
1Ø5Ø  LDX BIN+1
1Ø6Ø  STX AIN+1
1Ø7Ø  STA BIN+1
1Ø8Ø  ....
```

Toto přibližně odpovídá následujícímu BASIC-programu:

```
 1ØØ  H%=A%:A% = B%:B% = H%
 11Ø  ....
```

Assembler PROFI-ASS 64 běží sice jen v módu C-64 počítače C-128, zassemblerované programy mohou ale běžet samozřejmě také v módu C-128 a tím být i použity zkompilovaným programem. Nepoužíváte-li žádný Assembler, můžete přesto zpracovat tabulku symbolů. Stiskněte po startu programu SYMBOL a po zadání názvu souboru klávesu <2 >a dostanete listing. Program se nyní ptá na číslo zařízení, na kterém má být listing vydán. Smysl dávají následující adresy zařízení:

Ø = Obrazovka (monitor)
4 = Tiskárna
8 = Sekvenční soubor na disku

Program nyní vydá jméno (název) proměnné následované adresou v paměti a BANKem paměti.

Program SYMBOL nesmí být použit na symbol-soubor OVERLAY. K přeformování OVERLAY-souboru nechejte kompiler natáhnout soubor a uložte jej do jiného souboru, k čemuž je potřeba následující program zkompilovat v Nicht-Overlay-Modu s odpovídajícím kompilerovým nastavením

```
1Ø  REM
```

Program SYMBOL je pouze podnět (impuls) ke zpracování tabulky symbolů a může být libovolně rozšiřován, zde je v nekompilované formě. V úvahu by přicházel např. editor tabulky symbolů nebo přizpůsobení na jiný assembler.

9. PŘEHLED MOŽNOSTÍ BASIC-128

Tato kapitola nabízí pouze krátký přehled použití kompileru. Většina možností kompileru je detailněji popsána v příslušných kapitolách tohoto manuálu.

Start kompileru

Z hlavního menu klávesou <1> nebo <RETURN>. Menu můžete vždy opustit s <RETURN>, pokud kompiler nečeká na <RETURN> jako ukončující zadání.

Natažení kompilerových parametrů

Pomocí klávesy <2> v hlavním menu si můžete natáhnout všechna kompilerová nastavení z diskety a pak se dostat automaticky k bodu menu "3". Pomocí tohoto bodu menu můžete natahovat jen soubory, které byly uloženy pod názvem "B128" (viz bod "I").

Nastavení kompilerových parametrů

Při startu kompileru pomocí klávesy <1> nebo <RETURN> obsahuje kompiler zápis hodnot a příkazů, které potřebuje ke kompilování. K zobrazení tohoto zápisu na obrazovce stiskněte v hlavním menu klávesu <3>. Nyní můžete zápis změnit a pak se dostanete pomocí <RETURN> zpět do hlavního menu. Jednotlivé volby (body) zápisu jsou označeny písmeny a je možné je zvolit pomocí odpovídajících kláves.

"A" - Kódový generátor

Kompiler může volitelným způsobem vytvářet P-kódy, strojový jazyk (65Ø2/651Ø/85Ø2) nebo dokonce žádné kódy. Strojový jazyk má pak před názvem "M-", jiné programy "P-". Kompiler je přednastaven na generování P-kódů.

"B" - natažení tabulky symbolů

Pomocí této volby může kompiler natáhnout soupis proměnných, které byly uloženy při kompilování nějakého jiného programu. Toto má za následek, že oba programy pak používají ty samé adresy pro odpovídající proměnné (Overlay-pakety a podobná použití).

"C" - uložení tabulky symbolů

Uložené tabulky symbolů mohou být nataženy při kompilování jiných programů nebo použity assemblerem k obsazení adres návěští. To je užitečné, když BASIC-program pomocí povelu SYS volá podprogramy ve strojovém jazyku.

"D" - vygenerování výpisu řádků

Po zkompilování nějakého programu uloží kompiler, je-li mu to přikázáno, zápis řádků programu. Tento zápis může být natažen pomocí DLOAD "Z-název programu" a s LLIST vylistován. Každý BASIC-řádek (pravá strana) je přiřazen paměťové adrese (levá strana) (pro chybové hlášení nebo start nějakého programového dílu (části) pomocí SYS).

"E" - rozdělení paměti

Pomocí tohoto bodu menu si můžete změnit rozdělení paměti kompileru. Objeví se další podmenu s následujícími body:

"1" - Start paměti proměnných v BANKu Ø

"2" - Konec paměti programu a paměti dat v BANKu Ø

"3" - Start paměti dat v BANKu 1, spodní hranice pro stringy

"4" - Konec paměti dat v BANKu 1

Změnou hodnot pro tato 4 nastavení paměti může být rezervováno místo např. pro program ve strojovém kódu

"5" - Maximální start - hodnota pro kompilerem generované kódy

"6" - Pomocí této volby mohou být v kompileru vypnuty rychlé rutiny s PDC. Tím bude program asi o 3 kbyty kratší. Kompiler odpojí rutiny automaticky, jestliže jsou nepotřebné.

"F" - Fast modus kompileru

Při zapnutém módu 4Ø znaků na řádku obrazovky pracuje kompiler ve SLOW módu, aby byly čitelné informace na obrazovce. Můžete také přepnout do FAST módu a v tomto případě pak kompiler zapíná obrazovku pouze pro důležitá hlášení.

"G" - generování Runtime-modulu

Můžete si odpojit automatické připojení Runtime-modulu (asi 9 kbytů) a modul a program potom natahovat jednotlivě. Toto šetří při Overlay-paketech místo na disketě a natahovací čas.

"H" - Nastavení BASIC-rozšíření

Má-li být kompilerem přeložen program, který obsahuje basicová rozšíření, pak musí být zvolen tento bod a dále musí být nastaveno takové rozdělení paměti kompileru, aby paměť pro rozšíření byla volná. Nahlédněte do manuálu vašeho basicového rozšíření, zdali a jak dalece má být kompiler rozšířen.

"I" - Natažení a uložení všech kompilerových nastavení

Tímto bodem menu můžete natáhnout a uložit kompilerové nastavení. Kompiler vsadí před název souboru vždy "E-". K tomuto není potřeba přihlížet při zadávání názvu souboru (probíhá automaticky).

"J" - Řádky a povelová struktura

Ke zkompilování několika málo povelů Basicu 7.Ø (např. RESUME NEXT, TRON, COLLISION) musí kompiler zavést řádkovou a povelovou strukturu programu, který je kompilován do kom-

pilovaného programu, jinak tyto povely nebudou správně provedeny. Tento bod menu by měl být používán jen u krátkých programů, a u velkých programů by se měly používat kompilerové příkazy.

"K" - Chybový přerušovací řádek

Tento bod menu umožňuje automatické zavedení povelu TRAP do programu.

"L" - Overlay-Pass

Pomocí této volby může být nastaven Overlay-Pass. Normálně by to mělo být ale provedeno volbou bodu "4" v hlavním menu.

"M" - Povelový kanál k disku

Pomocí této volby můžete vysílat povely k disku. Stačí uvádět pouze první písmeno požadovaného povelu DOS. Kompiler se pak ptá na parametry tohoto povelu. Kromě povelů HEADER, SCRATCH, RENAME, COLLECT a DIRECTORY je zde ještě povel OTHERS, který může být odeslán k disku s libovolným stringem přes povelový kanál. Má-li být provedeno formátování (HEADER) s ID, pak je třeba oddělit písmena ID od názvu diskety čárkou.

"N" - Seznam diskety - directory

Tento bod menu slouží k přečtení directory diskety.

"O" - Optimalizační stupeň

Optimalizační stupeň 1

Tento stupeň je plně kompatibilní k basicovému překladači V 7.Ø. Všechny provedené optimalizace programu nemění v žádném případě průběh nebo chování programu, nýbrž slouží jen k dosažení vyšší rychlosti. Aby ve stupni 1 byla dosažena vyšší rychlost při počítání s celými čísly (integer), musí proměnné pracovat pouze s celými čísly a také tak musí být označeny. V Commodore basicu se toto děje pomocí připojení znaku "procento" (%) za název každé proměnné.

Optimalizační stupeň 2

Stupeň 2 se liší od stupně 1 tím, že všechny proměnné budou vyhlížet jako integer-proměnné, a to i u těch proměnných, u kterých to basicový překladač nezná. Má-li (nebo musí-li) být přesto použita některá proměnná s PDC, pak musí být toto označeno odpovídajícím kompilerovým příkazem. Dále pak vedou optimalizace stupněm 2 s integer dělením a integer převodem (INT) k tomu, že nejsou ve všech případech kompatibilní k překladači.

Kompilování Overlay-programů

Programy, které jsou nataženy a odstartovány pomocí DLOAD nebo LOAD, mohou většinou používat ty samé proměnné, resp. jejich obsah (z předchozího použití jiným programem) - tzv. Warm-Overlay. Při natažení pomocí RUN budou naproti tomu všechny proměnné smazány; takové programy mohou být normálně kompilovány (Kalt-Overlay). Kompilování Overlay paketů, které používají povely LOAD nebo DLOAD, se provádí dvěma průchody.

Průchod 1:

Kompiluje všechny programy Overlay-paketů. Stiskněte hned po aktivování kompileru <4> (Overlay) a pak <1> (průběh 1). Všechna kompilerová nastavení musí být převzata v Pass 1. Kompiler odpojí automaticky Runtime-modul. Jen při kompilování prvního programu bude automaticky zapojen, jelikož programy bez Runtime-modulu nemohou být odstartovány v přímém módu (direct mode). První program paketu musí být tzv. startovací program, který se startuje s RUN. Při průchodu 1 není vytvářen žádný program, nýbrž jen tabulka. Z těchto důvodů je tento průchod relativně rychle ukončen.

Průchod 2:

Kompiluje všechny programy ještě jednou, přičemž tentokrát stiskněte <4> a <2> (průchod 2). Kompiler nyní generuje požadovaný program. Jestliže nepostačuje místo na disketě, pak můžete také originál programy po jejich zkompilování

smazat (volba "M" v podmenu).

Aby bylo možné Overlay-paket odstartovat, musí být název zkompilovaných programů tak změněn, aby programy mohly být nataženy.

Kompilerové příkazy

Často je požadováno mít vliv na kompilování v jeho průběhu. Toto je možné pomocí speciálně označených (REM @) příkazů uvnitř programu.

Přepnutí na generování strojového kódu

Formát: REM @ M

Kompiler začne generovat program ve strojovém kódu. Kromě toho bude do programu zaveden příkaz, který sdělí překladači P-kódů, že nyní následuje strojový program.

Zpětný návrat na P-kódy

Formát: REM @ P

Kompiler generuje opět P-kódy. Současně bude do programu zaveden povel, který aktivuje překladač P-kódů.

Zapnutí a vypnutí obsloužení chyby

Formát: REM @ E číslo řádku

Tato možnost odpovídá povelu TRAP překladače, je ale aktivní teprve po zkompilování.

Deklarování Integer-proměnných

Formát: REM @ I = proměnná, proměnná ....

Všechny vyjmenované proměnné (s PDC, tedy bez znaků %) budou kompilerem převedeny jako proměnné integer, což má za následek rychlejší průběh programu. Současně můžete zavést integer proměnné uvnitř smyčky FOR-NEXT, což si normálně basicový překladač nedá líbit. Tento povel dává smysl jen ve spojení s optimalizačním stupněm 2.

Deklarování proměnných s PDC

Formát: REM @ R = proměnná, proměnná ....

Všechny vyjmenované proměnné budou převedeny na proměnné s PDC, což je samozřejmě potřebné při použití optimalizačního stupně č. 2.

Přepínání optimalizačních stupňů

Formát: REM @ Ø číslo stupně

Optimalizační stupeň bude přepnut. Toto má vliv pouze na proměnné, které v programu až do tohoto místa nebyly osloveny.

Uvolnění rezervovaného rozsahu

Formát: REM @ S adresa

Kompiler používá automaticky rozsah od $ 13ØØ do $ 1CØØ (který není používán překladačem) k ukládání dat, což s tímto povelem může být zabráněno.

Příklad: REM @ S 7168

Tento příkaz způsobí, že kompiler uloží proměnné nahoru nad $ 1CØØ.

Volba BANKu 1

Formát: REM @ B

Bezprostředně před dimenzováním pole bude nezávisle od použité paměti uloženo do BANKu 1. Normálně jsou pole kompilerem uložena do BANKu Ø, dokud je tam místo.

Přepnutí do módu FAST

Formát: REM @ F

Kompiler se přepne do režimu FAST. Tento povel je užitečný, je-li program kompilován v módu 4Ø znaků na řádek.

Volba značení programové struktury

Formát:	REM @ O	; vypíná značení
	REM @ L	; zapíná značení řádků
	REM @ C	; zapíná značení povelů
	REM @ B nebo REM @ LC	; zapíná obě značení

K provedení několika málo povelů basicu 7.Ø potřebuje Laufzeitsystem označení skokových povelových resp. řádkových struktur programu. Tím pádem bude program ovšem pomalejší a nedosáhne maximální rychlosti. Na tomto základě je možné s těmito povely označení struktury omezit na jednotlivé programové části.

Použití optimalizačních možností BASICu 128

Ačkoliv kompilování váš program podstatně urychlí, nechá se rychlost programů ještě zvýšit, jestliže si již při psaní programů dáte pozor na to, že některé operace po zkompilování jsou prováděny obzvláště rychle. Většinou stačí dát pozor na následující pravidla k dosažení vyšší rychlosti:

- Zkušenosti, které jste učinili s basicovým překladačem co se týče relativní rychlosti jednotlivých operací, neplatí již pro zkompilované programy.
- Operace s integer proměnnými pracují podstatně rychleji než tytéž operace s proměnnými s PDC. Zkušenosti říkají, že velká část všech proměnných ve většině programů je nahraditelná proměnnými typu integer (hodnoty mezi -32 768 a +32 768 bez desetinné čárky). Integer proměnné mohou být překladači a kompileru označeny s přidáním znaku "%" za název proměnné. Dále to můžete sdělit kompileru pomocí příkazů REM @ nebo s optimalizačním stupněm 2.

  Integer proměnné jsou výlučně použity např. v následujících funkcích:

  Čítač ve smyčkách, indexy polí, POKE, PEEK, ON, WAIT, parametry souboru při OPEN, SYS, TAB, SPC, FRE, číselné parametry u funkcí se stringy, ASC, CHR$, logické operace,

všechny grafické povely, často také porovnání a *, /, +, - a u skoro všech dalších basicových povelů. Ve všech těchto případech je použití proměnných s PDC nesmyslné a pomalé.
Při použití basicového překladače to je nepostřehnutelné, protože tento všechny výpočty provádí s čísly s PDC.

- Operace se stringy jsou prováděny s kompilovanými programy jinak jako překladačem. Tím je dosahováno vyšší prováděcí rychlosti, zvláště u komplexních stringových výrazů.
- Kompiler odebírá programu mnoho práce a nepotřebuje si všímat např. vyhledávání skokových řádků u GOTO, GOSUB, překládání povelů a výrazů, ověřování syntaxe, převádění dekadických celých čísel a čísel s PDC na binární tvar, vyhledávání proměnných, přeformování a optimalizování vzorců a výrazů, přepočet operací s konstantami a stringy, zpracování paměti atd.
- Komplexní aritmetické funkce basicu 7.Ø jsou BASICem 128 podstatně urychleny (SIN, COS, TAN, ATN, EXP, LOG, SQR, ↑). Již není nutné tyto funkce odstraňovat z časově kritických částí programu.

10. ROZDÍLY MEZI BASIC 128 a BASIC 64

BASIC 128 je následné rozvinutí známého kompilátoru BASICu 64. Zde je přehled důležitých inovací:

- BASIC 128 je plně kompatibilní basicu 7.Ø
- podstatně zvýšená ø rychlost zkompilovaných programů (kap.3)
- rychlejší provoz funkcí TAN, SIN, SQR atd. (kap. 4.4)
- kódový generátor generuje kratší P-kódy a rychlejší strojový jazyk než BASIC 64
- celá paměť 128 kbytů RAM u C-128 je kompilerem využívána.

Navíc k BANKu 1 je k použití paměť v BANKu Ø (která není potřebná pro programové kódy) pro proměnné a pole (kap. 4.5)

- BASIC 128 provádí navíc ke kompilerové optimalizaci také optimalizaci časového průběhu (kap. 3.6)
- ke kompilaci některých povelů basicu 7.Ø je nutné označení programové struktury. Zde používá BASIC 128 více pružnějších možností (kap. 4.8)
- možnosti nastavení paměti jsou širší než u BASIC 64 (kap. 8.1)
- kompilerová nastavení mohou být uložena jako soubory a jako takové opět natažena (kap. 4.9)
- kompiler může být bez dalšího nového natažení odstartován (kap. 4.10)
- postup kompilování Overlay-paketů byl vylepšen a nepotřebuje být používán při Kalt-Overlay (kap. 6.1)
- basicová rozšíření nemají u Commodore 128 žádný velký význam, protože basic 7.Ø je velmi výkonný.
  Z těchto důvodů je rozšíření BASICu 128 zohledněno, je-li přizpůsobeno kompileru (kap. 5.3)
- přístup do DOS disku je velmi komfortní, s možností přečíst directory disku (kap. 9.1)
- kompiler vydává navíc k chybovým hlášením překladače varování (kap. 4.2)
- u integer smyček je dovolena hodnota STEP (kap. 4.7)

11. DALŠÍ POUŽITÍ

11.1 Vstup - Výstup

Periferie, například disk nebo tiskárna, pracují po zkompilování nějakého programu samozřejmě stejně rychle jako dříve. Při ukládání a natahování dat z disku je proto doporučováno maximálně využít rychlost disku, která je k dispozici. Před a po každém basicovém povelu, který vede na disk, vysílá operační systém jeden řídící kód. Přenos tohoto kódu vyžaduje rovněž čas. Má-li tomu být zabráněno, musí se pokud možno velká datová množství přenášet jedním povelem.

```
 9Ø   REM @ I = I
1ØØ   FOR I=1 TO 1ØØ
11Ø   PRINT # 1, CHR$ (I);
12Ø   NEXT
```

Tento programový úsek může např. vypadat takto:

```
 9Ø   REM @ I = I
1ØØ   FOR I=1 TO 1ØØ
11Ø   PRINT # 1, CHR$ (I); CHR$ (I+1);
12Ø   I = I+1 : NEXT
```

Obzvláště pomalu pracuje povel GET#, protože načítá jednotlivé byty. K odvysílání dat by měl být proto po každých 8Ø znacích dán znak konce řádku, dokud to druh dat dovoluje:

```
 9Ø   REM @ I = I
1ØØ   A$=" ":FOR I=1 TO 8Ø
11Ø   GET# 1,B$:A$ = A$+B$
12Ø   NEXT
```

U odpovídajícího datového formátu by mohl být zjednodušen následující řádek

```
1ØØ   INPUT #1,A$
```

Zde by nebylo ušetřeno jen několikanásobné vysílání řídícího kódu, nýbrž také složení (sečtení) jednoho řetězce.

Výstup dat na obrazovku je většinou dostatečně rychlý. Při výstavbě obrazovkové masky se může ovšem stát, že se setkáme s efektem známým při psaní programů. Jestliže kursor stojí na spodním okraji, pak při pohybu kursoru dolů se posune celá obrazovka nahoru. Tomu se dá zabránit tím, že se nezavede nový řádek, dokud nebude starý obraz přetištěn.

11.2 Jemná grafika na 8Ø-ti znakové obrazovce

Kreslení grafiky 64Ø * 2ØØ bodů na 8Ø-ti znakové obrazovce trvá s basicovým překladačem velmi dlouho. Obzvláště u jemné grafiky se ukazuje, jaká je výhoda kompileru proti překladači. Následující rutiny používají skoro výlučně integer operace.

```
  5  REM Ø2
 1Ø  BANK 15:WR=52684:RE=52698:FOR I=Ø TO
     7:MA%(I)=2↑(7-I):NEXT
 2Ø  SYS WR,128,25:SYS WR,7*16+1,26
 3Ø  SYS WR,Ø,18:SYS WR,Ø,19
 35  FOR I=Ø TO 64:SYS WR,Ø,31:SYS WR,Ø,3Ø:NEXT
 4Ø  GOTO 1ØØØØ
1ØØ  REM DRAW X,Y,X2,Y2
1Ø5  X1=X:Y1=Y:DX=ABS(X2-X1):DY=ABS(Y2-Y1)
11Ø  IF DX=Ø THEN SX=Ø:ELSE SX=INT((X2-X1)/DX)
12Ø  IF DY=Ø THEN SY=Ø:ELSE SY=INT((Y2-Y1)/DY)
13Ø  IF DY>DX THEN BEGIN
14Ø  XS=Ø:DH=DY:DL=DX
15Ø  BEND:ELSE BEGIN
16Ø  XS=-1:DH=DX:DL=DY
17Ø  BEND
18Ø  X=X1:Y=Y1:C=INT(DH/2):CP=DX+DY:GOSUB 1ØØØ
19Ø  DO:C=C-DL:IFC>Ø THEN BEGIN
```

```
  2ØØ  C=C+DH:Y=Y+SY:X=X+SX:CP=CP-2
  21Ø  BEND:ELSE BEGIN
  22Ø  IF XS THEN X=X+SX:ELSE Y=Y+SY
  23Ø  CP=CP-1:BEND
  24Ø  GOSUB 1ØØØ
  25Ø  LOOP UNTIL CP<=Ø
  26Ø  RETURN
 1ØØØ  REM PLOT X,Y
 1ØØ1  REM M
 1ØØ5  AD=INT(X)/8+INT(Y)*8Ø:H=AD/256:L=AD AND 255
 1Ø1Ø  SYS WR,H,18:SYS WR,L,19:SYS RE,Ø,31:RREG BY
 1Ø2Ø  SYS WR,H,18:SYS WR,L,19:SYS WR,BY OR MA % (X AND 7),31
 1Ø3Ø  REM P
 1Ø4Ø  RETURN
 2ØØØ  REM CIRCLE X,Y,R
 2ØØ1  REM Ol
 2Ø1Ø  S=R*R:X1=X:Y1=Y
 2Ø3Ø  FOR X2=Ø TO COS(.786)*R+1
 2Ø4Ø  Y2=SQR (ABS(X2*X2-S))
 2Ø5Ø  X=X2+X1:Y=Y2+Y1:GOSUB 1ØØØ
 2Ø6Ø  Y=Y1-Y2:GOSUB 1ØØØ
 2Ø7Ø  X=Y2+X1:Y=X2+Y1:GOSUB 1ØØØ
 2Ø8Ø  X=X1-Y2:GOSUB 1ØØØ
 2Ø85  X2=-X2:IF X2<Ø THEN 2Ø5Ø
 2Ø9Ø  NEXT:RETURN
1ØØØØ  REM DEMO
1ØØØ5  FOR A=5Ø TO 6ØØ STEP 3Ø
1ØØ1Ø  X=A:Y=1Ø:X2=6ØØ-A:Y2=19Ø:GOSUB 1ØØ
1ØØ2Ø  NEXT
1ØØ3Ø  FOR W=5Ø TO 8Ø STEP 1Ø
1ØØ4Ø  X=2ØØ:Y=9Ø:R=W:GOSUB 2ØØØ
1ØØ5Ø  NEXT
1ØØ6Ø  GOTO 1ØØ6Ø
```

Tento program smaže celou paměť 8Ø-ti znakového video-čipu VDC a tím také znakovou sadu. Tato může být jednoduše

obnovena pomocí klávesy ASCII/DIN.

Program používá následující grafické rutiny:

GOSUB 1ØØ - nakreslí čáru z X,Y do X2,Y2
GOSUB 1ØØØ - nakreslí bod o souřadnicích X,Y
GOSUB 2ØØØ - nakreslí kružnici se středem X,Y a poloměrem R

U programu od řádku 1Ø ØØØ si můžete samozřejmě napsat vlastní grafický program. Program plně využívá možnosti C-128. Po odstranění kompilerových příkazů v řádcích 1ØØ1 a 1Ø3Ø může být dokonce přeložen úplně do strojového jazyka. K plnému využití grafických schopností C-128 jsou k použití rychlé rutiny, které mohou být kdykoliv zabudovány.

Pomocí vysoké rychlosti zkompilovaných programů stojí před vámi s programovacím jazykem BASIC nyní takové možnosti, o jakých se vám dříve ani nesnilo.

Praha 15. 11. 1987